Approximate Planning in Large POMDPs via Reusable Trajectories

Michael Kearns [*] AT&T Labs

Yishay Mansour [†] AT&T Labs

Andrew Y. Ng UC Berkeley

May 5, 1999

Abstract

We consider the problem of reliably choosing a near-best strategy from a restricted class of strategies Π in a partially observable Markov decision process (POMDP). In particular, we are interested in what might be considered the *sample complexity* — that is, the amount of data or experience we must generate in the POMDP in order to choose a good strategy. We assume we are given the ability to *simulate* the behavior of the POMDP, and we provide methods for generating simulated experience sufficient to accurately approximate the expected return of any strategy in Π.

We prove upper bounds on the amount of simulated experience our methods must generate in order to achieve such uniform approximation. These bounds have no dependence on the size or complexity of the underlying POMDP, depend only linearly on the complexity of the restricted strategy class Π, and depend exponentially on the horizon time. The main challenge in obtaining such bounds lies in generating trajectories that can be *reused*, in the sense that they simultaneously provide estimates of the return of *many* strategies in the class.

Our methods can be viewed as generating a "small" set of trajectories that provide an accurate estimate of the value of any strategy in the class. They can thus be easily used with many standard approaches to search in strategy space, such as gradient descent and local search. For such algorithms, the exponential dependence on the horizon time can be replaced by a factor linear in the number of steps of search to be performed. Although we emphasize the use of entire (finite-horizon) trajectories for obtaining accurate value function estimates, we note that our methods can also be combined with standard TD(λ) updates and variants.

Our measure of strategy class complexity generalizes the classical notion of VC dimension, and our methods develop connections between problems of current interest in reinforcement learning and well-studied issues in the theory of supervised learning. We also discuss a number of practical planning algorithms for POMDPs that arise from our methods.

[*]Contact author. Address: AT&T Labs, Room A235, 180 Park Avenue, Florham Park, New Jersey, 07932. E-mail: mkearns@research.att.com.

[†]On sabbatical from Tel Aviv University.

1 Introduction

Markov decision processes (MDPs) and reinforcement learning have become a standard framework for planning and learning in uncertain environments. The desire to attack problems of increasing complexity with this formalism has recently led researchers to focus particular attention on MDPs with (exponentially or even infinitely) large state spaces, and on partially observable MDPs (POMDPs). A number of interesting and basic issues arise when designing planning and learning algorithms for large POMDPs.

First, as the state space becomes large, the classical representation of a POMDP by explicit tables of transition probabilities, rewards and observations clearly becomes infeasible. To intelligently discuss the problem of planning — that is, computing a good strategy in a given POMDP — *compact* or implicit representations of POMDPs (such as representations in which the next-state distributions can be factored [BDH99, BK98, KP99]) must be developed. Second, even a compact representation of a POMDP is no guarantee that a good *strategy* in that POMDP has a compact representation. Thus, we must also be prepared to consider compact representations for strategies (such as those typically considered when using function approximation in standard MDPs [SB98]).

Motivated by these issues, in this paper we address the following question: given the ability to simulate experience in a POMDP M, and given a class of strategies Π, how can we choose a $\pi \in \Pi$ whose expected return is close to the best possible within the class Π? Here we are imagining that Π is a restricted class of strategies, perhaps given by some compact representation, or perhaps defined by some natural limitation on strategies (such as having bounded memory). We will focus on the question of how we should generate experience in the POMDP in a way that allows us to simultaneously estimate the value of as many strategies as possible.

More precisely, we consider a setting in which we are given access to a *generative model*, or simulator, for M. Informally, this is a "black box" that allows us to generate many trajectories of experience in the POMDP, from different states and under different strategies. Generative models are a natural way in which a large POMDP might be specified, and are more general than most compact, structured representations, in the sense that such representations usually provide an efficient way of implementing a generative model. Generative models provide less information than explicit tables of probabilities, rewards, and observations, but are more powerful than a single continuous, irreversible trajectory of experience generated according to some fixed strategy. In this sense, results obtained via a generative model blur the distinction between what is typically called "planning" and "learning" in POMDPs.

We will study the question: How many calls to a generative model are required in order to have data sufficient to choose a near-best strategy in a given class? This question is analogous to the classical question of *sample complexity* in supervised learning — but harder. The added difficulty lies in the *reuse* of data. In the supervised learning setting, *every* random example $\langle x, f(x)\rangle$ provides feedback about *every* hypothesis function $h(x)$ (namely, how close $h(x)$ is to $f(x)$). If $h(x)$ is restricted to lie in some "hypothesis class" $\mathcal{H}$, this reuse permits bounds on the number of random examples required that are far smaller than the number of functions in $\mathcal{H}$. For instance, if $\mathcal{H}$ contains only a finite number n of functions, $O(\log(n))$ bounds (ignoring parameters of the problem other than n for now) are obtained on the sample size required to choose a near-best approximation to $f(x)$ lying in $\mathcal{H}$. In the case that $\mathcal{H}$ is infinite, sample sizes are obtained that depend only on some measure of the *complexity* of $\mathcal{H}$ (such as VC dimension). Note that these bounds have *no dependence* on the complexity of the target function f or the size of the input domain.

In the POMDP setting, however, we must decide *how* to use the generative model — that is, which states and actions to feed to the generative model — in order to recover this same desirable reuse of experience across multiple strategies in our class. To see the issue more clearly, consider the "straw man" algorithm that, starting with some initial strategy $\pi \in \Pi$, uses the generative model to generate many Monte Carlo trials of π from the start state s_0, and thus form an accurate empirical estimate of $V^{\pi}(s_0)$. It is not clear that these trajectories under π are of much use in evaluating a different $\pi' \in \Pi$, as π and π' may quickly disagree on which actions to take. Thus, the naive method of generating Monte Carlo trials would result in $O(n)$ bounds on the number of calls to the generative model, rather than $O(\log(n))$, for the finite case $|\Pi| = n$.

In contrast, in the ensuing sections, we shall present two different ways of generating "reusable" trajectories. Both methods yield similar theoretical bounds. The first method, which we call *trajectory trees*, has an easier and more intuitive analysis, and also directly suggests some practical algorithms for approximate planning using a generative model. The second method, which we call *random trajectories*, requires a more difficult analysis, but uses a considerably weaker form of generative model.

Both methods generate a (relatively) small number of trajectories — a number that is independent of the state-space size of the POMDP, depends only linearly on a general measure of the *complexity* of the strategy class Π, and depends exponentially on the horizion time. We prove that these generated trajectories are enough to give us accurate estimates of the expected return of any strategy in Π. Both methods can be viewed as generating a "small" set of trajectories that provide an accurate estimate of the value of any strategy in the class. They can thus be easily used with many standard approaches to search in strategy space, such as gradient descent and local search. For such algorithms, the exponential dependence on the horizon time can be replaced by a factor linear in the number of steps of search to be performed. Although we emphasize the use of entire (finite-horizon) trajectories for obtaining accurate value function estimates, we note that our methods can also be combined with standard TD(λ) updates and variants.

Our measure of strategy class complexity is inspired by and generalizes the notion of VC dimension in supervised learning, and we give bounds that recover for our setting the most powerful analogous results in supervised learning — bounds for arbitrary, infinite strategy classes that depend on the dimension of the class rather than the size of the state space.

Although our development concentrates on restricted classes of strategies, this can be translated to many other types of restriction. For example, if we restrict our value function estimates to be of a certain parametric form, this implies a restriction on policies, namely to those policies that can be achieved by behaving greedily with respect to the parametric value functions. Similarly, if we estimate models of a POMDP that lie in some restricted class (for instance, POMDPs with factored next-state distributions), this implies a restriction to strategies that are optimal with respect to such POMDPs. In this way, our results for restricted strategy classes can be applied to any other hypothesized constraint on a POMDP planning problem.

Our main contributions are:

- Giving specific methods for generating "resuable" trajectories from generative models;
- Proving that these methods allow the generation of a set of trajectories sufficient to evaluate an entire class of strategies;
- Giving practical algorithms for approximate planning in POMDPs based on resuable trajectories;
- Establishing connections between natural problems in supervised learning and reinforcement learning, via generalizations of the VC dimension.

2 Preliminaries

We begin with the definition of a (partially observable) Markov decision process, explicitly allowing the possibility of an infinite number of states.

Definition 1 *A* **Markov decision process (MDP)** *over a* **state set** S*, with* **start state** $s_0 \in S$*, and* **actions** $\{a_1, \ldots, a_k\}$*, consists of:*

- **Next-state distributions***: For each state-action pair* (s, a)*, a next-state distribution* $P(s'|s, a)$ *that specifies the probability of transition to each state* s' *upon execution of action* a *from state* s*.*

Note that $\sum_{s'} P(s'|s,a) = 1$ if S is finite, and $\int_S P(s'|s,a)ds' = 1$ in the case of infinite S.

- **Rewards**: *For each state-action pair (s,a), a real-valued* **reward** [1] *$R(s,a)$ for executing action a from state s. We assume rewards are bounded in absolute value by $R_{\max}$.*

A **Partially Observable Markov Decision Process (POMDP)** *consists of an underlying MDP and* **observation distributions** *$Q(o|s)$ for each state s. Here o is a random variable called the* **observation** *made at state s.*

In the above definitions we have a assumed a designated start state. This is because once we limit the class of strategies we will entertain, there may not be a single "best" strategy in the class, unless we explicitly induce a metric of some kind — one strategy might be better from certain states, and another strategy better from other states, and so there may not be a single strategy that is optimal from all states. We thus adopt the common assumption of a fixed start state s_0 in the underlying MDP. (An equivalent definition is to assume a fixed distribution D over start states, since s_0 can be a "dummy" state whose next-state distribution under any action is D.)

An agent wandering in a POMDP takes actions and receives rewards, as in an MDP, but the agent never directly sees the identity of the current state. Rather, the agent has access only to the current observation. This step towards realism greatly complicates the problems of both planning and learning. Intuitively, the agent may never know the true state, but must attempt to track the current *belief state* — that is, the likelihood that it is in each of the possible states of the underlying MDP. In general, this belief state may be arbitrarily complex, depending strongly on both the initial state of the underlying MDP (or an initial distribution of states), as well as on the entire history of actions and observations. This is a sharp contrast to the fully observable case, where the optimal policy depends only on the current state. Recent work has made some interesting proposals for planning in POMDPs via approximate belief state tracking [BK98, MS99].

We will primarily be interested in POMDPs with a large or infinite number of states, thus precluding approaches that require access to explicit tables describing the next-state and observation distributions and the rewards. Instead, we assume that our algorithms are "given" a POMDP M in the form of the ability to *sample* the behavior of M. Thus, the model given is simulative rather than explicit. We call this ability to sample the behavior of M a *generative model*.

[1] Note that for simplicity, we have assumed that all rewards are in fact deterministic. However, all of our results have easy generalizations for the case of stochastic rewards (with an appropriate and necessary dependence on the variance of the reward distributions).

Definition 2 *A* **generative model** *for a POMDP M is randomized algorithm that, given as input a state-action pair (s,a), outputs a state s' that is distributed according to the next-state distribution $P(\cdot|s,a)$, an observation o that is distributed according to the distribution $Q(\cdot|s)$, and the reward $R(s,a)$.*

Thus, a generative model for a POMDP simply consists of a generative model for the underlying MDP, along with a random observation at each state. At first blush, this definition may seem unreasonably generous — we are essentially assuming that we are provided with a fully observable simulation of a partially observable process. However, the key point is that algorithms provided with this generative model must still find a strategy that performs well in the *partially observable* setting. For instance, although we could simply use the generative model to find a near-optimal policy (state-to-action mapping) for the underlying MDP [KMN99], this policy will be useless in the POMDP, where the state is unknown. We can really only use the generative model to find a strategy that maps from (histories of) observables to actions. As a concrete example, in designing an elevator control system [CB96], we may have access to a simulator that generates random rider arrival times, and keeps track of the waiting time of each rider, the number of riders waiting at every floor at every time of day, and so on. However helpful this information might be in *designing* the controller, this controller must only *use* information about which floors currently have had their call button pushed (the observables). In any case, readers uncomfortable with the power provided by our POMDP generative models are referred to the results of Section 5, where they are replaced by an extremely weak form of simulation — namely, a subroutine for generating only the *observable* history along truly random trajectories, with no ability to "reset" the simulation to any chosen state.

We now move on to define strategies and strategy classes in a POMDP. In general, an agent will, at any time t, have seen some sequence of observables $o_0, \ldots, o_t$, and will have chosen actions and received rewards for each of the t time steps prior to the current one. Thus, we may write its **observable history** as a list of triples of observations, actions and rewards:

$$h = \langle (o_0, a_0, r_0), \ldots, (o_{t-1}, a_{t-1}, r_{t-1}), (o_t, \_, \_) \rangle$$

where the last entry indicates that observable histories always conclude with the observation made at the (unknown) final state reached. We will also refer to observable histories as **trajectories**. Such trajectories are the inputs to strategies:

Definition 3 *A* **strategy** *π in a POMDP is any (stochastic) mapping from finite observable histories $\langle (o_0, a_0, r_0), \ldots, (o_{t-1}, a_{t-1}, r_{t-1}), (o_t, \_, \_) \rangle$ to actions. A* **strategy class** *Π is any set of strategies. The expected return of a strategy π starting from state s_0 is denoted by $V^{\pi}(s_0)$.*

Of course, this definition includes the special case in which the MDP is fully observable (the observations are the states), and Π is simply a class of policies. As we have already remarked, we can think of Π as representing a *constraint* or *bias* on strategies adopted with the hope of avoiding the intractability of full belief-state planning, while still permitting good, if suboptimal, performance.

Our results are rather general with respect to the notion of **return** that is used. Our main assumption is that for any history h, the return along h can be approximated by a (possibly discounted) sum of the first H_ϵ rewards, with an error of at most ϵ. We refer to H_ϵ as the ϵ-horizon. This assumption includes two well-studied notions of return. The **finite-horizon average** return, with horizon H, is handled by simply setting $H_\epsilon = H$. Here, by definition, the return depends only on the first H steps taken from the start state, so the parameter ϵ is unnecessary. The standard **infinite-horizon discounted** return is also easily handled. Let γ be the discount factor. The return along a history h is $\sum_{i=1}^{\infty} \gamma^{i-1} r_i$ where r_i is the ith reward in h. If we set $H_\epsilon = \log_\gamma(\epsilon(1-\gamma)/R_{max})$, we are guaranteed that the discounted return beyond the first H_ϵ steps cannot contribute more than ϵ. We also assume there is a bound V_{max} on the return of any trajectory; for example, $V_{max} = R_{max}/(1-\gamma)$ suffices in the discounted case. It is worth noting that the infinite undiscounted return does not fall into this category, and our techniques cannot be directly applied to it.

As discussed before, once we limit the class of strategies we will entertain, there may not be a single strategy in the class that is the best from every state, and therefore we compare the strategies with respect to the designated start state s_0. This permits the following definition.

Definition 4 *Let M be a POMDP with start state s_0, and let Π be a class of strategies. Then*

$$opt(M, \Pi) = \sup_{\pi \in \Pi} V^\pi(s_0) \tag{1}$$

where $V^\pi(s_0)$ is the expected return of π from s_0.

With these definitions, we can now state our problem more precisely. We are given a generative model for a POMDP M and a strategy class Π. How *many* calls to the generative model must we make in order to have enough data to choose a $\pi \in \Pi$ whose performance $V^\pi(s_0)$ approaches $opt(M, \Pi)$? And more importantly, *which* calls should we make to the generative model in order to minimize the number of calls required?

3 The Trajectory Tree Method

We now describe the first of our two methods for creating "reusable" trajectories from a generative model. Although this method requires a stronger generative

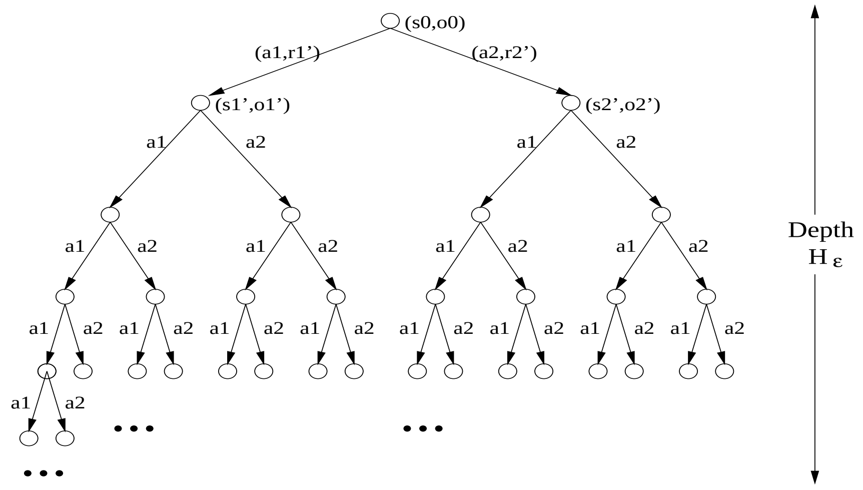

Figure 1: The structure of a typical trajectory tree. (Shown here with actions a_1 and a_2, and with observation, state, and reward labels omitted below the second level.)

model than that required of our second method (presented in Section 5), it enjoys two advantages over that method that justify our considering it first. First, it has a simpler and more intuitive analysis. Second, it directly leads to some practical gradient ascent algorithms that we describe in Section 4.

Recall that we are given a generative model for a POMDP M with distinguished start state s_0. For ease of exposition, we assume there are only two actions in M, action a_1 and action a_2, but our results easily generalize to any finite number of actions (see Appendix C).

A *trajectory tree* is simply a binary tree in which each node is labeled by both a state in M and an observation, and has a single child for action a_1 and a single child for action a_2. Additionally, each link from a parent to a child will have a reward labeling that link. The depth of the tree will always be H_ϵ, the ϵ-horizon time [2], so the total size (number of nodes) in each trajectory tree will be about 2^{H_ϵ}. We will eventually discuss various settings in which this exponential dependence on H_ϵ can be avoided.

A trajectory tree is built in a straightforward manner from the generative model. The root will always be labeled by the start state s_0 and the observation o_0. To generate the two children of the root, we call the generative model on (s_0, a_1) and (s_0, a_2), and the generative model returns the two next states reached (say s'_1 and s'_2, respectively), the two observations made (say o'_1 and o'_2,

[2] Again, here we are simultaneously covering both the case of finite-horizon average return and infinite-horizon discounted return.

respectively), and the two rewards received ($r'_1 = R(s_0, a_1)$ and $r'_2 = R(s_0, a_2)$). Then (s'_1, o'_1) and (s'_2, o'_2) will label the a_1-child and a_2-child of the root, and the links from the root to these children will have rewards r'_1 and r'_2. Recursively, for any node s of depth less than H_ϵ, we generate two children and rewards in the same way with the generative model. (See Figure 1.)

Now for any *deterministic* strategy π, and for any trajectory tree T, π accumulates a well-defined return on T. Strategy π defines a path through the tree T — we start π at the root, where it sees whatever observation is stored there (that is, the observable history so far consists only of this observation). Strategy π then decides to either take action a_1 or action a_2, which selects a child of the root. Inductively, if π has reached some internal node in T, we can feed to π the entire observable history generated along the path to this node, and discover which child of the current node π selects. In this way, we "run" π on T to reach some leaf node of T, and we define the return $R(\pi, T)$ to be the return along the path taken by π. In the general case that π is stochastic, π defines a *distribution* on paths in T, and $R(\pi, T)$ becomes the expected return according to this distribution. If we have created trajectory trees $T_1, \ldots, T_m$, a natural estimate for $V^\pi(s_0)$ is then

$$\hat{V}^\pi(s_0) = \frac{1}{m} \sum_{i=1}^{m} R(\pi, T_i). \tag{2}$$

For technical reasons, all trajectory trees in the remainder of this section are assumed to have depth equal to the $\epsilon/2$-horizon (rather than the ϵ-horizon) time. Our main goal in this section is now to establish a nontrivial relationship between the quality of this estimate and the "sample size" m. As is typical of analogous results in supervised learning, we will actually prove *uniform convergence* theorems. Section 3.1 first treats the easiest case, that of finite Π; Section 3.2 then extends the theorem to infinite Π for deterministic strategies; and Section 3.3 finally generalizes the result to infinite and stochastic Π.

3.1 The Case of Finite Π

To convey the intuition via the simplest analysis, we begin with the case where Π is a finite class of n strategies.

Theorem 3.1 *Let Π be any class of n (stochastic) strategies in an arbitrary POMDP M. Let m trajectory trees be created using a generative model for M, and let $\hat{V}^\pi(s_0)$ be the resulting estimates. If*

$$m = O((V_{\max}/\epsilon)^2 \log(n/\delta)) \tag{3}$$

then with probability at least $1 - \delta$, $|V^\pi(s_0) - \hat{V}^\pi(s_0)| \leq \epsilon$ holds simultaneously for all $\pi \in \Pi$. The total number of calls made to the generative model will thus be at most $2^{H_\epsilon} m = O(2^{H_\epsilon}(V_{\max}/\epsilon)^2 \log(n/\delta))$.

Proof (Sketch): Let us fix any strategy $\pi \in \Pi$. Then each trajectory tree is used to generate a run (or distribution on runs) of the strategy π, and our estimate $\hat{V}^{\pi}(s_0)$ for the return of strategy π is the average return of its m runs (distributions on runs). The crucial observation is that for this fixed π, the values $R(\pi, T_i)$ that are generated by the different trajectory trees T_i are independent. This is easily seen if we imagine that each trajectory tree is constructed by first constructing the path (or distribution on paths) determined by π, and then afterwards constructing the rest of the tree. The resulting distribution on trees is identical to the distribution generated by our original description.

This independence implies that we can apply the Chernoff bound for the deviation of an estimate from its mean. Since the maximum return is bounded by V_{max}, we have that the probability that the deviation is more than $\epsilon/2$ is bounded by $e^{-\epsilon^2 m/(4V_{max}{}^2)}$.

So far we have restricted our attention to a fixed policy π. By appealing to the so-called *union bound*, we have that the probability that *any* $\pi \in \Pi$ deviates by more than $\epsilon/2$ from its mean is bounded by $ne^{-\epsilon^2 m/(4V_{max}{}^2)} = \delta$. Finally, the error from truncation at depth $H_{\epsilon/2}$ is at most $\epsilon/2$. Hence, we have that with probability $1 - \delta$, the largest deviation of our estimates from the true values is at most $\epsilon/2 + \epsilon/2 = \epsilon$. $\square$

The crucial point to note about this result is the dependence on n: it is only *logarithmic* in n, as opposed to the linear bound expected for the straw-man Monte Carlo approach described earlier. Thus, the trajectory tree approach is achieving considerable *reuse* of the generated experience: with only on the order of $\log(n)$ data, we can get an excellent estimate of the value of all n strategies.

3.2 The Case of Infinite Deterministic Π

We now move on to the more general case of infinite classes of deterministic strategies. Our treatment will again parallel the theme of data reuse in supervised learning, and while this section will give a complete description and explanation of our result, some of the mathematical details will be left to Appendix A.

When addressing the sample complexity of supervised learning, perhaps the most important observation is that even though a class $\mathcal{H}$ may be infinite, the number of possible *behaviors* of $\mathcal{H}$ on a finite set of points is often not exhaustive. More precisely, in the case of a class of boolean functions, we say that the set $x_1, \ldots, x_d$ is *shattered* by $\mathcal{H}$ if every of the 2^d possible labelings of these points is realized by some $h \in \mathcal{H}$. The VC dimension of $\mathcal{H}$ is then defined as the size of the largest shattered set. It is known that if the VC dimension of $\mathcal{H}$ is d, then the number $\Phi_d(m)$ of possible labelings induced by $\mathcal{H}$ on a set of m points is bounded by $(em/d)^d$, which is much less than 2^d for $d \ll m$. This nontrivial bound provides the key leverage exploited by the classical VC dimension results, so we will concentrate on replicating this leverage in our setting.

In the interests of concreteness, we will now focus on and sketch the ideas behind the two-action case of our theorem, which can be done by appealing only to the familiar VC dimension of boolean functions. Generalizations of all of our results to the case of multiple actions are provided in Appendix C.

Suppose Π is an infinite class of deterministic strategies in a two-action POMDP. Then each strategy $\pi \in \Pi$ is simply a deterministic function mapping from the set of all observable histories to the set $\{a_1, a_2\}$, and thus can be viewed as a boolean function on observable histories. We can thus write $\mathrm{VC}(\Pi)$ to denote the familiar VC dimension of the set of binary functions Π.

We now show intuitively why a strategy class Π of bounded VC dimension d cannot induce exhaustive behavior on a set $T_1, \ldots, T_m$ of trajectory trees for $m \gg d$. Note that if $\pi_1, \pi_2 \in \Pi$ are such that their "labelings" $\langle R(\pi_1, T_1), \ldots, R(\pi_1, T_m)\rangle$ and $\langle R(\pi_2, T_1), \ldots, R(\pi_2, T_m)\rangle$ differ, then we must have $R(\pi_1, T_i) \neq R(\pi_2, T_i)$ for some $1 \leq i \leq m$. But if π_1 and π_2 give different returns on T_i, then they must choose different actions at some node in T_i. Thus, if h is the observable history leading to that node, $\pi_1(h) \neq \pi_2(h)$. In other words, every different labeling of the set of m *trees* yields a different labeling of the set of $m \cdot 2^{H_\epsilon}$ observable *histories* that are given by the trees. This means that the number of different tree labelings can be at most $\Phi_d(m \cdot 2^{H_\epsilon}) \leq (m \cdot 2^{H_\epsilon}/d)^d$. By developing this argument carefully, and appealing to classical uniform convergence techniques, we obtain the following theorem.

Theorem 3.2 *Let Π be any class of deterministic strategies for an arbitrary two-action POMDP, and let $\mathrm{VC}(\Pi)$ be the VC dimension of Π. Let m trajectory trees be created using a generative model for M, and let $\hat{V}^\pi(s_0)$ be the resulting estimates. If*

$$m = O\left((V_{\max}/\epsilon)^2 (H_\epsilon \mathrm{VC}(\Pi) + \log(1/\delta))\right) \tag{4}$$

then with probability at least $1-\delta$, $|V^\pi(s_0) - \hat{V}^\pi(s_0)| \leq \epsilon$ holds simultaneously for all $\pi \in \Pi$.

The full proof of this Theorem is in Appendix A.

3.3 The Case of Infinite Stochastic Π

We now address the general case of infinite stochastic strategy classes. Again for the sake of concreteness, we focus on the two-action case, deferring details of the multiple action case to Appendix C.

Our approach involves transforming stochastic strategies into deterministic ones, thereby reducing to the case handled in the previous section. In particular, given any class of stochastic functions Π, each with domain X (where X is the set of all possible finite observable histories), we first extend the domain from

X to $X \times [0,1]$. For each stochastic function $\pi \in \Pi$, we define a corresponding deterministic function π' over $X \times [0,1]$ as follows: $\pi'(h,r) = a_1$ if $r \leq \mathbf{Pr}[\pi(h) = a_1]$, and $\pi'(h,r) = a_2$ otherwise. Let Π' be the collection of these deterministic functions π'.

Given a set of stochastic strategies Π, we now define its *pseudo-dimension* $\mathrm{pVC}(\Pi)$ to be $\mathrm{VC}(\Pi')$, the VC dimension of the corresponding set of deterministic strategies. (This is equivalent to the conventional definition of the pseudo-dimension of Π, when Π is viewed as a set of maps into real-valued action-probabilities.)

We now show how we can modify the original POMDP such that the execution of the stochastic strategy π in the original POMDP is equivalent to the execution of the deterministic strategy π' in the modified POMDP. We first explain the construction assuming full observability. Intuitively, we simply add to each state an observable value $r \in [0,1]$ which is uniformly distributed. Formally, given an MDP with states S, we augment the state space to become $S \times [0,1]$. On state transitions that would have entered a state s in the original MDP, we now enter an augmented state (s,r), where r is distributed uniformly in $[0,1]$, and is independent of all previous events. It is easy to see that if in the original MDP the stochastic policy π had some probability $p = \mathbf{Pr}[\pi(s) = a_1]$ of taking action a_1, then π' has the same probability of doing so.

Thus, by construction the transformed MDP has left the underlying transition probabilities (viewed as functions only of s) unchanged. Indeed, to an observer seeing only a sequence of the s-components (and not the r-components), a stochastic $\pi : S \mapsto \{a_1, a_2\}$ executed on the original MDP would be indistinguishable from the corresponding deterministic $\pi' : S \times [0,1] \mapsto \{a_1, a_2\}$ executed on the augmented MDP, as desired. Thus, it is easy to see that $V^{\pi}(s_0) = \mathbf{E}_r[V^{\pi'}(s_0, r)]$, where the left-hand side is the value function in the original MDP, the right-hand side is the value function in the MDP with the augmented states, and $\pi \in \Pi$ and $\pi' \in \Pi'$ are any pair of corresponding stochastic and deterministic strategies.

Returning to the partially observable case, the observable histories can also be augmented with these (observed) extra random variables at each state, and the action taken now depends on r at the current state. Moreover, since we are now working with deterministic strategies, evaluation of $R(\pi', T')$ on a given trajectory tree T' (which is essentially an unaugmented trajectory tree T, with an additional random variable r at each of its state-nodes) would require only taking one path deterministically through the tree. Doing so, we get an estimate of the value of π' and therefore an estimate of the value of the corresponding stochastic π. Our result for deterministic strategies therefore applies, giving the following theorem.

Theorem 3.3 *Let Π be any class of stochastic strategies for an arbitrary two-*

action POMDP, and let pVC(Π) *be the pseudo-dimension of* Π. *Let m trajectory trees be created using a generative model for M with augmented states, and let $\hat{V}^{\pi}(s_0)$ be the resulting estimates derived using their corresponding deterministic strategies. If*

$$m = O\left((V_{\max}/\epsilon)^2 (H_\epsilon \mathrm{pVC}(\Pi) + \log(1/\delta))\right) \tag{5}$$

then with probability at least $1-\delta$, $|V^{\pi}(s_0) - \hat{V}^{\pi}(s_0)| \le \epsilon$ holds simultaneously for all $\pi \in \Pi$.

We note that a very similar result can be obtained for the original proposal of *averaging* over all paths in the original trajectory tree T (without the state augmentation proposed above).

4 Algorithms for Approximate Planning

Given a generative model for a POMDP M, the uniform convergence results of the preceding sections immediately suggest a class of algorithms for approximate planning, given a strategy class Π: we generate m trajectory trees $T_1, \ldots, T_m$, and search for a $\pi \in \Pi$ that maximizes $\hat{V}^{\pi}(s_0) = (1/m) \sum R(\pi, T_i)$. The following simple corollary to the uniform convergence results establishes the soundness of this approach.

Corollary 4.1 *Let Π be a class of strategies in a POMDP M, and let the number m of trajectory trees be as given in Theorem 3.1 (finite Π), Theorem 3.2 (infinite deterministic Π) or Theorem 3.3 (infinite stochastic Π). Let*

$$\hat{\pi} = \arg\max_{\pi \in \Pi} \{\hat{V}^{\pi}(s_0)\} \tag{6}$$

be the policy in Π with the highest empirical return on the m trajectory trees. Then with probability at least $1-\delta$, $\hat{\pi}$ is near-optimal within Π:

$$V^{\hat{\pi}}(s_0) \ge opt(M, \Pi) - 2\epsilon. \tag{7}$$

If it is computationally infeasible to perform the suggested maximization, one can search for a local maximum π instead, and uniform convergence again assures us that $\hat{V}^{\pi}(s_0)$ is a trusted estimate of our true performance. Of course, even lowering our ambitions to finding a local maximum of the surface $\hat{V}^{\pi}(s_0)$ remains an expensive proposition, since each trajectory tree is of size exponential in H_ϵ.

However, in practice it may be possible to significantly reduce the cost of the search, by means of a simple observation. Suppose we are using either a class Π of deterministic strategies (or transformed stochastic strategies), and that we perform a greedy local search over Π to optimize $\hat{V}^{\pi}(s_0)$. Then at any time in the search, to evaluate the policy we are currently considering, we really need to look at only a single path of length H_ϵ in each tree, corresponding to the path taken by the strategy being considered by our local search. Thus, we should build the trajectory trees *lazily* — that is, incrementally build each node of each tree only as it is needed to evaluate $R(\pi, T_i)$ for the current strategy π. If there are parts of a tree that are reached only by poor policies, then a good search algorithm may never even build these parts of the tree. In any case, each step of the local search now takes time only linear in H_ϵ.

Avoiding the exponential dependence on H_ϵ via lazy trajectory tree construction would appear to apply only to the case of deterministic strategies, or to stochastic strategies transformed into deterministic ones. Without the state augmentation described in the last section, stochastic strategies define a distribution over *all* the paths in a trajectory tree, and thus evaluation of the current π may in general require examining complete trees. However, suppose $\Pi = \{\pi_\theta : \theta \in \mathbb{R}^d\}$ is a smoothly parameterized family of stochastic strategies. It turns out that there is a practical implementation of *stochastic* gradient ascent on $\hat{V}^{\pi_\theta}(s_0)$ that again has per-step time that is only linear in H_ϵ. The key to the stochastic gradient ascent algorithm is that it *subsamples* the trajectory trees, again permitting lazy construction. The update made to the current position θ_0 at each step will be a learning-rate parameter times an unbiased estimate of the gradient $(d/d\theta)\hat{V}^{\pi_\theta}(s_0)$ evaluated at θ_0.

In fact, in the discounted case, rather than doing stochastic gradient ascent on $\hat{V}^{\pi_\theta}(s_0)$, it is possible to perform stochastic gradient ascent directly on the *true* value function $V^{\pi_\theta}(s_0)$. Formally, in Appendix B we give an algorithm that, given a generative model for the POMDP and a setting of the parameters θ_0, enjoys the following properties:

- (Efficiency) The algorithm has expected running time $O(1/(1-\gamma))$;
- (Unbiasedness) The algorithm outputs an unbiased estimate of $(d/d\theta)V^{\pi_\theta}(s_0)$;
- (Bounded Variance) This estimate has bounded variance (for fixed γ, R_{max}, and given a bound on the gradient $|(d/d\theta)\mathbf{Pr}[\pi_\theta(s) = a]|$ of the parameterized family itself).

It should be clear that the three conditions above are exactly what we need to do stochastic gradient ascent directly on the surface $V^{\pi_\theta}(s_0)$. We note that this algorithm leads us close to an interesting line of research pursued by Kimura, Yamamura and Kobayashi [KYK95], and that our procedure is also similar in spirit to William's REINFORCE [Wil92]. (See also the recent paper by Baird and Moore [BM99], which gives a generalization of REINFORCE.) The main differences between our approach and Kimura et al. and Williams are the following. First, we find an unbiased estimate of the gradient in finite time, whereas they only converge to such an estimate asymptotically. Second, we explicitly bound the variance of our estimator, whereas if we assume only a bound on the derivative of $\mathbf{Pr}[\pi_\theta(s) = a]$ as we did above, either of the previous algorithms can still have arbitrarily large variance[3].

5 The Random Trajectory Method

We now move on to present the second of our two methods for generating only a small number of trajectories sufficient to evaluate all the strategies in a large class. We call this second approach the *random trajectory* method, and one advantage it enjoys over the trajectory tree method is that it does not need the full power of a generative model for the POMDP. In fact, the random trajectory method requires only the observable histories generated by truly random trajectories from the start state. Resets to states other than the start state are unnecessary, as is the ability to see the underlying states along the trajectory. We begin with a definition capturing this weaker form of simulation.

Definition 5 *A* **random trajectory generator** *for a partially observable Markov decision process M generates an observable history of a given length H starting from the start state s_0 by following the truly random policy (at each state each action is equally likely). Thus, the generator outputs only the observable history*

$$\langle (o_0, a_0, r_0), \ldots, (o_{H-1}, a_{H-1}, r_{H-1}), (o_H, \_, \_) \rangle \tag{8}$$

generated from s_0 by choosing each a_i uniformly from the set of actions.

As was the case for the depth of our trajectory trees in Section 3, we will choose the length H of our random trajectories to be the ϵ-horizon time H_ϵ. In this section, we discuss only deterministic policy classes. Our results can be extended to stochastic policy classes following the same lines of Section 3.3.

Recall that in the method of trajectory trees, we used a (stronger) generative model to create a finite set of trajectory trees, and proved that this set gave uniformly good estimates of expected return within Π. Here the proposal is even simpler, but its analysis will be more challenging: we will simply take m truly random histories, derive from these histories an estimate of $V^\pi(s_0)$ for every $\pi \in \Pi$, and show that a relatively small value for m again yields uniformly good estimates.

Recall that each trajectory tree T allowed us to get an evaluation of *any* deterministic strategy π, since any π always defines some path in T. But if h is just a single random history, how can we evaluate an arbitrary π on t, given

[3]Though there are stronger assumptions that will allow bounding these variances. Basically, infinite variance should not occur if we implement π_θ via a function approximator with sigmoidal or softmax outputs. Details in Appendix B.

that π may diverge from h? The answer is that we cannot. Instead let us define the variable $\mathrm{acc}_\pi(h)$ to be 1 if π "accepts" the history h and 0 otherwise. More precisely, given a history h, if for any prefix of h the strategy π would have generated the same action, then $\mathrm{acc}_\pi(h) = 1$ (recall that π is deterministic).

We can now define the estimate of $V^\pi(s_0)$ we derive from a set of random trajectories. Let $\mathcal{H} = \{h_1, \ldots, h_m\}$ be a set of histories from the random trajectory generator. For each strategy $\pi \in \Pi$, define $\hat{V}^\pi(s_0)$ as follows. Let $S_\pi(\mathcal{H}) \subseteq \mathcal{H}$ include all the histories for which $\mathrm{acc}_\pi(h) = 1$; thus,

$$S_\pi(\mathcal{H}) = \{h | h \in \mathcal{T} \text{ and } \mathrm{acc}_\pi(h) = 1\}. \tag{9}$$

Then $\hat{V}^\pi(s_0)$ is the average return of the histories in $S_\pi(\mathcal{T})$:

$$\hat{V}^\pi(s_0) = (1/|S_\pi(\mathcal{H})|) \sum_{h \in S_\pi(\mathcal{H})} r(h) \tag{10}$$

where $r(h)$ is the return of trajectory h.

Thus, in analogy with Section 3, we now have a method of generating a set of m random observable histories $\mathcal{H} = \{h_1, \ldots, h_m\}$ (as opposed to m trajectory trees), and for any strategy π, there is a well-defined estimate $\hat{V}^\pi(s_0)$ based on the set $\mathcal{H}$. As in Section 3, we wish to establish a nontrivial relationship between the "sample size" m and the deviations $|V^\pi(s_0) - \hat{V}^\pi(s_0)|$ for all $\pi \in \Pi$. We again begin with the case of finite Π.

5.1 Random Trajectory Analysis: Finite Strategy Classes

Let D_π be the distribution on observable histories h induced by π; thus, $D_\pi(h)$ is the probability that following policy π from s_0 for H steps generates h. Similarly, let $D_\$$ be distribution on observable histories induced by the truly random trajectory generator.

We start with a few simple lemmas about these distributions. Lemma 5.1 establishes that for any fixed strategy π, there is a $1/2^H$ probability that the random trajectory generator will produce a history that is accepted by (that is, consistent with) π. Lemma 5.2 establishes that for any fixed strategy π, a history generated by the random trajectory generator and accepted by π can be viewed as a Monte Carlo trial of π.

Lemma 5.1 *For any strategy π,*

$$\mathbf{Pr}_{h \sim D_\$}[\mathrm{acc}_\pi(h) = 1] = \frac{1}{2^H}. \tag{11}$$

Proof: Note that $D_\$(h) = (1/2^H)P(h)$, where

$$P(h) = Q(o_0|s_0)P(s_1|s_0,a_0)\cdots Q(o_{H-1}|s_{H-1})P(s_H|s_{H-1},a_{H-1})Q(o_H|s_H). \tag{12}$$

Therefore,

$$\mathbf{Pr}_{h\sim D_\$}[\mathrm{acc}_\pi(h)=1] = \sum_h \frac{1}{2^H}P(h)\mathrm{acc}_\pi(h) = \frac{1}{2^H}\sum_h P(h)\mathrm{acc}_\pi(h). \tag{13}$$

But $D_\pi(h) = P(h)\mathrm{acc}_\pi(h)$, and for this reason $\sum_h P(h)\mathrm{acc}_\pi(h) = 1$. $\square$

Lemma 5.2 *For any policy π, and any history h,*

$$D_\pi(h) = D_\$(h|\mathrm{acc}_\pi(h)=1). \tag{14}$$

Proof: Recall that $D_\pi(h) = P(h)\mathrm{acc}_\pi(h)$ and $D_\$(h) = (1/2^H)P(h)$. This implies that $D_\$(h$ and $\mathrm{acc}_\pi(h)=1) = (1/2^H)P(h)\mathrm{acc}_\pi(h)$. Finally, by Lemma 5.1, $\mathbf{Pr}_{h\sim D_\$}[\mathrm{acc}_\pi(h)=1] = 1/2^H$. Using Bayes formula one can see that the two probabilities are identical. $\square$

Next we show that, if we let the set $\mathcal{H}$ of histories returned by the random trajectory generator be sufficiently large, with high probability, the sets S_π of accepted histories are "large" for every $\pi \in \Pi$, where Π is a finite class of deterministic strategies.

Lemma 5.3 *Let $\mathcal{H}$ be a set of m histories returned by the random trajectory generator, and let Π be a finite class of n deterministic strategies. Provided that $m > 2^{H+3}\log(2n/\delta)$, the probability that there exists a strategy $\pi \in \Pi$ for which $|S_\pi(\mathcal{H})| < m/2^{H+1}$ is at most $\delta/2$.*

Proof: For a given strategy π, the expected number of trajectories in S_π is $\sum_{h\in\mathcal{H}}\mathrm{acc}_\pi(h) = m/2^H$ by Lemma 5.1. Therefore the probability that π will accept fewer than $m/2^{H+1}$ trajectories is bounded by $e^{-m/2^{H+3}}$. The probability that some π has $|S_\pi| < m/2^{H+1}$ is bounded by $ne^{-m/2^{H+3}}$. Therefore, for $m > 2^{H+3}\log(2n/\delta)$ the probability is at most $\delta/2$. $\square$

The final step in the finite Π analysis is showing that if S_π is sufficiently large, then the value of $\hat{V}^\pi(s_0)$ is a good approximation to the expected return of π.

Lemma 5.4 *Let $\mathcal{H}$ be a set of histories returned by the random trajectory generator, and let Π be a finite class of n deterministic strategies. Conditioned on the event that*

$$|S_\pi(\mathcal{H})| > m_1 = \left(\frac{V_{\max}}{\epsilon}\right)^2\log(2n/\delta) \tag{15}$$

for every $\pi \in \Pi$, with probability $1 - \delta/2$,

$$|\hat{V}^\pi(s_0) - V^\pi(s_o)| \le \epsilon \tag{16}$$

for every $\pi \in \Pi$.

Proof: Under the conditioning event, for a given strategy π the probability that the estimate deviates by ϵ is at most $e^{-\epsilon^2 m_1/V_{max}{}^2}$; here we are using the fact that the histories in $S_\pi(\mathcal{H})$ are distributed according to $D_\pi(\cdot)$ (Lemma 5.2). The probability that some π deviates by ϵ is thus bounded by $ne^{-\epsilon^2 m_1/V_{max}{}^2}$. $\square$

We can now state and prove our main theorem for the case of finite Π.

Theorem 5.5 *Let $\mathcal{H}$ be a set of m histories returned by the random trajectory generator, and let Π be a finite class of n deterministic strategies. Provided that*

$$m > 2^{H+3}(V_{\max}/\epsilon)^2 \log(2n/\delta) \tag{17}$$

with probability at least $1 - \delta$ we have

$$\left|V^\pi(s_0) - \hat{V}^\pi(s_0)\right| \le \epsilon \tag{18}$$

simultaneously for all $\pi \in \Pi$.

Proof: By our choice of m, Lemma 5.3 guarantees that with probability $1-\delta/2$, for each $\pi \in \Pi$ we have that $|S_\pi(\mathcal{H})| > m_1$. By Lemma 5.4 this implies that, with probability $1 - \delta/2$, we have $|\hat{V}^\pi(s_0) - V^\pi(s_o)| \le \epsilon$ for all $\pi \in \Pi$. $\square$

If we compare Theorem 5.5 to Theorem 3.1 (the uniform convergence result for trajectory trees in the finite Π case), we see that the total amount of experience that must be generated in the POMDP is quite similar. The main differences are in how that experienced is *organized* — in one case into trajectory trees, and in the current case a flat list of random histories — and in how it is *generated*, in the current case from a much weaker simulative model than is required to build trajectory trees.

5.2 Random Trajectory Analysis: Infinite Strategy Classes

We now move on to sketch the derivation of our results for infinite classes of strategies, still limiting our attention to the case where all strategies are deterministic for simplicity.

Let Π be an infinite class of deterministic strategies in a two-action POMDP. Let us associate with Π the class $\mathcal{F}_\pi^H = \{\mathrm{acc}_\pi(h) : \pi \in \Pi\}$ of binary functions mapping observable histories h of length exactly H to $\{0, 1\}$. We first bound the VC dimension of $\mathcal{F}_\pi^H$ in terms of the VC dimension of Π (which is well-defined, since each $\pi \in \Pi$ maps histories of any length to $\{a_1, a_2\}$).

Lemma 5.6 *Let* Π *be any class of deterministic strategies in a two-action POMDP, and let* $\mathrm{VC}(\Pi)$ *be the VC dimension of* Π. *Then*

$$\mathrm{VC}(\mathcal{F}_\pi^H) = O(\mathrm{VC}(\Pi)\log(H)). \tag{19}$$

Proof: Let $\mathcal{H}$ be any set of observable histories of length H. Note that if $\pi_1, \pi_2 \in \Pi$ have identical behavior on all prefixes of histories in $\mathcal{H}$ — that is, $\pi_1(h[j]) = \pi_2(h[j])$ for all $h \in \mathcal{T}$ and all $1 \leq j \leq H$ — then we must have $\mathrm{acc}_{\pi_1}(h) = \mathrm{acc}_{\pi_2}(h)$ for all $h \in \mathcal{T}$. Thus, each different labeling of $\mathcal{H}$ by the class $\mathcal{F}_\pi^H$ gives a different labeling by Π of the set of all prefixes of $\mathcal{H}$. If $\mathcal{F}_\pi^H$ shatters d length-H histories, we must have $2^d \leq \Phi_{\mathrm{VC}(\Pi)}(dH) \leq (dH/\mathrm{VC}(\Pi))^{\mathrm{VC}(\Pi)}$, where $\Phi_d(m)$ is the dichotomy-counting function of uniform convergence theory [Vap82]. The lemma follows from some algebraic manipulations. □

The following lemma states that for each strategy in Π, we have many trajectories that it accepts.

Lemma 5.7 *Let* $\mathcal{H}$ *be a set of* m *histories returned by the random trajectory generator, and let* Π *be any class of deterministic strategies. Provided that* $m = O(2^H \log(\mathrm{VC}(\Pi)/\delta))$, *the probability that there exists a strategy* $\pi \in \Pi$ *for which* $|S_\pi(\mathcal{H})| \leq m/2^{H+1}$ *is at most* $\delta/2$.

Let us review a bit. Lemma 5.1 shows that the expected fraction of the $h \in \mathcal{H}$ satisfying $\mathrm{acc}_\pi(h) = 1$ is exactly $1/2^H$, while Lemma 5.6 establishes that the VC dimension of the variables $\mathrm{acc}_\pi(h)$ is bounded. From this we proved in Lemma 5.7 that for sufficiently large sample size m, all the sets $S_\pi(h)$ will be large. However, this is not quite enough — merely having large samples of Monte Carlo trials of each π (which by Lemma 5.2 is exactly what the $S_\pi(\mathcal{H})$ are) does not ensure that the average return over each $S_\pi(\mathcal{H})$ will be near $V^\pi(s_0)$. In other words, since the reader can readily verify that $V^\pi(s_0) = \mathbf{E}_{h\sim D_\$}[2^H \mathrm{acc}_\pi(h) r(h)]$, and that $\hat{V}^\pi(s_0) = (1/m_\pi)\sum_{h\in\mathcal{H}} \mathrm{acc}_\pi(h) r(h)$ (where $m_\pi = |S_\pi(\mathcal{H})|$), it is not uniform convergence of the variables $\mathrm{acc}_\pi(h)$ we are concerned with so much as uniform convergence of the variables $\mathrm{acc}_\pi(h)r(h)$. For this we require generalizations of VC dimension such as combinatorial dimension [Hau92], since the $\mathrm{acc}_\pi(h)r(h)$ are now real-valued rather than binary variables.

For this reason we need to extend the family $\mathcal{F}_\Pi$ to $\mathcal{G}_\Pi$, by defining $g_\pi(h) = f_\pi(h)r(h)$. The important point is that $r(h)$ is fixed by the model, and therefore independent from π. For the function class $\mathcal{G}_\Pi$ it is easy to show that combinatorial dimension is equal to the VC dimension of $\mathcal{F}_\Pi$. Since we bounded the VC dimension of $\mathcal{F}_\Pi$ by $O(d\log H)$ we immediately have a bound for the combinatorial dimension of $\mathcal{G}_\Pi$. Given that $\mathcal{G}_\Pi$ has a bounded combinatorial dimension, we can deduce that the estimates for all g_π converge uniformly. The following lemma formalizes this.

Lemma 5.8 *Given that*

$$m > c(\frac{d'}{\epsilon'^2}(\log 1/\epsilon') + \frac{1}{\epsilon'^2}\log 1/\delta), \tag{20}$$

we have that for every $\pi \in \Pi$, with probability at least $1-\delta$,

$$|\hat{V}^{\pi}(s_0) - V^{\pi}(s_0)| \leq \epsilon, \tag{21}$$

where $\epsilon' = \epsilon/2^H V_{\max}$ and c is a constant. The probability is over the draw of the random trajectories in $\mathcal{H}$.

Now we can state the main result of this section.

Theorem 5.9 *Let Π be any class of deterministic strategies in a two-action POMDP M, and let $\mathcal{H} = \{h_1, \ldots, h_m\}$ be m histories from the random trajectory generator for M. If*

$$m = O\left(\left(\frac{2^H V_{\max}}{\epsilon}\right)^2 (\mathrm{VC}(\Pi)\log(H)(H + \log(V_{\max}/\epsilon)) + \log(1/\delta))\right) \tag{22}$$

then with probability at least $1-\delta$ we have

$$|V^{\pi_i}(s_0) - \hat{V}^{\pi}(s_0)| \leq \epsilon, \tag{23}$$

simultaneously for all $\pi \in \Pi$.

Proof: By Lemma 5.6 the VC-dimension of $\mathcal{F}_\Pi$, d', is bounded by $O(d \log H)$. As we explained, the combinatorial dimension of $\mathcal{G}_\Pi$ is also d'. By Lemma 5.7 for each policy $\pi \in \Pi$ we accept at least $m/2^{H+1}$ trajectories. Given a sample of m trajectories, the theorem follows from Lemma 5.8. $\square$

References

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999. To appear.

[BK98] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI*, pages 33–42, 1998.

[BM99] Leemon Baird and Andrew W. Moore. Gradient descent for general Reinforcement Learning. In *Advances in Neural Information Processing Systems 11*, 1999.

[CB96] R. Crites and A. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pages 1017–1023, 1996.

[Hau92] David Haussler. Decision-theoretic generalizations of the PAC model for neural networks and other applications. *Information and Computation*, 100:78–150, 1992.

[KMN99] M. Kearns, Y. Mansour, and A. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[KP99] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[KYK95] H. Kimura, M. Yamamura, and S. Kobayashi. Reinforcement learning by stochastic hill climbing on discounted reward. In *Proceedings of the 12th International Conference on Machine Learning*, pages 295–303, 1995.

[MS99] D. McAllester and S. Singh. Approximate planning for factored POMDPs using simplified belief states. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, 1999.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning.* MIT Press, 1998.

[Vap82] V.N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer, 1982.

[Wil92] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

A Appendix

In this section, we provide the proof of Theorem 3.2 (infinite deterministic Π).

We assume the reader is familiar with the definition of VC dimension of sets of binary functions. There are several ways to generalize this to sets of real-valued functions, and we now introduce one of them, given in [Vap82]. If $\mathcal{H} = \{h : X \mapsto [-B, B]\}$ is a set of real-valued functions bounded by B, define $\mathrm{VC}_r(\mathcal{H})$ to be the (convention) VC dimension of the set of binary functions $\{I(h, r, \cdot) : h \in \mathcal{H}, r \in (-B, B)\}$, where $I(h, r, x) = 1$ if $h(x) \geq r$, $I(h, r, x) = 0$ otherwise. That is, we take $\mathcal{H}$, introduce all possible thresholds to get indicator functions, and finally take the conventional VC dimension of the resulting set of indicators.

For a fixed deterministic strategy π, we have a map $R(\pi, \cdot)$ from trajectory trees to real numbers $[-V_{max}, V_{max}]$. Thus, Π may be viewed as a set of real-valued functions with trees as its domain, and so it makes sense to ask what $\mathrm{VC}_r(\Pi)$ is. (This is *not* to be confused with the alternative view of Π as a set of binary functions mapping from histories to actions.) We then have the following lemma.

Lemma A.1 *Let Π be a set of deterministic strategies for a two-action POMDP, with VC dimension* $\mathrm{VC}(\Pi)$ *when viewed as a set of maps from observable histories to actions. Then when viewed as a set of maps from the space of all depth-H trajectory trees to* $[-V_{\max}, V_{\max}]$*, the set Π has dimension bounded by*

$$\mathrm{VC}_r(\Pi) = O\left(H\mathrm{VC}(\Pi)\right) \tag{24}$$

Proof: Let $d = \mathrm{VC}(\Pi)$. From Sauer's Lemma (see [Vap82]), Π can realize at most $(ek/d)^d$ different action labelings on any set of k observable histories. Now on m trajectory trees, there are at most $k = m2^{(H+1)}$ different observable histories (one for each node). So if we view each π as selecting a path through each tree, then Π can realize at most $(ek/d)^d = (em2^{(H+1)}/d)^d$ different selections (where each "selection" is a set of m paths taken by a strategy π, one per tree). Moreover, this set of trees has a total of $m2^H$ paths from roots to leaves, and so $R(\pi, T)$ can take on at most $m2^H$ values for $\pi \in \Pi$ and T in our set of m trees. Thus, we need consider only $m2^H$ settings of the threshold parameter r.

Multiplying the quantities together, we see therefore that the set of indicator functions used to define $\mathrm{VC}_r(\Pi)$ (where Π is now viewed as a map from trees to $[-V_{max}, V_{max}]$) can, on m trees, realize at most $m2^H(em2^{(H+1)}/d)^d$ different labelings. Now in order for Π (viewed a a set of real functions) to shatter m trees, it must be able to realize at least 2^m different labelings, so that

$$m2^H(em2^{(H+1)}/d)^d \geq 2^m \tag{25}$$

must hold. A little algebra shows this implies $m = O\left(H\mathrm{VC}(\Pi)\right)$, proving the lemma. $\square$

We now state one more result due to Vapnik [Vap82], with which we will be ready to prove our theorem. Let $\mathcal{H}$ be a set of bounded real-valued functions $h : X \mapsto [-B, B]$ bounded by B, and let $d = \mathrm{VC}_r(\mathcal{H})$. Let D be some distribution over X, and let $x_1, \ldots, x_m$ be m iid samples drawn according to D. Then with probability $1 - \delta$,

$$\sup_{h\in\mathcal{H}} \left| E_D[h(x)] - \frac{1}{m}\sum_{i=1}^{m} h(x_i) \right| \leq O\left(B\sqrt{\frac{d\log\frac{m}{d} + \log\frac{1}{\delta}}{m}} \right) \tag{26}$$

holds (where the randomization is over the draw of the x_i's).

We are now ready to prove the theorem.

Proof (Theorem 3.2): There are two sources in the error in our estimate of $V^\pi(s_0)$: Error from truncating at depth H, and error from the randomness in the sampling of trajectory trees. Let $V^\pi_{H_\epsilon}(s_0)$ be the expected H_ϵ-step sum of discounted reinforcements for π starting at s_0. Clearly, $\mathbf{E}_T[R(\pi, T)] = V^\pi_{H_\epsilon}(s_0)$ holds for all π. From our choice of H_ϵ, it also holds by construction that

$|V^\pi_{H_\epsilon}(s_0) - V^\pi(s_0)| \leq \epsilon/2$ for all π. Finally, we apply Equation (26) with $x = T$ being the trajectory trees, $\mathcal{H} = \Pi$ (viewed as a set of real-valued functions), $B = V_{max}$, $h = \pi$, $h(x) = R(\pi, T)$, $d = \mathrm{VC}_r(\Pi)$ and $\mathbf{E}_D[h(x)] = \mathbf{E}_T[R(\pi, T)] = V^\pi_{H_\epsilon}(s_0)$, and find that with probability $1 - \delta$,

$$\left| V^\pi_{H_\epsilon}(s_0) - \frac{1}{m}\sum_{i=1}^{m} R(\pi, T_i) \right| \leq O\left(V_{max}\sqrt{\frac{d \log \frac{m}{d} + \log \frac{1}{\delta}}{m}} \right) \tag{27}$$

holds simultaneously for all $\pi \in \Pi$. Substituting $d = O(H_\epsilon \mathrm{VC}(\Pi))$ from Lemma A.1, we therefore see that with a choice of

$$m = O\left((V_{max}/\epsilon)^2 (H_\epsilon \mathrm{VC}(\Pi) + \log(1/\delta)) \right) \tag{28}$$

we have that with probability $1 - \delta$, it holds simultaneously for all $\pi \in \Pi$ that $|V^\pi_{H_\epsilon}(s_0) - (1/m)\sum_{i=1}^m R(\pi, T_i)| \leq \epsilon/2$. When this is true, the triangle inequality gives $|V^\pi(s_0) - (1/m)\sum_{i=1}^m R(\pi, T_i)| \leq |V^\pi(s_0) - V^\pi_{H_\epsilon}(s_0)| + |V^\pi_{H_\epsilon}(s_0) - (1/m)\sum_{i=1}^m R(\pi, T_i)| \leq \epsilon/2 + \epsilon/2 = \epsilon$ simultaneously for all π, which proves the theorem. □

B Appendix

In this Appendix, we give the details of our gradient descent algorithms described in Section 4. Appendix B.1 gives the first algorithm, which finds an unbiased estimate of $(d/d\theta)R(\pi_\theta, T)$ for a given trajectory tree T, and may be used with stochastic gradient ascent to optimize the empirical return on a set of trees. Appendix B.2 gives the second algorithm, which finds an unbiased estimate of $(d/d\theta)V^{\pi_\theta}(s_0)$ and which may be therefore be used to directly optimize the value itself of our strategy; it then closes with discussions of some related algorithmic issues.

Throughout this Appendix, we assume Π is a a **smoothly parameterized family of stochastic strategies** parameterized by $\mathbb{R}^d$, by which we mean a set of stochastic strategies $\Pi = \{\pi_\theta \mid \theta \in \mathbb{R}^d\}$, such that for any action a and any fixed observable history $h = \langle (o_1, a_1, r_1), \ldots, (o_{i-1}, a_{i-1}, r_{i-1}), (o_i, \_, \_) \rangle$, $\mathbf{Pr}[\pi_\theta(h) = a]$ is continuously differentiable in θ. To simplify the notation, we also use $(\pi_\theta(h))_i$ to denote $\mathbf{Pr}[\pi_\theta(h) = a_i]$; a convenient way to think about it is as π_θ mapping finite histories to k-dimensional vectors (in the $k-1$-dimensional simplex), where the ith element of the output is the probability of choosing action a_i. As mentioned, we also assume a bound on the gradient with respect to θ — that for all h and all i, $|\frac{d}{d\theta}(\pi_\theta(h))_i| \leq B$ for some constant B. Finally, to reduce notational clutter, we will frequently drop the "$|_{\theta=\theta_0}$" used denote evaluation at the current θ_0 (for example, as in $\frac{d}{d\theta}|_{\theta=\theta_0} R(\pi_\theta, T)$) when there is no risk of confusion.

Function: **EstimateTreeGradient**$(\Pi, \theta_0, T, \gamma)$
Input: Strategy class Π, parameter setting θ_0, trajectory tree T, discount γ.
Output: A vector that is an unbiased estimate of $\frac{d}{d\theta}|_{\theta=\theta_0} R(\pi_\theta, T)$.

1. Randomly choose a depth $d \in \{0, \ldots, H-1\}$, with $\mathbf{Pr}[d=k] \propto \gamma^k$.
2. Perform a random walk on the tree T starting from the root with actions chosen by π_{θ_0} (that is, where the action at each node is determined by π_{θ_0} applied to the history of the path to that node), down to depth d on the tree. Let s_d denote the node we arrive at, and h_d the history leading to s_d.
3. For each action $a \in \{a_1, \ldots, a_k\}$, start from the node s_d, execute action a (meaning go to the action-a child), and then again follow π_{θ_0} down the tree until we reach a leaf. Let r_a be the sum of discounted reinforcements received executing the trajectory from s_d down to the leaf taking action a at s_d, where the discounting starts at 1, not γ^d, for the sequence. (Thus, the reinforcement for taking a at s_d is undiscounted, the next reinforcement is discounted by γ and so on, with the last step's reinforcement discounted by γ^{H-d-1}.)
4. Output
$$\frac{1-\gamma^{H+1}}{1-\gamma} \sum_{i=1}^{k} r_{a_i} \frac{d}{d\theta}\bigg|_{\theta=\theta_0} (\pi_\theta(h_d))_i \tag{29}$$
(where as before, $(\pi_\theta(h_d))_i$ denotes $\mathbf{Pr}[\pi_\theta(h_d) = a_i]$).

Figure 2: Algorithm for finding an unbiased estimate of the gradient of $R(\pi_\theta, T)$ with respect to θ.

B.1 Appendix

We now describe our algorithm that, given a smoothly parameterized family of stochastic strategies and a fixed setting of the parameters θ_0, finds an unbiased, bounded-variance estimate of the gradient of $(1/m)\sum_{i=1}^{m} R(\pi_\theta, T_i)$ with respect to θ, evaluated at θ_0. For a fixed trajectory tree T, Figure 2 gives an algorithm for obtaining an unbiased estimate of the gradient of $R(\pi_\theta, T)$. The estimate for the gradient for m trees is then easily obtained by averaging over all m trees.

We will shortly give and prove a formal statement of this algorithm's correctness, but let us first informally sketch the rationale behind its different steps. First consider Steps 3 and 4 of the algorithm. In Step 3, we have already fixed some state s_d with history h_d, and we take k runs – one for each action – to get the r_a, each of which is exactly a Monte Carlo estimate of $Q^{\pi_{\theta_0}}(h_d, a)$. Given these estimates, a natural step to take is to tweak the parameters θ so as to increase the probability of taking actions with high r_a. And indeed, it is easy to check that the gradient estimate in Step 4 is exactly so that gradient ascent will do this—the output is (proportional to) the gradient of $\sum_i ((\pi_\theta(h_d))_i r_{a_i}$.

Steps 1 and 2 choose s_d, and there are also two interesting ideas regarding this that are worth describing. First, if s_d is far from the root, then because of discounting, its gradient should be discounted by a factor of γ^d. But if d is large, then it would seem a waste to perform all the simulation and computation required by the algorithm, only to multiply the result by γ^d and discount it almost all the way to 0. What we do instead is rather than multiply s_d's estimate by γ^d, we choose the depth d with probability proportional to γ^d (Step 1), which avoids the need to discount by γ^d. This is a standard idea from Monte Carlo sampling, often used to significantly reduce the variance of estimators. Also along the same lines, the derivative of $\pi_\theta(h_d)$ should be given more weight if there is a high probability of getting to h_d. In the trajectory-tree setting, a node s_d's gradient should have some weight proportional to how often it is visited; so again, rather than weighting according to visitation probability, we instead choose a node with probability proportional to its visitation probability (Step 2), and then do not further weight the resulting gradient.

The above sketches the rationale for **EstimateTreeGradient**. Our algorithm for finding an unbiased estimate of $(d/d\theta)V^{\pi_\theta}(s_0)$ is based on this; the main idea is to consider applying the same algorithm to trajectory trees of "infinite" depth, and this is described in detail in Appendix B.2. The remainder of this section is devoted to proving the correctness of the **EstimateTreeGradient** algorithm—that it gives an unbiased and bounded-variance estimate of the gradient. (That its running time is $O(kH)$ is obvious.)

Our strategy for proving the algorithm's correctness closely follows our discussion above. We will, for the fixed tree T, considering the change to $R(\pi_\theta, T)$ if any of the probabilities $(\pi_\theta(h_i))_j$ is changed for some node i in the tree. As discussed, the result of such a change at a node at depth d should have a "weight" proportional to γ^d, and also proportional to its visitation probability by a π_{θ_0}-random walk. After steps 1 and 2 sample a node with probability proportional to its weight, we output just that node's contribution to the gradient; the $(1-\gamma^{H+1})/(1-\gamma)$ factor in our output is the normalization constant of the sampling distribution. Here follows a more rigorous proof that **EstimateTreeGradient** outputs an unbiased and bounded-variance (bounded in terms of k, γ, R_{max}, and B) estimate of the gradient.

Theorem B.1 *Let the random vector Z be the output of the* **EstimateTreeGradient** *algorithm. Then, $E[Z] = \frac{d}{d\theta}|_{\theta=\theta_0} R(\pi_\theta, T)$. In addition,* $\mathrm{Var}(Z_i) \leq (R_{\max} kB/(1-\gamma))^2$ *for each i (where Z_i is the i-th element of Z).*

Proof (Sketch): Let $\ell = 2^{H+1} - 1$ be the number of nodes in our trajectory tree T, and let us fix some labeling of the nodes from 1 to ℓ, so that we may refer to the "i-th node" in the tree. Also, for $i = 1, \ldots, \ell$, let h_i denote the history at the i-th node in the tree (from the root down to that node), and let $p_{ij} = (\pi_\theta(h_i))_j$ be the probability of taking action a_j at node i, as an implicit function of θ. Then for fixed T, $R(\pi_\theta, T)$ is completely determined by $\{p_{ij} | i = 1, \ldots, \ell, j = 1, \ldots, k\}$, since the p_{ij} completely specify the probabilities of taking each action at any node in the tree T. Thus writing R to denote $R(\pi_\theta, T)$, by the chain rule of differentiation,

$$\frac{d}{d\theta} R(\pi_\theta, T) = \sum_{i=1}^{\ell} \left(\sum_{j=1}^{k} \frac{dR}{dp_{ij}} \frac{dp_{ij}}{d\theta} \right) \tag{30}$$

where we recall $p_{ij} = (\pi_\theta(h_i))_j$ are implicit functions of θ. We compare this with (29), the expression of the algorithm's output in Figure 2, and notice (29) bears some similarity to the term in parenthesis above, with r_{a_j} playing the role of $\frac{dR}{dp_{ij}}$. Now notice also that, if p_{ij} is increased by ϵ, then $R = R(\pi, T)$ will increase by ϵ times the probability of reaching the subtree rooted at node i times the average reward of the action-a_j subtree of node i. But if the i-th node is s_d, then $\gamma^d r_{a_j}$ is an unbiased estimate of the (discounted) value of this action-a_j subtree, since r_{a_j} is just a Monte Carlo estimate of the value of this subtree. Letting $P[i]$ denote the probability that a π_{θ_0}-random walk will reach node i, we therefore see, by our argument, that $P[i]\gamma^d r_{a_j}$ is in fact an *unbiased* estimate of $\frac{dR}{dp_{ij}}$ when the node i is s_d (e.g. when r_{a_j} is gotten from a random walk as in Step 3 of the algorithm, starting from node i and taking a_j first). Putting this back into Equation 30 and letting $d(i)$ denote the depth of node i in the tree, we see that

$$\begin{aligned} \frac{d}{d\theta} R(\pi_\theta, T) &= \sum_{i=1}^{\ell} \left(\sum_{j=1}^{k} \gamma^{d(i)} \mathrm{P}[i] \mathbf{E}[r_{a_j} | s_d = \text{node } i] \frac{dp_{ij}}{d\theta} \right) && (31) \\ &= \sum_{i=1}^{\ell} \gamma^{d(i)} \mathrm{P}[i] \left(\sum_{j=1}^{k} \mathbf{E}[r_{a_j} | s_d = \text{node } i] \frac{dp_{ij}}{d\theta} \right) && (32) \end{aligned}$$

This is just a sum over nodes i, with the node-i term having a "weight" of $\gamma^{d(i)}\mathrm{P}[i]$. To find an unbiased estimate of the sum, we may, as in Steps 1-2 of the algorithm, randomly choose one of the terms with probability proportional to its weight and then (with appropriate normalization, by $\sum_{i=i}^{\ell} \gamma^{d(i)}\mathrm{P}[i] = (1-\gamma^{H+1})/(1-\gamma)$ in our case to make the "weights" sum to one and be a probability distribution) output just that one term in the sum, to get an unbiased estimate of the entire sum. This is of course just a standard Monte Carlo estimate of a mean. Doing this, and further replacing the expectation with a sample of r_{a_j} then recovers the expression (29) used in our algorithm.

Finally, by our assumption of $|\frac{d}{d\theta}(\pi_\theta(h))_i|$ being bounded as $\leq B$, we see the output (29) is bounded by $kBR_{max}/(1-\gamma)$, which therefore gives our bound on the variance. $\square$

B.2 Appendix

In this section, we describe how the **EstimateTreeGradient** algorithm can be modified to give in $O(kH)$ time an unbiased estimate of $(d/d\theta)V^{\pi_\theta}(s_0)$. This enables us to perform stochastic gradient ascent directly on the true value itself. Note however that this version of the algorithm requires that we run the generative model on each gradient ascent step (that is, it is not sufficient to sample just a fixed number of trajectory trees, and then make no further use of the generative model).

The idea is simple. We have given an algorithm for finding an unbiased estimate of the gradient with respect to a finite trajectory tree. If we could apply it to an "infinite" trajectory tree (corresponding to a depth $H = \infty$), then we would get an unbiased estimate of the gradient of $V^{\pi_\theta}(s_0)$. To see this, note that when we sample infinite trees, we have $\mathbf{E}_T[R(\pi_\theta, T)] = V^{\pi_\theta}(s_0)$ exactly for each θ. So, applying $d/d\theta$ to both sides and interchanging expectations and derivatives, we get

$$\mathbf{E}_T\left[\frac{d}{d\theta}R(\pi_\theta, T)\right] = \frac{d}{d\theta}V^{\pi_{\theta_0}}(s_0) \tag{33}$$

which shows unbiasedness.

If we do use such an "infinite" tree (built lazily, of course) and try to apply **EstimateTreeGradient** of the previous section, there are no difficulties with Steps 1 and 2 of the algorithm – d is chosen with $\mathbf{Pr}[d = k] = \gamma^k/(1-\gamma)$, and we take a random walk down to depth d to get s_d. But Step 3 now tells us to take a walk to depth H, which is infinitely far away, and this is not tractably implementable. Fortunately, given a POMDP with discount γ, we may transform it into an *undiscounted* POMDP, but which now has a $(1-\gamma)$ chance of transitioning to a zero-cost absorbing state after each step. Indeed, one of the common justifications for discounting is that it represents a $(1-\gamma)$ chance after each step of the POMDP ending; we are simply making use of this fact here. Now in the transformed POMDP, we may proceed to sum the "infinite" trajectory in Step 3, and this will terminate finitely once we transition into the absorbing state in the transformed POMDP (which will happen with probability 1, and on expectation after $O(1/(1-\gamma))$ steps). Incidentally, our simple transformation can also be applied to William's REINFORCE algorithm to get it to output unbiased estimates in finite time, but its variance may still be unbounded.

To see that this new algorithm still has bounded variance, note that each r_a is the sum of a geometrically distributed number of individually bounded terms, so each r_a also has bounded variance, and hence our final estimator also has bounded variance (again assuming a bound on the derivative of π_θ).

Figure 3 gives the step-by-step details of this proposal, with an additional parameter H' that represents a slight improvement over what we have described so far. Setting $H' = 0$ gives the algorithm we have just described. The parameter H' is a modification that tries to reduce the estimator's variance. When we modified **EstimateTreeGradient** to get this new algorithm, we added a source of variance in that the r_a's are now sums of d' reward terms from the "undiscounted" POMDP, where d' is geometrically distributed. The variability in d', the number of terms we are summing, adds another source of variability to our final estimate. Our modification is that, with the H' "execution depth" parameter, we will always take at least H' *discounted* steps from s_d, after which we take a further, geometrically distributed number of steps. But now, the contribution of the geometrically distributed number of steps is discounted by $\gamma^{H'}$, and therefore contributes less to the variance. So, with H' set on the order of $1/(1-\gamma)$ or of the ϵ-horizon time, we incur a time penalty of an additional expected H' steps per run, but can have a significantly smaller variance in our estimator.

Finally, we close this section by very briefly addressing a number of issues relating to practical applications of these algorithms (perhaps the most pressing of which is the lack of mention of exploration), consider possible further modifications to the algorithm, and relate it also to William's REINFORCE and to Kimura et. al.'s algorithm.

Exploration. We have presented a reinforcement learning algorithm that can perform stochastic gradient ascent to optimize $V^{\pi_0}(s_0)$, but which does not seem to need to do any exploration. Is there any magic here? It turns out that, even though we have bounded the variance of our gradient estimators, the *gradient* itself may still be exponentially small in H. To see this, imagine a trajectory tree where only one of the leaves gives a reward; if we were choosing actions at random, it would take us $O(2^H)$ tries just to find this node. Thus, our reinforcement learning algorithm may be best applied mainly to domains where not too much exploration is needed for the (for instance, if there are many intermediate rewards), or if we are somehow able to initialize θ_0 to a "good" strategy, in which case this algorithm may be used to improve upon it from there. A particularly intriguing example is if we were to, by other means, estimate the Q-function (or value function) as $\hat{Q}_\theta(s,a)$, using some smooth function approximator parameterized by θ. Then adding a inverse-temperature parameter β, we may define a stochastic policy $(\pi_{\theta,\beta}(s))_j = \exp(\beta\hat{Q}_\theta(s,a_j))/\sum_i \exp(\beta\hat{Q}_\theta(s,a_i))$, and then do gradient ascent on $V^{\pi(\theta,\beta)}(s_0)$ with parameters (θ,β).

Deterministic strategies. For learning deterministic strategies, the main algorithm we had suggested was local greedy search with lazy tree construction.

```
Function: **EstimateValueGradient**($\Pi, \theta_0, s_0, \gamma, G, H'$)
Input: strategy class $\Pi$, current parameters $\theta_0$, start state $s_0$, discount $\gamma$ generative model
$G$, execution depth $H'$.
Output: A row vector that is an unbiased estimate of $\frac{d}{d\theta}|_{\theta=\theta_0} V^{\pi_\theta}(s_0)$.

1. Set $s$ to be the root ($s_0$) node in the (to be lazily built) tree.

2. With probability $1-\gamma$, jump to step 4.

3. Let $a$ be an action chosen randomly by the stochastic policy $\pi_{\theta_0}$ applied to the
   history at node $s$. Take action $a$ from $s$, setting $s$ to be the action-$a$ child of $s$.
   Goto step 2.

4. Set node $s_d = s$, and let $h_d$ be the history at $s$.

5. For each action $a \in \{a_1, \ldots, a_k\}$
       5.1 Set $r_a := 0$.
       5.2 (Using $G$,) execute action $a$ from state $s_d$, getting some resulting reward $R$
   and child-node $s'$. Set $r_a := r_a + R$.
       5.3 For $i = 1$ to $H'$,
           5.3.1 (Using $G$), take a random step according to $\pi_{\theta_0}$ applied to the history
   at $s$, letting $R$ be the reward and $s'$ the action's corresponding child node.
           5.3.2 Set $r_a := r_a + \gamma^i R$, and $s := s'$.
       5.4 With probability $1-\gamma$, jump to 5.6.
       5.5 (Using $G$) Take a random step from $s$ according to $\pi_{\theta_0}$, getting some resulting
   reward $R$ and next-state $s'$. Set $r_a := r_a + \gamma^{H'} R$, and $s := s'$. Goto step 5.4.
       5.6 end. (of for-loop)

6. Output
```

$$\frac{1}{1-\gamma} \sum_{j=1}^{k} r_{a_j} \left. \frac{d}{d\theta} \right|_{\theta=\theta_0} (\pi_\theta(s_d))_j \tag{34}$$

Figure 3: Algorithm for finding an unbiased estimate of $\frac{d}{d\theta}|_{\theta=\theta_0} V^{\pi_\theta}(s_0)$.

But to avoid local maxima, it may also be worthwhile *embedding* our class of deterministic strategies in a larger, smoothly parameterized class of stochastic strategies, and then do the search in that larger, smoother space, for example with our stochastic gradient ascent algorithm. Then, a standard trick from the neural network literature called the soft-barrier can be used to re-constrain ourselves to deterministic strategies (if so desired): Rather than doing gradient ascent on $V^{\pi_\theta}(s_0)$, we do gradient ascent on $V^{\pi_\theta}(s_0) - \beta\mathbf{E}[\mathcal{H}(\pi(h))]$ where $\mathcal{H}$ is the entropy function and the expectation is over an appropriate distribution of histories (say choose $h = h_d$, the history at s_d, on each step of the algorithm). This way, we penalize strategies that have entropy or randomness in their choices of actions. By letting β slowly increase from 0 to ∞, we have an increasing bias towards deterministic strategies. (Note we do not recommend learning a stochastic strategy π_θ, and then making it deterministic by picking the most likely action (under π_θ) at each step; this may work for some problems, but seems to us much more difficult to justify.)

A weaker simulator. Throughout Section 4 and this Appendix, we had assumed a generative model that allows us to sample a next-state/next-observation pair starting from any state. What if, as in Section 5, our generative model only allows us to "reset" to s_0 and to take Monte Carlo runs from there, but could not reset to any intermediate state? Recall that, after reaching s_d in our algorithm, we iterated through all k actions, and then our final output in (34) is a value that is a sum of k terms, with one term corresponding to each Monte Carlo run from s_0. If we have only the weaker simulator, we are able to try only one Monte Carlo run from s_d. To handle this, we may randomly choose one action, take just that one Monte Carlo run, and then return k times the one term (within the summation) corresponding to the chosen action. It is then immediate that this still gives a bounded-variance unbiased estimator. Incidentally, if we have an estimate of the Q-function at s_d, this can also be used to reduce the variance resulting from the randomization of the action's choice.

A random walk? Williams and Kimura et. al. had some nice ideas regarding this. Using the weaker simulator described in the previous paragraph, we uniformly chose a random action upon reaching s_d, and then gave that one term in the sum a multiplier of k. Alternatively, we may choose our action at s_d according to the distribution given by $\pi_{\theta_0}(h_d)$ also, so that at every step we are taking a random walk according to π_{θ_0}. Then, to keep our estimator unbiased, we would multiply by $1/(\pi_{\theta_0}(h_d))_j$ if action a_j was chosen, making the algorithm's final output

$$\frac{1}{1-\gamma}\frac{1}{(\pi_{\theta_0}(h_d))_j} r_{a_j} \left.\frac{d(\pi_\theta(h_d))_j}{d\theta}\right|_{\theta=\theta_0} \tag{35}$$

This should be compared with (34). Unfortunately, the $1/(\pi_{\theta_0}(h_d))_j$ multiplier may be arbitrarily large, and so our estimator may have arbitrarily large or even infinite variance (and examples are easy to construct). This (minus the undiscounted POMDP idea to make the algorithm terminate in finite time, rather than only converge only asymptotically) is essentially Kimura et. al.'s algorithm. (Actually, they also had a second version, and some interesting ideas regarding eligibility traces, that we will mention shortly.) Particularly if used with the method we described earlier to learn deterministic strategies, this would seem disastrous, as we are driving most of the $(\pi_{\theta_0}(h_d))_j$ to zero. But fortunately, we found a simple, additional condition to ensure bounded variance; namely if $\left|\frac{1}{(\pi_{\theta_0}(h_d))_j}\frac{d(\pi_\theta(h_d))_j}{d\theta}\right|$ were bounded, then there would be no problems. As a simple example, if the π_θ were a neural network with a sigmoid activation function at the output (that, say, represents probability of taking action a_1) in a two-action POMDP, then probability of taking action a_1 is $(\pi_\theta(h_d))_1 = \sigma(f_\theta(h_d))$ for some f_θ, and its derivative is $\sigma(f_\theta(h_d))(1-\sigma(f_\theta(h_d))f'_\theta(h_d)$. Notice how the $\sigma(\cdot)$ term therefore cancels the $1/(\pi_{\theta_0}(h_d))_1 = 1/\sigma(\cdot)$ term, and so we are no longer worried about the estimator blowing up (making appropriate assumptions about f). A similar argument may be made about softmax (which generalizes sigmoids to many outputs) activation functions for representing k-action stochastic strategies.

Eligibility traces. Related to the last point, if we are taking just one random walk, then it seems that we might as well try letting every node on the random walk be s_d, to get rid of one more source of variance in our estimate. That is, rather than choosing a single depth d at random and then picking a single node s_d at that depth, we might as well enumerate over as many d as we can, letting each node on our random walk be s_d in turn, and then taking the appropriate weighted average. REINFORCE and Kimura et. al.'s algorithm have a nice way of doing this and of efficiently taking care of the bookkeeping, by making use of eligibility traces. If indeed $\left|\frac{1}{(\pi_{\theta_0}(h_d))_j}\frac{d(\pi_{\theta_0}(h_d))_j}{d\theta}\right|$ were bounded, then their eligibility trace ideas (coupled with the the undiscounted POMDP ideas to ensure finite termination) could give another viable gradient ascent algorithm.

C Appendix

In this Appendix we give a transformation of an POMDP with multiple (but finite) actions to a POMDP with binary actions. This reduction can be used to immediately extend our results for binary action POMDPs to multiple actions POMDPs.

Consider a multiple action POMDP $M = (S, A, s_0, P, R, Q)$, where S is the set of states, A is the set of actions, s_0 is the start state, P is the transition probability distribution, R is the reward function and Q is the observable distribution. For simplicity we assume that $A = \{0,1\}^k$. Given M we define a new POMDP $M' = (S', A', s_0', P', R', Q')$, which we call the equivalent POMDP of M.

We start by describing the components of M'. Since M' is limited to binary actions, we set $A' = \{0,1\}$. Intuitively, in the transformation we divide a single k-bit action in M to k single bit actions in M'. In M' we wait till we have all the k bits of the action, while recording them in the state information. For this reason we set $S' = S \times B$ where $B = \cup_{i=0}^{k-1}\{0,1\}^i$. A state $s \in S$ is matched to a state $[s, \epsilon] \in S'$. The new start state $s_0' = [s_0, \epsilon]$.

The new transition function P' is constructed following our idea that each k-bit action in M is mapped to a sequence of k single bit actions in M'. The transition function is constructed such that it enables us to store the single bit actions in M' until we have all k bits. This part of the transition function is deterministic. Once we have all the k bits, we perform the same transition as in M. More formally, let $x \in B$ be a string such that $|x| \leq k-2$. For any $b \in \{0,1\}$ and any $s \in S$ we have that $P'_{[s,x],b}([s, xb]) = 1$ (and zero for any state other than $[s, xb]$). This implies that until we have all the k bits of the action we store them in the state information, and make deterministic transitions. Once we have a string of length $k-1$, when we perform another action in M' we have completely specified a k-bit action in M, and we can execute it. Formally, let $y \in B$ be a string such that $|y| = k-1$. For any $b \in \{0,1\}$ and any $s \in S$ we have that $P'_{[s,y],b}([s', \epsilon]) = P_{s,yb}(s')$.

Our construction of P' ensures that there is a matching between executions in M and M' (when considering in M' executions that end in some $[s, \epsilon]$). This matching is not only a one-to-one mapping, but also the two matched trajectories have the same transition probabilities. (The transition probability of a trajectory is the product of all the local transition probabilities, i.e., $\prod P_{s_i,a_i}(s_{i+1})$.)

The reward function R' is set to zero in 'transient' states, and to R at states that match states in S. More precisely, $R'_{[s,y],b} = R_{s,yb}$, where $y \in \{0,1\}^{k-1}$, and $R'_{[s,x],b} = 0$, where $x \in \{0,1\}^i$ and $i \leq k-2$.

The observable function is set such that it gives no information at 'transient' states. More formally, $Q'(*|[s,x]) = 1$ for any $x \neq \epsilon$ and $Q'(o|[s,\epsilon]) = Q(o|s)$.

It is easy to see that there is a matching between the trajectories generated in M and M', and the trajectories have the same transition probabilities. Let t' be a trajectory in M'. We can decompose t' to sequences of the form,

$$([s,\epsilon], b_1, 0, o), ([s, b_1], b_2, 0, *), \ldots, ([s, b_1 \cdots b_{k-1}], b_k, r, *).$$

Each such sequence we map to $(s, b_1 \cdots b_k, r, o)$ in M. By mapping each such sequence we map the trajectory t' to a trajectory t in M. It is easy to see that the transition probability of t in M and of t' in M' is identical. Also note that the inverse mapping, from trajectories t in M to trajectories t' in M', is unique. When we refer to this mapping we say that t and t' are matched.

Lemma C.1 *For any POMDP $M = (S, A, s_0, P, R, Q)$ there exists an POMDP $M' = (S', A', s'_0, P', R', Q')$ such that $A' = \{0, 1\}$, $|S'| \leq 2|S| \cdot |A|$, and there is a matching between trajectories in M and in M' such that the transition probability of matched trajectories is identical. (We say that M' is equivalent to M.)*

We have showed that for a POMDP with multiple actions there exists an equivalent POMDP that uses only binary actions. Now we need to show that we can also transform the strategy from one POMDP to the other POMDP. Given a strategy π for M we construct an equivalent strategy π' on M'. If π executes action $b_1 \cdots b_k \in A$ after observing trajectory h, then π' executes a sequence of k actions b_1 to b_k, after observing trajectory h', where h' is the trajectory matched to h. We call π' the strategy that is equivalent to π.

Lemma C.2 *Let M and M' be equivalent POMDPs. Let π and π' be equivalent strategies. Let t be any trajectory in M and t' its matched trajectory in M'. The probability of t under π in M is identical to the probability of t' under π' in M'.*

Now we need to show that the return can also be maintained. Here we show that it holds for two important return criteria: the undiscounted bounded horizon return and the discounted infinite horizon return. We modify the parameters so that the return of matched trajectories is identical.

For the bounded horizon return we have a parameter H which is the horizon in M. When running M' we use a horizon parameter of $H' = kH$. It is easy to see that matched trajectories have identical return.

For infinite horizon discounted return we have a parameter γ for the discounting in M. For M' we define $\gamma' = \gamma^{1/k}$. Again it is easy to see that the ratio between the return of matched trajectories is $(\gamma')^{k-1}$ (because the first reward is delayed by $k-1$ steps). If desired this difference can be be eliminated by rescaling R' by a γ^{k-1} factor.

Lemma C.3 *For any strategy class Π let Π' include all the strategies π' which are equivalent to some $\pi \in \Pi$. Let M and M' be equivalent POMDPs. For any $\pi \in \Pi$ and its equivalent strategy $\pi' \in \Pi'$ the expected return of π in M is, possibly up to a multiplicative constant, identical to the expected return of π' in M', when the return function is either undiscounted finite horizon or discounted infinite horizon.*

We would like to note that the transformation can be applied also in the average-return undiscounted infinite horizon case.