Approximate Planning in Large POMDPs via Reusable Trajectories

Michael Kearns * AT&T Labs
Yishay Mansour † AT&T Labs
Andrew Y. Ng UC Berkeley

January 27, 1999

KEYWORDS: Reinforcement Learning, POMDPs, Planning, VC Dimension

Abstract

We consider the problem of choosing a near-best strategy from a restricted class of strategies Π in a partially observable Markov decision process (POMDP). 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 the class 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, but depend only on the complexity of the restricted strategy class Π. The main challenge is in generating trajectories in the POMDP that can be *reused*, in the sense that they simultaneously provide estimates of the return of *many* strategies in the class.

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 reusable trajectories.

*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]) 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 a description of a POMDP M and 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).

Our notion of the manner in which the given POMDP M is "described" is simulative rather than representational. That is, rather than assume that M is specified in some particular representation, 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. Our generative models provides 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. Thus, results obtained via a generative model blur the distinction between what is typically called "planning" and "learning" in POMDPs. However, we will continue to refer to our problem as one of (approximate) planning to emphasize that we are given a simulator for the POMDP that is more powerful that what one typically has in the "pure learning" scenario.

The question of how many calls to a generative model are required to choose a near-best strategy in a given class 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 rather 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, and depends only linearly on a general measure of the *complexity* of the strategy class Π. We prove that these generated trajectories are enough to give us accurate estimates of the expected return of any strategy in Π. 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 only on the dimension of the class.

Our main contributions are:

- Giving specific methods for generating "resuable" trajectories from generative models;
- Proving that these methods allow the generation of a small number 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)** *consists of a set of* **states** S *and with* **actions** $\{a_1, \dots, a_k\}$ *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 $\sum_{s'} P(s'|s, a) = 1$ ($\int_S P(s'|s, a)ds' = 1$ 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*.*

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.

We will primarily be interested in POMDPs with a 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*.

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

[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).

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 4, where they are replaced by an extremely weak form of simulation — namely, a subroutine for generating only the observable history along truly random trajectories.

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:

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

where the last entry indicates that observable histories always conclude with the observation made at the final state reached. Such observable histories 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 class of strategies.*

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.

We will restrict our attention to the case of discounted return [2], so we assume we are given the **discount factor** $0 \leq \gamma < 1$, which then determines the **value function** V^π for any strategy π:

$$V^\pi(s) = \mathbf{E}\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_i \middle| s, \pi\right] \tag{1}$$

where r_i is the reward received on the ith step when executing π starting from state s, and the expectation is over the transition and observation probabilities (and any randomization in π). Note that for any s and any π, $V^\pi(s) \leq V_{max}$, where we define $V_{max} = R_{max}/(1-\gamma)$. We define the **$\epsilon$-horizon time** $H_\epsilon = \log_\gamma(\epsilon/(2V_{max}))$. Note that ignoring all the rewards after the ϵ-horizon time can decrease the discounted cumulative reward by at most ϵ.

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 pairs of strategy may now be incomparable. We thus adopt the common assumption of a fixed start state s_0 in the underlying MDP. (This is also equivalent to assuming a fixed distribution D over start states, since s_0 can be a "dummy" state whose next-state distribution under any action is D.) 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*

$$opt(M, \Pi) = \sup_{\pi \in \Pi} V^\pi(s_0) \tag{2}$$

where $V^\pi(s_0)$ is the expected discounted return of π from s_0. (Dependence of $opt(M,\Pi)$ on s_0 is suppressed for notational brevity.)

With these definitions, we can now state our problems 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 (that is, from which state-action pairs) should we make to the generative model in order to minimize the number of calls required?

[2] However, most of our results have straightforward generalizations to the undiscounted finite-horizon case for any fixed horizon H.

3 The Trajectory Tree Method

We now describe the first of our two methods for creating "reusable" trajectories from a generative model. 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.

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, so the total size (number of nodes) in each trajectory tree will be 2^{H_ϵ}.

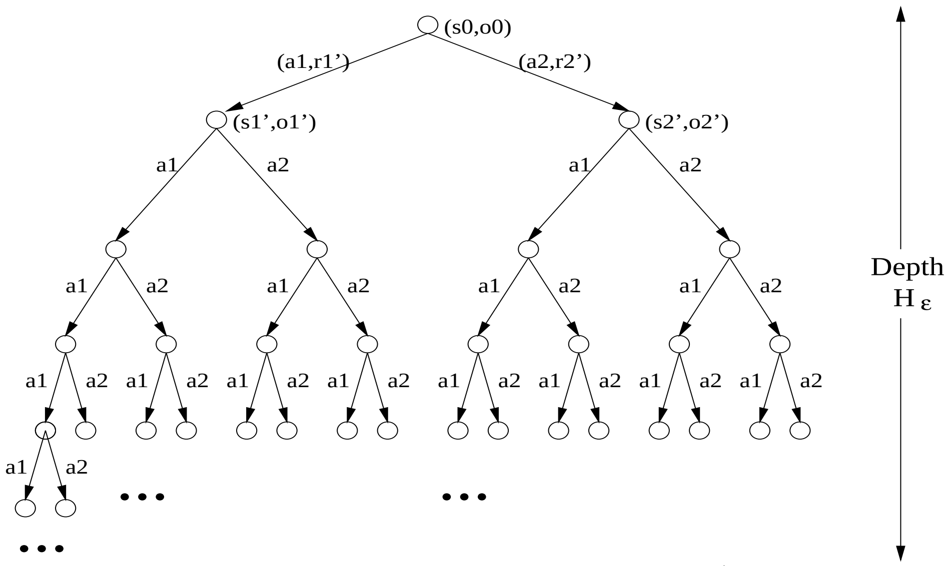

Figure 1: The structure of a typical trajectory tree. (Shown here with actions a_1 and a_2, and with observation, state, and reward labels omitted below the second level.)

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, 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 discounted sum of rewards 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 discounted sum of rewards 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

$$\hat{V}^\pi(s_0) = (1/m) \sum_{i=1}^{m} R(\pi, T_i). \tag{3}$$

The main goal of this section is 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 a *uniform convergence* theorems.

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*

$$m = O((V_{\max}/\epsilon)^2 \log(n/\delta)) \tag{4}$$

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 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 a 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 the estimate from the value of $V^{\pi}(s_0)$. 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 $e^{-\epsilon^2 m/(4{V_{max}}^2)}$.

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 *some* $\pi \in \Pi$ deviates by more than $\epsilon/2$ is bounded by $ne^{-\epsilon^2 m/(4{V_{max}}^2)} = \delta$. Note that if all of the estimates are within $\epsilon/2$ from the true value, then the policy with the highest estimate can be at most ϵ from the optimal value. $\square$

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 Π

Let us now move on to consider the general case of infinite strategy classes. We do not have space to provide details of the analysis; here we simply sketch some of the highlights, draw the connections with supervised learning, and state the resulting uniform convergence theorem.

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 for natural classes. 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 $m > d$. 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.

For real-valued classes $\mathcal{H}$, the notions of "shattering" or "exhaustive" behavior on a finite set of points must already be generalized in a way that is beyond the scope of this paper [Hau92]. But by analogy, in the full paper we define a notion of a set $T_1, \ldots, T_d$ of *trajectory trees* being "shattered" by the strategy class Π. Intuitively, this is a measure of how complex or exhaustive the set of "labelings" $\{\langle R(\pi, T_1), \ldots, R(\pi, T_d)\rangle : \pi \in \Pi\}$ is. From this, we can define the *dimension* $dim(\Pi)$ to be the largest shattered set of trajectory trees. For most natural infinite parametric classes, $dim(\Pi)$ be will polynomially related to the number of parameters. Once this machinery is developed, we can prove a result generalizing Theorem 3.1 for infinite classes, with $dim(\Pi)$ playing the role of $\log(n)$.

In the interests of concreteness, we will now sketch the ideas behind a simple version of this general theorem, for infinite, two-action, deterministic strategy classes, which can be done by appealing only to the familiar VC dimension of boolean functions. Generalizations to multiple actions or stochastic strategies require the machinery of *combinatorial dimension* or *pseudo-dimension* [Hau92].

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 $dim(\Pi)$ to denote the VC dimension of the set of binary functions Π.

We now show intuitively why a strategy class Π of bounded VC dimension d cannot have induce exhaustive behavior on a set $T_1, \ldots, T_m$ of trajectory trees for $m >> d$. 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 $R(\pi_1, T_i) \neq R(\pi_2, T_i)$ 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 $dim(\Pi)$ 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*

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

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

3.3 Practical 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, parameterized by the 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 3.3 *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) or Theorem 3.2 (infinite 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:

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

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 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. In particular, suppose we perform a greedy local search over a class Π of deterministic strategies. 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 our local search is currently considering. 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, since stochastic strategies define a distribution over *all* the paths in a trajectory tree, and thus evaluation of the current π requires examining entire trees. However, suppose $\Pi = \{\pi_\theta : \theta \in \Re^d\}$ 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 only linear dependence on H_ϵ. This stochastic gradient ascent algorithm works by only *subsampling* the trajectory trees, again permitting lazy construction. The update made to the current position θ_0 at each step will be an unbiased estimate of the gradient $(d/d\theta)\hat{V}^{\pi_\theta}(s_0)$ evaluated at θ_0. (Details in the full paper.)

Having come this far, there is also a simple further modification to the subsampling algorithm that leads us close to a line of research pursued by Kimura, Yamamura and Kobayashi [KYK95], and which gives a procedure also bearing some similarity to William's REINFORCE algorithm [Wil92]. 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 $V^{\pi_\theta}(s_0)$. In the full paper, 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) Has expected running time $O(1/(1-\gamma))$;
- (Unbiasedness) Outputs an unbiased estimate of $(d/d\theta)V^{\pi_\theta}(s_0)$;
- (Bounded Variance) The 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 $V^{\pi_\theta}(s_0)$. 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. Details, including discussion of exploration and of gradient ascent methods for learning deterministic policies, are in the long version of this paper.

4 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 POMDP M, with designated start state s_0, generates an observable history of a given length t starting from s_0 by following the truly random policy (at each state each action is equally likely). Thus, the generator outputs only the observable history*

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

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 of our random trajectories to be the ϵ-horizon time H_ϵ.

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 is 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.

We again restrict our attention to the case of deterministic strategies, solely for expository purposes. Recall that in this case, each trajectory tree T allowed us to get an evaluation of *any* 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 h, given that π may diverge from t? 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 $\mathrm{acc}_\pi(h) = 1$.

We can now define the estimate of $V^\pi(s_0)$ we derive from a set of random trajectories. Let $\mathcal{T} = \{h_1, \ldots, h_m\}$ 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 $S_\pi(\mathcal{T}) \subseteq \mathcal{T}$ include all the histories h for which $\mathrm{acc}_\pi(h) = 1$; thus,

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

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

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

where $R(h)$ is the discounted sum of rewards along h. We note that the generalization of this estimate to the case of stochastic strategies can be viewed as a form of importance sampling [SB98].

Thus, in analogy with Section 3, we now have a method of generating a set of m random observable histories $\mathcal{T} = \{h_1, \ldots, h_m\}$ (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 $\mathcal{T}$. 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 $\pi \in \Pi$. Of course, now the concern is that unless m is very large, some set $S_\pi(\mathcal{T})$ may be too small to obtain a good estimate. The following theorem, which is the analogue of Theorem 3.2, asserts that this is not the case.

Theorem 4.1 *Let Π be any class of deterministic strategies in a two-action POMDP M, and let $dim(\Pi)$ be the VC dimension of Π. Let m observable histories be generated from the random trajectory generator for M, and let $\hat{V}^\pi(s_0)$ be the resulting estimates. If*

$$m = O\left(\left(\frac{2^{H_\epsilon} V_{\max}}{\epsilon}\right)^2 (dim(\Pi) \log(H_\epsilon)(H_\epsilon + \log(V_{\max}/\epsilon)) + \log(1/\delta))\right) \tag{11}$$

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

If we compare Theorem 4.1 to Theorem 3.2, 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. As with the trajectory tree methods, a class of algorithms for approximate planning based on the random trajectory method can now be formulated.

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.

[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. 1999. Unpublished manuscript.

[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.

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

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