A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes

Michael Kearns[*]
Syntek Capital
mkearns@cis.upenn.edu

Yishay Mansour
Tel Aviv University
mansour@math.tau.ac.il

Andrew Y. Ng
UC Berkeley
ang@cs.berkeley.edu

October 3, 2001

Abstract

A critical issue for the application of Markov decision processes (MDPs) to realistic problems is how the complexity of planning scales with the size of the MDP. In stochastic environments with very large or infinite state spaces, traditional planning and reinforcement learning algorithms may be inapplicable, since their running time typically grows linearly with the state space size in the worst case. In this paper we present a new algorithm that, given only a *generative model* (a natural and common type of simulator) for an arbitrary MDP, performs on-line, near-optimal planning with a per-state running time that has *no dependence* on the number of states. The running time is exponential in the *horizon time* (which depends only on the discount factor γ and the desired degree of approximation to the optimal policy). Our algorithm thus provides a different complexity trade-off than classical algorithms such as value iteration — rather than scaling linearly in both horizon time and state space size, our running time trades an exponential dependence on the former in exchange for no dependence on the latter.

Our algorithm is based on the idea of *sparse sampling*. We prove that a randomly sampled look-ahead tree that covers only a vanishing fraction of the full look-ahead tree nevertheless suffices to compute near-optimal actions from any state of an MDP. Practical implementations of the algorithm are discussed, and we draw ties to our related recent results on finding a near-best strategy from a given class of strategies in very large partially observable MDPs [KMN00].

1 Introduction

In the past decade, Markov decision processes (MDPs) and reinforcement learning have become a standard framework for planning and learning under uncer-

[*]This research was conducted while the author was at AT&T Labs.

tainty within the artificial intelligence literature. The desire to attack problems of increasing complexity with this formalism has recently led researchers to focus particular attention on the case of (exponentially or even infinitely) large state spaces. A number of interesting algorithmic and representational suggestions have been made for coping with such large MDPs. Function approximation [SB98] is a well-studied approach to learning value functions in large state spaces, and many authors have recently begun to study the properties of large MDPs that enjoy compact representations, such as MDPs in which the state transition probabilities factor into a small number of components [BDG95, MHK$^+$98, KP99].

In this paper, we are interested in the problem of computing a near-optimal policy in a large or infinite MDP that is given — that is, we are interested in *planning*. It should be clear that as we consider very large MDPs, the classical planning assumption that the MDP is given explicitly by tables of rewards and transition probabilities becomes infeasible. One approach to this representational difficulty is to assume that the MDP has some special structure that permits compact representation (such as the factored transition probabilities mentioned above), and to design special-purpose planning algorithms that exploit this structure.

Here we take a slightly different approach. We consider a setting in which our planning algorithm is given access to a *generative model*, or simulator, of the MDP. Informally, this is a "black box" to which we can give any state-action pair (s, a), and receive in return a randomly sampled next state and reward from the distributions associated with (s, a). Generative models have been used in conjunction with some function approximation schemes [SB98], and are a natural way in which a large MDP might be specified. Moreover, they are more general than most structured representations, in the sense that many structured representations (such as factored models [BDG95, MHK$^+$98, KP99]) usually provide an efficient way of implementing a generative model. Note also that generative models also provide less information than explicit tables of probabilities, but more information than a single continuous trajectory of experience generated according to some exploration policy, and so we view results obtained via generative models as blurring the distinction between what is typically called "planning" and "learning" in MDPs.

Our main result is a new algorithm that accesses the given generative model to perform near-optimal planning in an on-line fashion. By "on-line," we mean that, similar to real-time search methods [Kor90, BBS95, KS98], our algorithm's computation at any time is focused on computing an actions for a single "current state," and planning is interleaved with taking actions. More precisely, given any state s, the algorithm uses the generative model to draw samples for many state-action pairs, and uses these samples to compute a near-optimal action from s, which is then executed. The amount of time required to compute a near-optimal action from any particular state s has *no dependence* on the number of states in the MDP, even though the next-state distributions from s may be very diffuse (that is, have large support). The key to our analysis is in showing that appropriate sparse sampling suffices to construct enough information about the environment near s to compute a near-optimal action. The analysis relies on a combination of Bellman equation calculations, which are standard in reinforcement learning, and uniform convergence arguments, which are standard in supervised learning; this combination of techniques was first applied in [KS99]. As mentioned, the running time required at each state does have an exponential dependence on the horizon time, which we show to be unavoidable without further assumptions. However, our results leave open the possiblity of an algorithm that runs in time polynomial in the accuracy parameter, which remains an important open problem.

Note that one can view our planning algorithm as simply implementing a (stochastic) policy — a policy that happens to use a generative model as a subroutine. In this sense, if we view the generative model as providing a "compact" representation of the MDP, our algorithm provides a correspondingly "compact" representation of a near-optimal policy. We view our result as complementary to work that proposes and exploits particular compact representations of MDPs [MHK$^+$98], with both lines of work beginning to demonstrate the potential feasibility of planning and learning in very large environments.

The remainder of this paper is structured as follows: In Section 2, we give the formal definitions needed in this paper. Section 3 then gives our main result, an algorithm for planning in large or infinite MDPs, whose per-state running time does not depend on the size of the state space. Finally, Section 4 describes related results and open problems.

2 Preliminaries

We begin with the definition of a Markov decision process on a set of $N = |S|$ states, explicitly allowing the possibility of the number of states being (countably or uncountably) infinite.

Definition 1 *A* **Markov decision process** M *on a set of* **states** S *and with* **actions** $\{a_1, \ldots, a_k\}$ *consists of:*

- **Transition Probabilities***: For each state-action pair* (s, a)*, a next-state distribution* $P_{sa}(s')$ *that specifies the probability of transition to each state* s' *upon execution of action* a *from state* s*.*[1]

- **Reward Distributions***: For each state-action pair* (s, a)*, a distribution* R_{sa} *on real-valued* **rewards** *for executing action* a *from state* s*. We assume rewards are bounded in absolute value by* $R_{\max}$*.*

[1]Henceforth, everything that needs to be measurable is assumed to be measurable.

For simplicity, we shall assume in this paper that all rewards are in fact deterministic — that is, the reward distributions have zero variance, and thus the reward received for executing a from s is always exactly R_{sa}. 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.

Throughout the paper, we will primarily be interested in MDPs with a very large (or even infinite) number of states, thus precluding approaches that compute directly on the full next-state distributions. Instead, we will assume that our planning algorithms are given 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 Markov decision process M is a randomized algorithm that, on input of a state-action pair (s, a), outputs R_{sa} and a state s', where s' is randomly drawn according to the transition probabilities $P_{sa}(\cdot)$.*

We think of a generative model as falling somewhere in between being given explicit next-state distributions, and being given only "irreversible" experience in the MDP (in which the agent follows a single, continuous trajectory, with no ability to reset to any desired state). On the one hand, a generative model may often be available when explicit next-state distributions are not; on the other, a generative model obviates the important issue of exploration that arises in a setting where we only have irreversible experience. In this sense, planning results using generative models blur the distinction between what is typically called "planning" and what is typically called "learning".

Following standard terminology, we define a (stochastic) **policy** to be any mapping $\pi : S \mapsto \{a_1, \ldots, a_k\}$. Thus $\pi(s)$ may be a random variable, but depends only on the current state s. We will be primarily concerned with discounted MDPs,[2] so we assume we are given a number $0 \leq \gamma < 1$ called the **discount factor**, with which we then define the **value function** V^π for any policy π:

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

where r_i is the reward received on the ith step of executing the policy π from state s, and the expectation is over the transition 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 also define the Q**-function** for a given policy π as

$$Q^\pi(s,a) = R_{sa} + \gamma \mathbf{E}_{s' \sim P_{sa}(\cdot)}\left[V^\pi(s')\right] \quad (2)$$

[2]However, our results can be generalized to the undiscounted finite-horizon case for any fixed horizon H [MS99a].

(where the notation $s' \sim P_{sa}(\cdot)$ means that s' is drawn according to the distribution $P_{sa}(\cdot)$). We will later describe an algorithm $\mathcal{A}$ that takes as input any state s and (stochastically) outputs an action a, and which therefore implements a policy. When we have such an algorithm, we will also write $V^{\mathcal{A}}$ and $Q^{\mathcal{A}}$ to denote the value function and Q-function of the policy implemented by $\mathcal{A}$. Finally, we define the optimal value function and the optimal Q-function as $V^*(s) = \sup_\pi V^\pi(s)$ and $Q^*(s,a) = \sup_\pi Q^\pi(s,a)$, and the **optimal policy** π^*, $\pi^*(s) = \arg\max_a Q^*(s,a)$ for all $s \in S$.

3 Planning in Large or Infinite MDPs

Usually, one considers the *planning* problem in MDPs to be that of computing a good policy, given as input the transition probabilities $P_{sa}(\cdot)$ and the rewards R_{sa} (for instance, by solving the MDP for the optimal policy). Thus, the input is a complete and exact model, and the output is a total mapping from states to actions. Without additional assumptions about the structure of the MDP, such an approach is clearly infeasible in very large state spaces, where even reading all of the input can take N^2 time, and even specifying a general policy requires space on the order of N. In such MDPs, a more fruitful way of thinking about planning might be an *on-line* view, in which we examine the *per-state* complexity of planning. Thus, the input to a planning algorithm would be a single state, and the output would be which single action to take from that state. In this on-line view, a planning algorithm is itself simply a policy (but one that may need to perform some nontrivial computation at each state).

Our main result is the description and analysis of an algorithm $\mathcal{A}$ that, given access to a generative model for an arbitrary MDP M, takes any state of M as input and produces an action as output, and meets the following performance criteria:

- The policy implemented by $\mathcal{A}$ is near-optimal in M;
- The running time of $\mathcal{A}$ (that is, the time required to compute an action at any state) has *no dependence* on the number of states of M.

This result is obtained under the assumption that there is an $O(1)$ time and space way to refer to the states, a standard assumption known as the *uniform cost model* [AHU74], that is typically adopted to allow analysis of algorithms that operate on real numbers (such as we require to allow infinite state spaces). The uniform cost model essentially posits the availability of infinite-precision registers (and constant-size circuitry for performing the basic arithmetic operations on these registers). If one is unhappy with this model, then algorithm $\mathcal{A}$ will suffer a dependence on the number of states only equal to the space required to name the states (at worst $\log(N)$ for N states).

3.1 A Sparse Sampling Planner

Here is our main result:

Theorem 1 *There is a randomized algorithm $\mathcal{A}$ that, given access to a generative model for any k-action MDP M, takes as input any state $s \in S$ and any value $\varepsilon > 0$, outputs an action, and satisfies the following two conditions:*

- *(Efficiency) The running time of $\mathcal{A}$ is $O((kC)^H)$, where*

$$H = \lceil \log_\gamma(\lambda/V_{\max}) \rceil ,$$

$$C = \frac{{V_{\max}}^2}{\lambda^2}\left(2H \log \frac{kH{V_{\max}}^2}{\lambda^2} + \log \frac{R_{\max}}{\lambda}\right),$$

$$\lambda = (\epsilon(1-\gamma)^2)/4,\ V_{\max} = R_{\max}/(1-\gamma).$$

  In particular, the running time depends only on $R_{\max}$, γ, and ε, and does not *depend on $N = |S|$. If we view $R_{\max}$ as a constant, the running time bound can also be written*

$$\left(\frac{k}{\varepsilon(1-\gamma)}\right)^{O\left(\frac{1}{1-\gamma}\log\left(\frac{1}{\varepsilon(1-\gamma)}\right)\right)}. \tag{3}$$

- *(Near-Optimality) The value function of the stochastic policy implemented by $\mathcal{A}$ satisfies*

$$|V^{\mathcal{A}}(s) - V^*(s)| \leq \varepsilon \tag{4}$$

  simultaneously for all states $s \in S$.

As we have already suggested, it will be helpful to think of algorithm $\mathcal{A}$ in two different ways. On the one hand, $\mathcal{A}$ is an algorithm that takes a state as input and has access to a generative model, and as such we shall be interested in its resource complexity — its running time, and the number of calls it needs to make to the generative model (both per state input). On the other hand, $\mathcal{A}$ produces an action as output in response to each state given as input, and thus implements a (possibly stochastic) *policy*.

The proof of Theorem 1 is given in Appendix A, and detailed pseudo-code for the algorithm is provided in Figure 1. We now give some high-level intuition for the algorithm and its analysis.

Given as input a state s, the algorithm must use the generative model to find a near-optimal action to perform from state s. The basic idea of the algorithm is to sample the generative model from states in the "neighborhood" of s. This allows us to construct a *small* "sub-MDP" M' of M such that the optimal action in M' from s is a near-optimal action from s in M.[3] There will be no guarantee

[3] M' will not literally be a sub-MDP of M, in the sense of being strictly embedded in M, due to the variations of random sampling. But it will be very "near" such an embedded MDP.

```
Function: EstimateQ$(h, C, \gamma, G, s)$
Input: depth $h$, width $C$, discount $\gamma$, generative model $G$, state $s$.
Output: A list $(\hat{Q}^*_h(s,a_1), \hat{Q}^*_h(s,a_2), \ldots, \hat{Q}^*_h(s,a_k))$, of estimates of the $Q^*(s,a_i)$.
  1. If $h = 0$, return $(0, \ldots, 0)$.
  2. For each $a \in A$, use $G$ to generate $C$ samples from the next-state distribution $P_{sa}(\cdot)$. Let
     $S_a$ be a set containing these $C$ next-states.
  3. For each $a \in A$ and let our estimate of $Q^*(s,a)$ be
         $\hat{Q}^*_h(s,a) = R(s,a) + \gamma \frac{1}{C} \sum_{s' \in S_a} \textbf{EstimateV}(h-1, C, \gamma, G, s').$   (5)
  4. Return $(\hat{Q}^*_h(s,a_1), \hat{Q}^*_h(s,a_2), \ldots, \hat{Q}^*_h(s,a_k))$.

Function: EstimateV$(h, C, \gamma, G, s)$
Input: depth $h$, width $C$, discount $\gamma$, generative model $G$, state $s$.
Output: A number $\hat{V}^*_h(s)$ that is an estimate of $V^*_h(s)$.
  1. Let $(\hat{Q}^*_h(s,a_1), \hat{Q}^*_h(s,a_2), \ldots, \hat{Q}^*_h(s,a_k)) :=$ EstimateQ$(h, C, \gamma, G, s)$.
  2. Return $\max_{a \in \{a_1, \ldots, a_k\}} \{\hat{Q}^*_h(s,a)\}$.

Function: Algorithm $\mathcal{A}(\epsilon, \gamma, R_{max}, G, s_0)$
Input: tolerance $\epsilon$, discount $\gamma$, max reward $R_{max}$, generative model $G$, state $s_0$.
Output: An action $a$.
  1. Let the required horizon $H$ and width $C$ parameters be calculated as given as functions of
     $\epsilon$, $\gamma$ and $R_{max}$ in Theorem1.
  2. Let $(\hat{Q}^*_H(s,a_1), \hat{Q}^*_H(s,a_2), \ldots, \hat{Q}^*_H(s,a_k)) :=$ EstimateQ$(H, C, \gamma, G, s_0)$.
  3. Return $\arg\max_{a \in \{a_1, \ldots, a_k\}} \{\hat{Q}^*_H(s,a)\}$.
```

Figure 1: Algorithm $\mathcal{A}$ for planning in large or infinite state spaces. **EstimateV** finds the $\hat{V}^*_h$ described in the text, and **EstimateQ** finds analogously defined $\hat{Q}^*_h$. Algorithm $\mathcal{A}$ implements the policy.

that M' will contain enough information to compute a good action from any state other than s. However, in exchange for this limited applicability, the MDP M' will have a number of states that does not depend on the number of states in M.

The graphical structure of M' will be given by a *directed tree* in which each node is labeled by a state, and each directed edge to a child is labeled by an action and a reward. For the sake of simplicity, let us consider only the two-action case here, with actions a_1 and a_2. Each node will have C children in which the edge to the child is labeled a_1, and C children in which the edge to the child is labeled a_2.

The root node of M' is labeled by the state of interest s, and we generate the $2C$ children of s in the obvious way: we call the generative model C times on the state-action pair (s, a_1) to get the a_1-children, and on C times on (s, a_2) to get the a_2-children. The edges to these children are also labeled by the rewards returned by the generative model, and the child nodes themselves are labeled by the states returned. We will build this $(2C)$-ary tree to some depth to be determined. Note that M' is essentially a *sparse look-ahead tree*.

We can also think of M' as an MDP in which the start state is s, and in which taking an action from a node in the tree causes a transition to a (uniformly) random child of that node with the corresponding action label; the childless leaf nodes are considered absorbing states. Under this interpretation, we can compute the optimal action to take from the root s in M'. Figure 2 shows a conceptual picture of this tree for a run of the algorithm from an input state s_0, for $C = 3$. (C will typically be much larger.) From the root s_0, we try action a_1 three times and action a_2 three times. From each of the resulting states, we also try each action C times, and so on down to depth H in the tree. Zero values assigned to the leaves then correspond to our estimates of $\hat{V}_0^*$, which are "backed-up" to find estimates of $\hat{V}_1^*$ for their parents, which are in turn backed-up to their parents, and so on, up to the root to find an estimate of $\hat{V}_H^*(s_0)$.

The central claim we establish about M' is that its size can be independent of the number of states in M, yet still result in our choosing near-optimal actions at the root. We do this by establishing bounds on the required depth H of the tree and the required degree C.

Recall that the optimal policy at s is given by $\pi^*(s) = \arg\max_a Q^*(s, a)$, and therefore is completely determined by, and easily calculated from, $Q^*(s, \cdot)$. Estimating the Q-values is a common way of planning in MDPs. ¿From the standard duality between Q-functions and value functions, the task of estimating Q-functions is very similar to that of estimating value functions. So while the algorithm uses the Q-function, we will, purely for expository purposes, actually describe here how we estimate $V^*(s)$.

There are two parts to the approximation we use. First, rather than estimating V^*, we will actually estimate, for a value of H to be specified later, the H-step expected discounted reward $V_H^*(s)$, given by

$$V_h^*(s) = \mathbf{E}\left[\sum_{i=1}^{h} \gamma^{i-1} r_i \,\middle|\, s, \pi^* \right] \tag{6}$$

where r_i is the reward received on the ith time step upon executing the optimal policy π^* from s. Moreover, we see that the $V_h^*(s)$, for $h \geq 1$, are recursively given by

$$\begin{aligned} V_h^*(s) &= R_{sa^*} + \gamma \mathbf{E}_{s' \sim P_{sa^*}(\cdot)}[V_{h-1}^*(s')] \\ &\approx \max_a \{ R_{sa} + \gamma \mathbf{E}_{s' \sim P_{sa}(\cdot)}[V_{h-1}^*(s')] \} \end{aligned} \tag{7}$$

where a^* is the action taken by the optimal policy from state s, and $V_0^*(s) = 0$. The quality of the approximation in Equation (7) becomes better for larger values of h, and is controllably tight for the largest value $h = H$ we eventually choose. One of the main efforts in the proof is establishing that the error incurred by the recursive application of this approximation can be made controllably small by choosing H sufficiently large.

Thus, if we are able to obtain an estimate $\hat{V}_{h-1}^*(s')$ of $V_{h-1}^*(s')$ for any s', we can inductively define an algorithm for finding an estimate $\hat{V}_h^*(s)$ of $V_h^*(s)$ by making use of Equation (7). Our algorithm will *approximate* the expectation in Equation (7) by a sample of C random next states from the generative model, where C is a parameter to be determined (and which, for reasons that will become clear later, we call the "width"). Recursively, given a way of finding the estimator $\hat{V}_{h-1}^*(s')$ for any s', we find our estimate $\hat{V}_h^*(s)$ of $V_h^*(s)$ as follows:

1. For each action a, use the generative model to get R_{sa} and to sample a set S_a of C independently sampled states from the next-state distribution $P_{sa}(\cdot)$.

2. Use our procedure for finding $\hat{V}_{h-1}^*$ to estimate $\hat{V}_{h-1}^*(s')$ for each state s' in any of the sets S_a.

3. Following Equation (7), our estimate of $V_h^*(s)$ is then given by

$$\hat{V}_h^*(s) = \max_a \left\{ R_{sa} + \gamma \frac{1}{C} \sum_{s' \in S_a} \hat{V}_{h-1}^*(s') \right\}. \tag{8}$$

To complete the description of the algorithm, all that remains is to choose the depth H and the parameter C, which controls the width of the tree. Bounding the required depth H is the easy and standard part. It is not hard to see that if we choose depth $H = \log_\gamma \epsilon(1-\gamma)/R_{max}$ (the so-called ϵ-*horizon time*), then the discounted sum of the rewards that is obtained by considering rewards beyond this horizon is bounded by ϵ.

The central claim we establish about C is that it can be chosen *independent* of the number of states in M, yet still result in choosing near-optimal actions at the root. The key to the argument is that even though small samples may give very poor approximations to the next-state distribution at each state in the tree, they will, nevertheless, give good estimates of the *expectation* terms of Equation (7), and that is really all we need. For this we apply a careful combination of uniform convergence methods and inductive arguments on the tree depth. Again, the technical details of the proof are in Appendix A.

In general, the resulting tree may represent only a vanishing fraction of all of the H-step paths starting from s_0 that have non-zero probability in the MDP — that is, the sparse look-ahead tree covers only a vanishing part of the

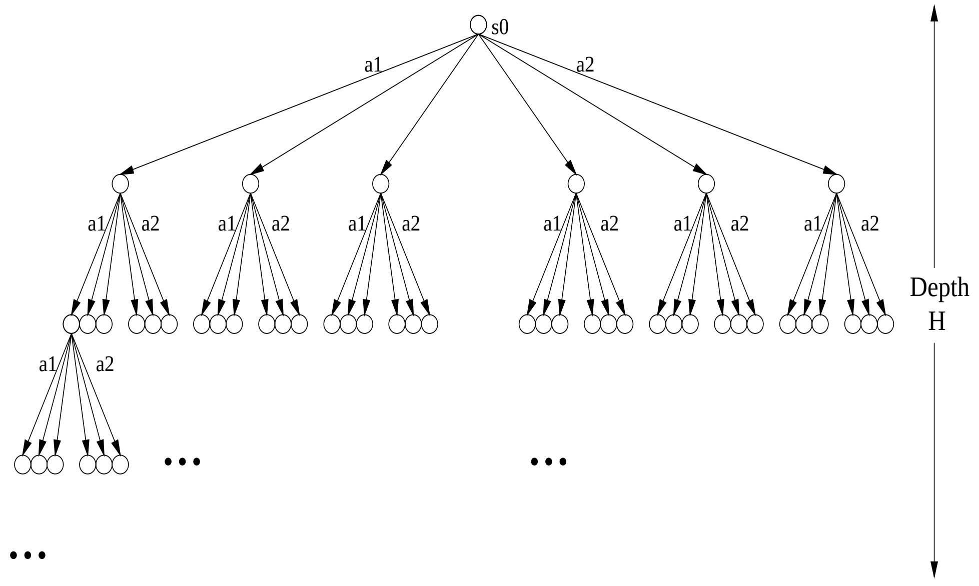

Figure 2: Sparse look-ahead tree of states constructed by the algorithm. (Shown with $C = 3$, actions a_1, a_2.)

full look-ahead tree. In this sense, our algorithm is clearly related to and inspired by classical look-ahead search techniques [RN95] including various real-time search algorithms [Kor90, BBS95, BLG97, KS98] and receding horizon controllers. Most of these classical search algorithms, however, run into difficulties in very large or infinite MDPs with diffuse transitions, since their search trees can have arbitrarily large (or even infinite) branching factors. Our main contribution is showing that in large stochastic environments, clever random sampling suffices to reconstruct nearly all of the information available in the (exponentially or infinitely) large full look-ahead tree. Note that in the case of deterministic environments, where from each state-action pair we can reach only a single next state, the sparse and full trees coincide (assuming a memoization trick described below), and our algorithm reduces to classical deterministic look-ahead search.

3.2 Practical Issues and Lower Bounds

Even though the running time of algorithm $\mathcal{A}$ does not depend on the size of the MDP, it still runs in time exponential in the ϵ-horizon time H, and therefore exponential in $1/(1-\gamma)$. It would seem that the algorithm would be practical only if γ is not too close to 1. In a moment, we will give a lower bound showing it is not possible to do much better without further assumptions on the MDP. Nevertheless, there are a couple of simple tricks that may help to reduce the running time in certain cases, and we describe these tricks first.

The first idea is to allow different amounts of sampling at each level of the tree. The intuition is that the further we are from the root, the less influence our estimates will have on the Q-values at the root (due to the discounting). Thus, we can sample more sparsely at deeper levels of the tree without having too adverse an impact on our approximation.

We have analyzed various schemes for letting the amount of sampling at a node depend on its depth. None of the methods we investigated result in a running time which is polynomial in $1/\epsilon$. However, one specific scheme that reduces the running time significantly is to let the number of samples per action at depth i be $C_i = \gamma^{2i}C$, where the parameter C now controls the amount of sampling done at the root. The error in the Q-values using such a scheme does not increase by much, and the running time is the square root of our original running time. Beyond this and analogous to how classical search trees can often be pruned in ways that significantly reduce running time, a number of standard tree pruning methods may also be applied to our algorithm's trees [RN95] (see also [DB94]), and we anticipate that this may significantly speed up the algorithm in practice.

Another way in which significant savings might be achieved is through the use of memoization in our subroutines for calculating the $\hat{V}_h^*(s)$'s. In Figure 2, this means that whenever there are two nodes at the same level of the tree that correspond to the same state, we collapse them into one node (keeping just one of their subtrees). While it is straightforward to show the correctness of such memoization procedures for deterministic procedures, one must be careful when addressing randomized procedures. We can show that the important properties of our algorithm are maintained under this optimization. Indeed, this optimization is particularly nice when the domain is actually deterministic: if each action deterministically causes a transition to a fixed next-state, then the tree would grow only as k^H (where k is the number of actions). If the domain is "nearly deterministic," then we have behavior somewhere in between. Similarly, if there are only some $N_0 \ll |S|$ states reachable from s_0, then the tree would also never grow wider than N_0, giving it a size of $O(N_0 H)$.

In implementing the algorithm, one may wish not to specify a targeted accuracy ϵ in advance, but rather to try to do as well as is possible with the computational resources available. In this case, an "iterative-deepening" approach may be taken. This would entail simultaneously increasing C and H by decreasing the target ϵ. Also, as studied in Davies, Ng and Moore [DNM98], if we have access to an initial estimate of the value function, we can replace our estimates $\hat{V}_0^*(s) = 0$ at the leaves with the estimated value function at those states. Though we shall not do so here, it is again easy to make formal performance guarantees depending on C, H and the supremum error of the value function estimate we are using.

Unfortunately, despite these tricks, it is not difficult to prove a lower bound that shows that any planning algorithm with access only to a generative model, and which implements a policy that is ϵ-close to optimal in a general MDP, must have running time at least exponential in the ϵ-horizon time. We now describe this lower bound.

Theorem 2 *Let $\mathcal{A}$ be any algorithm that is given access only to a generative model for an MDP M, and inputs s (a state in M) and ϵ. Let the stochastic policy implemented by $\mathcal{A}$ satisfy*

$$|V^{\mathcal{A}}(s) - V^*(s)| \leq \epsilon \tag{9}$$

simultaneously for all states $s \in S$. Then there exists an MDP M on which $\mathcal{A}$ makes at least $\Omega(2^H) = \Omega((1/\epsilon)^{(1/\log(1/\gamma))})$ calls to the generative model.

Proof: Let $H = \log_\gamma \epsilon = \log(1/\epsilon)/\log(1/\gamma)$. Consider a binary tree T of depth H. We use T to define an MDP in the following way. The states of the MDP are the nodes of the tree. The actions of the MDP are $\{0, 1\}$. When we are in state s and perform an action b we reach (deterministically) state s_b, where s_b is the b-child of s in T. If s is a leaf of T then we move to an absorbing state. We choose a random leaf v in the tree. The reward function for v and any action is R_{max}, and the reward at any other state and action is zero.

Algorithm $\mathcal{A}$ is given s_0, the root of T. For algorithm $\mathcal{A}$ to compute a near optimal policy, it has to "find" the node v, and therefore has to perform at least $\Omega(2^H)$ calls to the generative model. $\square$

4 Summary and Related Work

We have described an algorithm for near-optimal planning from a generative model, that has a per-state running time that does not depend on the size of the state space, but which is still exponential in the ϵ-horizon time. An important open problem is to close the gap between our lower and upper bound. Our lower bound shows that the number of steps has to grow polynomially in $1/\epsilon$ while in the upper bound the number of steps grows sub-exponentially in $1/\epsilon$, more precisely $(1/\epsilon)^{O(\log(1/\epsilon))}$. Closing this gap, either by giving an algorithm that would be polynomial in $1/\epsilon$ or by proving a better lower bound, is an interesting open problem.

Two interesting directions for improvement are to allow partially observable MDPs (POMDPs), and to find more efficient algorithms that do not have exponential dependence on the horizon time. As a first step towards both of these goals, in a separate paper [KMN00] we investigate a framework in which the goal is to use a generative model to find a near-best strategy within a restricted class of strategies for a POMDP. Typical examples of such restricted strategy classes include limited-memory strategies in POMDPs, or policies in large MDPs that implement a linear mapping from state vectors to actions. Our main result in this framework says that as long as the restricted class of strategies is not too "complex" (where this is formalized using appropriate generalizations of standard notions like VC dimension from supervised learning), then it is possible to find a near-best strategy from within the class, in time that again has no dependence on the size of the state space. If the restricted class of strategies is smoothly parameterized, then this further leads to a number of fast, practical algorithms for doing gradient descent to find the near-best strategy within the class, where the running time of each gradient descent step now has only linear rather than exponential dependence on the horizon time.

Another approach to planning in POMDPs that is based on the algorithm presented here is investigated by McAllester and Singh [MS99b], who show how the approximate belief-state tracking methods of Boyen and Koller [BK98] can be combined with our algorithm.

Acknowledgements

We give warm thanks to Satinder Singh for many enlightening discussions and numerous insights on the ideas presented here.

References

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[BBS95] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using Real-Time Dynamic Programming. *Artificial Intelligence*, 72:81–138, 1995.

[BDG95] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, 1995.

[BK98] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 1998 Conference on Uncertainty in Artificial Intelligence*. Morgan Kauffmann, 1998.

[BLG97] Blai Bonet, Góbor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the Fourteenth National Conference on Artifial Intelligence*, 1997.

[DB94] Richard Dearden and Craig Boutilier. Integrating planning and execution in stochastic domains. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence*, 1994.

[DNM98] Scott Davies, Andrew Y. Ng, and Andrew Moore. Applying online-search to reinforcement learning. In *Proceedings of AAAI-98*, pages 753–760. AAAI Press, 1998.

[KMN00] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large POMDPs via reusable trajectories. In *Neural Information Processing Systems 13*, (to appear) 2000.

[Kor90] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[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.

[KS98] Sven Koenig and Reid Simmons. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 1998.

[KS99] Michael Kearns and Satinder Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Neural Information Processing Systems 12*. MIT Press, 1999.

[MHK$^+$98] N. Meuleau, M. Hauskrecht, K-E. Kim, L. Peshkin, L.P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled Markov decision processes. In *Proceedings of AAAI*, pages 165–172, 1998.

[MS99a] D. McAllester and S. Singh. 1999. Personal Communication.

[MS99b] D. McAllester and S. Singh. Approximate planning for factored POMDPs using belief state simplification. 1999. Preprint.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

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

[SY94] Satinder Singh and Richard Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16:227–233, 1994.

Appendix A: Proof Sketch of Theorem 1

In this appendix, we give the proof of Theorem 1.

THEOREM 1 *There is a randomized algorithm $\mathcal{A}$ that, given access to a generative model for any k-action MDP M, takes as input any state $s \in S$ and any value $\varepsilon > 0$, outputs an action, and satisfies the following two conditions:*

- *(Efficiency) The running time of $\mathcal{A}$ is $O((kC)^H)$, where*

$$H = \lceil \log_\gamma(\lambda / V_{\max}) \rceil ,$$

$$C = \frac{{V_{\max}}^2}{\lambda^2}\left(2H \log \frac{kH{V_{\max}}^2}{\lambda^2} + \log \frac{R_{\max}}{\lambda}\right),$$

$$\lambda = (\epsilon(1-\gamma)^2)/4,$$

$$\delta = \lambda / R_{\max},$$

$$V_{\max} = R_{\max}/(1-\gamma).$$

  In particular, the running time depends only on $R_{\max}$, γ, and ε, and does not *depend on $N = |S|$. If we view $R_{\max}$ as a constant, the running time bound can also be written*

$$\left(\frac{k}{\varepsilon(1-\gamma)}\right)^{O\left(\frac{1}{1-\gamma}\log\left(\frac{1}{\varepsilon(1-\gamma)}\right)\right)} . \qquad (10)$$

- *(Near-Optimality) The value function of the stochastic policy implemented by $\mathcal{A}$ satisfies*

$$|V^{\mathcal{A}}(s) - V^*(s)| \leq \varepsilon \qquad (11)$$

  simultaneously for all states $s \in S$.

Throughout the analysis we will rely on the pseudo-code provided for algorithm $\mathcal{A}$ given in Figure 1.

The claim on the running time is immediate from the definition of algorithm $\mathcal{A}$. Each call to **EstimateQ** generates kC calls to **EstimateV**, C calls for each action. Each recursive call also reduces the depth parameter h by one, so the depth of the recursion is at most H. Therefore the running time is $O((kC)^H)$.

The main effort is in showing that the values of **EstimateQ** are indeed good estimates of Q^* for the chosen values of C and H. There are two sources of inaccuracy in these estimates. The first is that we use only a finite sample to approximate an expectation — we draw only C states from the next-state distributions. The second source of inaccuracy is that in computing **EstimateQ**, we are not actually using the values of $V^*(\cdot)$ but rather values returned by **EstimateV**, which are themselves only estimates. The crucial step in the proof is to show that as h increases, the overall inaccuracy *decreases*.

Let us first define an intermediate random variable that will capture the inaccuracy due to the limited sampling. Define $U^*(s,a)$ as follows:

$$U^*(s,a) = R_{sa} + \gamma \frac{1}{C}\sum_{i=1}^{C} V^*(s_i) \qquad (12)$$

where the s_i are drawn according to $P_{sa}(\cdot)$. Note that $U^*(s,a)$ is averaging values of $V^*(\cdot)$, the unknown value function. Since $U^*(s,a)$ is used only for the proof and not in the algorithm, there is no problem in defining it this way. The next lemma shows that with high probability, the difference between $U^*(s,a)$ and $Q^*(s,a)$ is at most λ.

Lemma 3 *For any state s and action a, with probability at least $1-e^{-\lambda^2 C/V_{\max}{}^2}$ we have*

$$|Q^*(s,a) - U^*(s,a)| \quad = \quad \gamma \left| \mathbf{E}_{s\sim P_{sa}(\cdot)}[V^*(s)] - \frac{1}{C}\sum_i V^*(s_i) \right| \leq \lambda,$$

where the probability is taken over the draw of the s_i from $P_{sa}(\cdot)$.

Proof: Note that $Q^*(s,a) = R_{sa} + \gamma \mathbf{E}_{s\sim P_{sa}(\cdot)}[V^*(s)]$. The proof is immediate from the Chernoff bound. □

Now that we have quantified the error due to finite sampling, we can bound the error from our using values returned by **EstimateV** rather than $V^*(\cdot)$. We bound this error as the difference between $U^*(s,a)$ and **EstimateV**. In order to make our notation simpler, let $V^n(s)$ be the value returned by **EstimateV**(n, C, γ, G, s), and let $Q^n(s,a)$ be the component in the output of **EstimateQ**(n, C, γ, G, s) that corresponds to action a. Using this notation, our algorithm computes

$$Q^n(s,a) = R_{sa} + \gamma \frac{1}{C} \sum_{i=1}^{C} V^{n-1}(s_i) \tag{13}$$

where $V^{n-1}(s) = \max_a\{Q^{n-1}(s,a)\}$, and $Q^0(s,a) = 0$ for every state s and action a.

We now define a parameter α_n that will eventually bound the difference between $Q^*(s,a)$ and $Q^n(s,a)$. We define α_n recursively:

$$\alpha_{n+1} = \gamma(\lambda + \alpha_n) \tag{14}$$

where $\alpha_0 = V_{max}$. Solving for α_H we obtain

$$\alpha_H = \left(\sum_{i=1}^{H} \gamma^i \lambda \right) + \gamma^H V_{max} \leq \frac{\lambda}{1-\gamma} + \gamma^H V_{max}. \tag{15}$$

The next lemma bounds the error in the estimation, at level n, by α_n. Intuitively, the error due to finite sampling contributes λ, while the errors in estimation contribute α_n. The combined error is $\lambda + \alpha_n$, but since we are discounting, the effective error is only $\gamma(\lambda + \alpha_n)$, which by definition is α_{n+1}.

Lemma 4 *With probability at least $1 - (kC)^n e^{-\lambda^2 C/V_{\max}{}^2}$ we have that*

$$|Q^*(s,a) - Q^n(s,a)| \leq \alpha_n. \tag{16}$$

Proof: The proof is by induction on n. It clearly holds for $n = 0$. Now

$$\begin{aligned} |Q^*(s,a) - Q^n(s,a)| &= \gamma \left| \mathbf{E}_{s\sim P_{s,a}(\cdot)}[V^*(s)] - \frac{1}{C}\sum_i V^{n-1}(s_i) \right| \\ &\leq \gamma \left(\left| \mathbf{E}_{s\sim P_{s,a}(\cdot)}[V^*(s)] - \frac{1}{C}\sum_i V^*(s_i) \right| \right. \\ &\qquad \left. + \left| \frac{1}{C}\sum_i V^*(s_i) - \frac{1}{C}\sum_i V^{n-1}(s_i) \right| \right) \\ &\leq \gamma(\lambda + \alpha_n) = \alpha_{n+1} \end{aligned}$$

We require that all of the C child estimates be good, for each of the k actions. This means that the probability of a bad estimate increases by a factor of kC, for each n. By Lemma 3 the probability of a single bad estimate is bounded by $e^{-\lambda^2 C/{V_{max}}^2}$. Therefore the probability of some bad estimate is bounded by $1 - (kC)^n e^{-\lambda^2 C/{V_{max}}^2}$. $\square$

From $\alpha_H \leq \gamma^H V_{max} + \lambda/(1-\gamma)$, we also see that for $H = \log_\gamma(\lambda/V_{max})$, with probability $1 - (kC)^H e^{-\lambda^2 C/{V_{max}}^2}$ all the final estimates $Q^H(s_0, a)$ are within $2\lambda/(1-\gamma)$ from the true Q-values. The next step is to choose C such that $\delta = \lambda/R_{max} \geq (kC)^H e^{-\lambda^2 C/{V_{max}}^2}$ will bound the probability of a bad estimate during the entire computation. Specifically,

$$C = \frac{{V_{max}}^2}{\lambda^2}\left(2H \log \frac{kH{V_{max}}^2}{\lambda^2} + \log\frac{1}{\delta}\right) \tag{17}$$

is sufficient to ensure that with probability $1-\delta$ all the estimates are accurate.

At this point we have shown that with high probability, algorithm $\mathcal{A}$ computes a good estimate of $Q^*(s_0, a)$ for all a, where s_0 is the input state. To complete the proof, we need to relate this to the expected value of a stochastic policy. We give a fairly general result about MDPs, which does not depend on our specific algorithm. (A similar result appears in [SY94].)

Lemma 5 *Assume that π is a stochastic policy, so that $\pi(s)$ is a random variable. If for each state s, the probability that $Q^*(s,\pi^*(s)) - Q^*(s,\pi(s)) < \lambda$ is at least $1-\delta$, then the discounted infinite horizon return of π is at most $(\lambda + 2\delta V_{\max})/(1-\gamma)$ from the optimal return, i.e., for any state s $V^*(s) - V^\pi(s) \leq (\lambda + 2\delta V_{\max})/(1-\gamma)$.*

Proof: Since we assume that the rewards are bounded by R_{max}, it implies that the expected return of π at each state s is at least

$$\mathbf{E}[Q^*(s,\pi(s))] \geq (1-\delta)(Q^*(s,\pi^*(s)) - \lambda) - \delta V_{max} \geq Q^*(s,\pi^*(s)) - \lambda - 2\delta V_{max}. \tag{18}$$

Now we show that if π has the property that at each state s the difference between $\mathbf{E}[Q^*(s,\pi(s))]$ and $Q^*(s,\pi^*(s))$ is at most β, then $V^*(s) - V^\pi(s) \leq \beta/(1-\gamma)$. (A similar result was proved by Singh and Yee [SY94], for the case that *each* action chosen has $Q^*(s,\pi^*(s)) - Q(s,\pi(s)) \leq \beta$. It is easy to extend their proof to handle the case here, and we sketch a proof only for completeness.)

The assumption on the Q^* values immediately implies $|\mathbf{E}[R(s,\pi^*(s))] - \mathbf{E}[R(s,\pi(s))]| \leq \beta$. Consider a policy π_j that executes π for the first $j+1$ steps and then executes π^*. We can show by induction on j that for every state s, $V^*(s) - V^{\pi_j}(s) \leq \sum_{i=0}^{j} \beta\gamma^i$. This implies that $V^*(s) - V^\pi(s) \leq \sum_{i=0}^{\infty} \beta\gamma^i = \beta/(1-\gamma)$.

By setting $\beta = \lambda + 2\delta V_{max}$ the lemma follows. □

Now we can combine all the lemmas to prove our main theorem.

Proof of Theorem 1: As discussed before, the running time is immediate from the algorithm, and the main work is showing that we compute a near-optimal policy. By Lemma 4 we have that the error in the estimation of Q^* is at most α_H, with probability $1 - (kC)^H e^{-\lambda^2 C/{V_{max}}^2}$. Using the values we chose for C and H we have that with probability $1-\delta$ the error is at most $2\lambda/(1-\gamma)$. By Lemma 5 this implies that such a policy π has the property that from every state s,

$$V^*(s) - V^\pi(s) \leq \frac{2\lambda}{(1-\gamma)^2} + \frac{2\delta V_{max}}{1-\gamma}. \tag{19}$$

Substituting back the values of $\delta = \lambda/R_{max}$ and $\lambda = \epsilon(1-\gamma)^2/4$ that we had chosen, it follows that

$$V^*(s) - V^\pi(s) \leq \frac{4\lambda}{(1-\gamma)^2} = \epsilon. \tag{20}$$

□