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

Michael Kearns
AT&T Labs
mkearns@research.att.com

Yishay Mansour
AT&T Labs and Tel-Aviv University
mansour@research.att.com

Andrew Y. Ng
UC Berkeley
ang@research.att.com

Abstract

An issue that is critical 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 even infinite state spaces, traditional planning and reinforcement learning algorithms are often inapplicable, since their running time typically scales linearly with the state space size in the worst case. In this paper we present a new algorithm that, given only a *generative model* (simulator) for an arbitrary MDP, performs near-optimal planning with a running time that has *no dependence* on the number of states. Although 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 results establish for the first time that there are no theoretical barriers to computing near-optimal policies in arbitrarily large, unstructured MDPs.

1 Introduction

In the past decade, Markov decision processes (MDPs) and reinforcement learning have become a standard framework for planning and learning under uncertainty 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 [MHK+98].

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 an MDP becomes very large, the classical planning assumption that the MDP is given *explicitly* by tables of rewards and transition probabilities becomes infeasible. One approach to this 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 rather 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 are a natural way in which a large MDP might be specified, and are more general than most structured representations, in the sense that structured representations usually provide an efficient way of implementing a generative model. Note also that since a generative model provides less information than explicit tables of probabilities, but more information than a single continuous trajectory of experience generated according to some exploration policy, results obtained via a generative model blur 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. From any given state s, the algorithm samples the generative model for many different state-action pairs, and uses these samples to compute a near-optimal action from s. 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 of course be spread over the entire state space. 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 can be shown to be unavoidable without further assumptions).

Note that this learning algorithm is itself simply a (stochastic) 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 complimentary 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.

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*.*
- **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* $R_{\max}$*.*

For simplicity, we shall assume in this paper 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 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* $P_{sa}(\cdot)$*.*

Following standard terminology in reinforcement learning, 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 reinforcement learning [1], 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

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

(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 near-optimal 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 the input state to $\mathcal{A}$ requires only $O(1)$ space, 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). 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).

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

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 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, this can also be written*

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

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

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

  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*.

While a sketch of 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.

For the sake of simplicity, let us consider only the two-action case here, with actions a_1 and a_2. 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, and the basic idea of our algorithm is to find good estimates of $Q^*(s,a)$ for all actions a by looking only within a *small* neighborhood of s. In particular, for our algorithm to run in time that does not depend on $N = |S|$, it is critical that the size of this neighborhood does not depend on N, even though, for example, s may have very diffuse transition probabilities, so that it is possible to reach any other state in S from s.

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

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

where r_i is the reward received on the ith time step upon executing the optimal policy π^* from s. Note the "0-step" expected discounted reward is easy to estimate: Since $V_0^*(s) = 0$, we may simply pick our 0-step estimates to be $\hat{V}_0^*(s) = 0$. Moreover, we see that the $V_h^*(s)$, for $h \geq 1$, are recursively given by

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

where a^* is the action taken by the optimal policy from state s. 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

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

We have described our algorithm "bottom up," but it is also informative to view it "top down." Our algorithm is essentially building a *sparse look-ahead tree*. 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

Function: $\mathbf{EstimateQ}(h, C, \gamma, G, s)$
Input: depth h, width C, discount γ, A 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 $n = 0$, 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
$$\hat{Q}_h^*(s,a) = R(s,a) + \gamma \frac{1}{C} \sum_{s' \in S_a} \mathbf{EstimateV}(h-1, C, \gamma, G, s'). \tag{5}$$
4. Return $(\hat{Q}_h^*(s,a_1), \hat{Q}_h^*(s,a_2), \ldots, \hat{Q}_h^*(s,a_k))$.

Function: $\mathbf{EstimateV}(h, C, \gamma, G, s)$
Input: depth h, width C, discount γ, generative model G, state s.
Output: A number $\hat{V}_h^*(s)$ that is an estimate of $V_h^*(s)$.

1. Let $(\hat{Q}_h^*(s,a_1), \hat{Q}_h^*(s,a_2), \ldots, \hat{Q}_h^*(s,a_k)) := \mathbf{EstimateQ}(h, C, \gamma, G, s)$.
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 ϵ, discount γ, 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 ϵ, γ and R_{max} in Theorem1.
2. Let $(\hat{Q}_H^*(s,a_1), \hat{Q}_H^*(s,a_2), \ldots, \hat{Q}_H^*(s,a_k)) := \mathbf{EstimateQ}(H, C, \gamma, G, s_0)$.
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.

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 $\hat{V}_H^*(s_0)$.

To complete the description of the algorithm, all that remains is to choose the depth H, depth, and 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.

However, such a tree may still be as large as M itself, depending on the choice of C. For instance, if the next-state distribution from s is uniform or nearly uniform over all the states in M, then it would naively seem that, in order to approximate the next-state distributions well, we would need to take at least $C = O(N)$ samples, if only to make sure we see most of possible next-states at least once in our samples.

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 of Theorem 1 are sketched in Appendix A.

The resulting tree thus represents 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 full look-ahead tree. In this sense, our algorithm is clearly related to and inspired by classical look-ahead search techniques [RN95] our main contribution is in showing that in very 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.

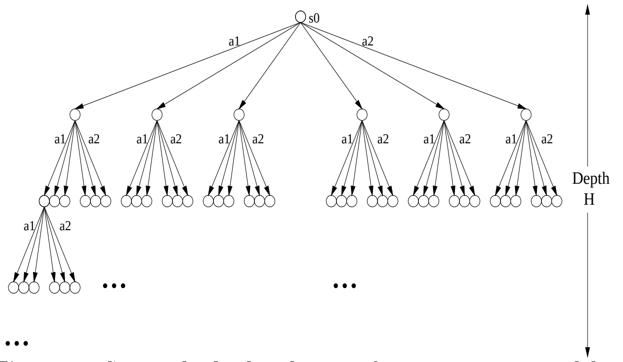

Figure 2: Sparse look-ahead tree of states constructed by the algorithm. (Shown with $C = 3$, actions a_1, a_2.)

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. Nevertheless, there are a couple of simple tricks that may help to reduce the running time in certain cases.

The first idea is simply to use 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 should be careful when addressing randomized procedures; we can show that the properties of the algorithm are maintained under this optimization (details are deferred to the full version of the paper).

In implementing the algorithm, one may also wish not to specify ϵ in advance, but rather just try to do as well as is possible with the computational resources available, in which case an "iterative-deepening" approach may be taken. In our case, this would entail simultaneously increasing C and H by decreasing the target ϵ. Also, as studied in Davies et. al. [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.

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. Two interesting directions for improvement are to allow partially observable MDPs, 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 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.

References

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

[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.

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

[RN95] S. Russell and P. 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 sketch the proof of Theorem 1. 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:

$$U^*(s,a) = R_{sa} + \gamma \frac{1}{C}\sum_{i=1}^{C} V^*(s_i) \tag{9}$$

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 (proof omitted) shows that with high probability, the difference between $U^*(s,a)$ and $Q^*(s,a)$ is at most λ.

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

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

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

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

$$Q^n(s,a) = R_{sa} + \gamma \frac{1}{C}\sum_{i=1}^{C} V^{n-1}(s_i) \tag{10}$$

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:

$$\alpha_{n+1} = \gamma(\lambda + \alpha_n) \tag{11}$$

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

$$\alpha_H = \left(\sum_{i=1}^{H} \gamma^i \lambda\right) + \gamma^H V_{max} \le \frac{\lambda}{1-\gamma} + \gamma^H V_{max}. \tag{12}$$

The next lemma (proof omitted) 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 3 *With probability at least $1 - (kC)^n e^{-\lambda^2 C/V_{\max}{}^2}$ we have that*

$$|Q^*(s,a) - Q^n(s,a)| \le \alpha_n. \tag{13}$$

From $\alpha_H \le \gamma^H V_{max} + \lambda/(1-\gamma)$, 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 $\delta = \lambda/R_{max} \ge (kC)^H e^{-\lambda^2 C/V_{max}{}^2}$ will bound the probability of a bad estimate during the entire computation. Specifically,

$$C = \frac{V_{max}{}^2}{\lambda^2}\left(2H \log \frac{kHV_{max}{}^2}{\lambda^2} + \log\frac{1}{\delta}\right) \tag{14}$$

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 4 *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, that is, for any state s $V^*(s) - V^\pi(s) \le (\lambda + 2\delta V_{\max})/(1-\gamma)$.*

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 3 we have that the error in the estimation of Q^* is at most α_H, with probability $1-(kC)^H$. 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 4 this implies that such a policy π has the property that from every state s,

$$V^*(s) - V^\pi(s) \le \frac{2\lambda}{(1-\gamma)^2} + \frac{2\delta V_{max}}{1-\gamma}. \tag{15}$$

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

$$V^*(s) - V^\pi(s) \le \frac{4\lambda}{(1-\gamma)^2} = \epsilon. \tag{16}$$

□