Theoretical Analysis of A Sparse Sampling Algorithm for Near-Optimal Planning in Large MDPs

Jiaming Shen
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
js2@illinois.edu

Abstract

In this project report, we present detailed analyses of the sparse sampling algorithm in [3] for near-optimal planning in large Markov Decision Processes (MDPs). For arbitrary MDP, this algorithm takes its simulator as input and performs online, near-optimal planning with a *per-state* running time that has *no dependence* on the number of states. As a result, this algorithm scales to large MDPs with possibly infinite state spaces. We will focus on discussing the key ideas of this algorithm and prove it indeed achieves both efficiency and near-optimality, simultaneously. Furthermore, we show this algorithm is exponential in the horizon time, and such exponential dependency is inevitable for under the setting of paper [3]. Finally, we present some discussions on the implications of this algorithm and compare/relate it to the (villain) Monto Carlo Tree Search algorithm.

1 Introduction

Reinforcement learning with Markov Decision Process (MDP) is a powerful framework for long-term planning and learning under certainty. Traditionally, people consider *planning* in MDP as a task that takes the MDP model as input and returns a good (ideally optimal) policy. When the number of states N is very large and no further assumption on the MDP structure, planning under the above viewpoint is clearly infeasible, as even reading the model input requires N^2 time. As a result, planning in large MDP with a possibly infinite number of states requires rethinking what *planning* means.

The authors of paper [3] take an alternative view of planning and treat it as a task that takes a single state as input and outputs a single action to take from that state. In this online view, a planning algorithm itself serves as a (stochastic) policy. Furthermore, they describe a sparse sampling algorithm and prove it is near-optimal and has no dependency on the state space size. This algorithm is exponential in the horizon time, and they prove such exponential dependency is inevitable under their setting. The descriptions and proofs of this algorithm are the primary focuses of this report.

The remaining of this report is organized as follows. First, we formalize the problem setting and describe all useful notations in Sect. (2). Then, we present the algorithm and discuss its key ideas and intuitions in Sect. (3). After that, we provide all detailed proofs in Sect. (4). Finally, in Sect. (5), we discuss the implications of this algorithm and conclude this report in Sect. (6).

2 Problem Setting

We begin with the formal definition of a MDP M and then define its corresponding *generative model*.

Definition 1. *A Markov Decision Process M on a set of states S and with a set of actions $A = \{a_1, \ldots, a_k\}$ consists of two components: (1) Transition probabilities P where $P_{sa}(s')$ specifies the*

probability of transition to each state s' upon execution of action a from state s, and (2) Reward distributions R where R_{sa} indicates the reward distribution for executing action a from state s.

In this paper, for simplicity, we assume all rewards are deterministic, that is, the reward distributions have zero variance. Therefore, with a slight abuse of notation, we view R_{sa} as the mean of reward distribution and executing a from s is always exactly R_{sa}. Finally, we assume rewards are bounded in **absolute** value by R_{max}.

We define a stochastic *policy* to be a mapping $\pi : S \mapsto A$. This paper is primarily concerned with **discounted** MDPs and thus we assume a discount factor $0 \leq \gamma < 1$ is given. We then define the *value function* V^π for any policy π:

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

where r_i is the rewards on the ith step of executing the policy π from starting state s. Because $|R_{sa}| \leq R_{max}$, we have $|V^\pi(s)| \leq V_{max}$ where $V_{max} = \frac{R_{max}}{1-\gamma}$.

Following standard terminology, we also define the Q-function for a given policy π as:

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

When the policy is implemented by a stochastic algorithm $\mathcal{A}$, we use $V^{\mathcal{A}}$ and $Q^{\mathcal{A}}$ to denote the value function and Q-function of the policy implemented by $\mathcal{A}$. We define the optimal value function $V^*(s) = \sup_\pi V^\pi(s)$ and optimal Q-function $Q^*(s,a) = \sup_\pi Q^\pi(s,a)$. Finally, we define the optimal policy π^* where $\pi^*(s) = \text{argmax}_a Q^*(s,a)$ for all $s \in S$.

When the state space size $|S|$ of a MDP M is very large or even infinite, we can no longer specify its transition probability or any policy π for this MDP explicitly. Instead, we give a generative model G which possesses the ability to sample the behavior of M. As a result, the given MDP is simulative rather than explicit.

Definition 2. *A generative model G for a Markov Decision Process M is a randomized algorithm that takes a state-action pair (s,a) as input, and returns R_{sa} and a state s', where s' is drawn following the transition probabilities $P_{sa}(\cdot)$.*

This generative model is a natural formalization of a "simulator" for an arbitrary MDP. Although generative models provide less information than explicit tables of probabilities, they are often available when explicit tables are not. Furthermore, generative models contain more information than a single trajectory sampled from an exploration policy, and to some extent, enables explorations. Essentially, generative models are "compact representations" of MDPs and we will show in the next section, with the knowledge of a generative model, we can design a planning algorithm that is near optimal.

3 A Sparse Sampling Algorithm for Planning in Large MDP

Recall that in paper [3], we take an online view of planning in which a planning algorithm itself is a simply a stochastic policy mapping a state s to an action a. In Fig. (1), we describe a planning algorithm $\mathcal{A}$. One of main contributions of paper [3] is the following theorem.

Theorem 1. *Given a generative model for any k-action MDP M, the planning algorithm $\mathcal{A}$ takes as input any state $s \in S$ and any value $\epsilon > 0$, outputs an action, and satisfies the following two conditions: (1)* ***Near-optimality:*** *The value function of the policy implemented by $\mathcal{A}$ satisfies $|V^{\mathcal{A}}(s) - V^*(s)| \leq \epsilon$ for all $s \in S$, and (2)* ***Efficiency:*** *The running time of $\mathcal{A}$ is $O((kC)^H)$, where $H = \lceil log_\gamma(\lambda/V_{max})\rceil$, $C = \frac{V_{max}^2}{\lambda^2}\left(2Hlog\frac{kHV_{max}^2}{\lambda^2} + log\frac{R_{max}}{\lambda}\right)$, $\lambda = (\epsilon(1-\gamma)^2)/4$, and $V_{max} = R_{max}/(1-\gamma)$.*

We define the value H to be ϵ-horizon time. First, we notice that although this algorithm $\mathcal{A}$ has exponential dependency on H, it has no dependency on the state space size $|S|$ because both C and H are independent of $|S|$. Second, although the algorithm $\mathcal{A}$ itself implements a stochastic policy, the near-optimality condition is not defined probabilistically. The detailed proofs of Theorem 1 is given in the next section. We first present some high-level intuitions for the algorithm and its analysis.

Given an input state s_0, the planning algorithm $\mathcal{A}$ aims to return a near-optimal action a. The key idea is to sample the generative model from states in the "neighborhood" of s_0. These samples form a

Function: $\mathbf{EstimateQ}(h, C, \gamma, G, s)$
Input: depth h, width C, discount γ, 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

$$\hat{Q}_h^*(s,a) = R(s,a) + \gamma \frac{1}{C} \sum_{s' \in S_a} \mathbf{EstimateV}(h-1, C, \gamma, G, s'). \tag{5}$$

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: $\mathbf{Algorithm}\ \mathcal{A}(\epsilon, \gamma, R_{max}, G, s_0)$
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: A sample sampling algorithm for planning in large MDP. *EstimateQ* finds $\hat{Q}_h^*$. *EstimateV* finds $\hat{V}_h^*$, and Algorithm $\mathcal{A}$ implements the planning algorithm. Credits to [3].

small "sub-MDP" M' of M such that the optimal action in M' from s_0 is a near-optimal action in M. The algorithm running time is essentially determined by the number of samples from the generative model, and thus if we sample the generative model carefully (*i.e.*, without causing dependency of state space size), the final running time will also be independent of state space size.

The algorithm $\mathcal{A}$ starts from the input state s_0 and for each action $a \in A$, it calls the generative model C times to generate a set of C next states denoted as S_a. Totally kC samples are drawn here, where $k = |A|$. Then, for each state $s' \in \cup_{a \in A} S_a$, the algorithm $\mathcal{A}$ repeats the above process and generates another kC samples. So till now totally $O((kC)^2)$ samples are drawn. By recursively conducting this process H times, we will generate $O((kC)^H)$ samples which defines the "sub-MDP" M'. The graphical structure of M' is a directed tree in which each node is labeled by a state and each edge is labeled with an action, as shown in Fig. (2). We consider M' as an MDP in which s_0 is the start state and taking an action from a node in the tree causes a transition to random child of that node with the corresponding action label. Finally, we treat the leaf nodes as absorbing states. Under this view, we can compute an optimal action for root s_0 in M'. Specifically, we assign zero values to leaf nodes as our estimates of $\hat{V}_0^*$, which are backed-up to compute estimates of $\hat{V}_1^*$ for their parents, which are in turn further backed-up to the root node to find an estimate of $\hat{V}_H^*(s_0)$. The core of our later proofs is to show this estimate $\hat{V}_H^*(s_0)$ (in sub-MDP M') is close to $V^*(s_0)$ (in original full MDP M) for any $s_0 \in S$, if the value of H is specified properly.

Finally, we show a lower bound for the running time of any ϵ-optimal algorithm with access only to a generative model in the following theorem.

Theorem 2. *For any algorithm $\mathcal{A}$ that has access only to a generative model for a MDP M and implements an ϵ-optimal policy π satisfying $|V^{\mathcal{A}}(s) - V^*(s)| \leq \epsilon$ for all states $s \in S$, there exist a MDP M on which $\mathcal{A}$ makes at least $\Omega(2^H) = \Omega((1/\epsilon)^{1/log(1/\gamma)})$ calls to the generative model.*

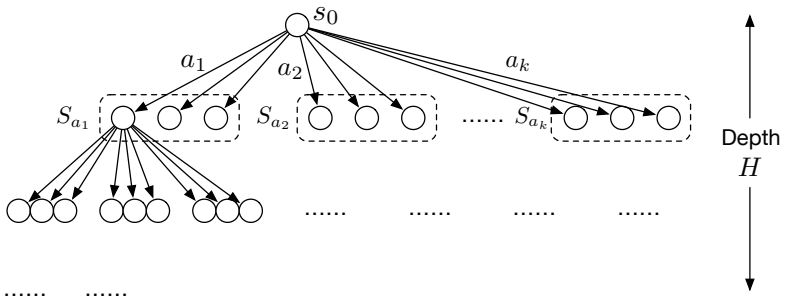

Figure 2: A tree-structured MDP M' conducted by the algorithm $\mathcal{A}$ with C=3.

Theorem 2 shows the exponential dependency on ϵ-horizon time H is inevitable in this paper's setting. We prove this theorem in the next section.

4 Theorem Proofs

4.1 Proof of Theorem 1

Theorem 1 states that the Algorithm $\mathcal{A}$ achieves both near-optimality and efficiency. The claim on **efficiency** follows immediately from the algorithm definition. Each call to $EstimateQ$ generates kC calls to $EstimateV$ and generative model G, and reduces the depth parameter h by 1. As the depth of recursion is at most H, the running time is $O((kC)^H)$.

The claim on **near-optimality** states the return values of $EstimateQ$ are indeed good estimates of Q^*. By comparing $\hat{Q}_h^*(s,a)$ (*i.e.*, equation 5 in Fig. 1) and $Q^*(s,a)$, we can find that there are two sources of inaccuracy in these estimates. First, we use only a finite set of samples to approximate the expectation $\mathbf{E}_{s'\sim P_{sa}(\cdot)}$. Second, we are not using V^* but rather values returned by $EstimateV$, which are themselves only estimates. We need to prove that when h increases, the overall inaccuracy decreases. In the following, we use Lemma 1 to bound the inaccuracy from limited sampling, and use Lemmas 2 and 3 to bound the inaccuracy from discrepancy of V^* and $EstimateV$.

Lemma 1. *Let $U^*(s,a) := R_{sa} + \gamma \frac{1}{C}\sum_{i=1}^{C} V^*(s_i)$, for any state s and action a, with probability at least $1 - 2e^{-\lambda^2 C/V_{max}^2}$, we have $|Q^*(s,a) - U^*(s,a)| \leq \lambda$, where s_i is drawn from $P_{sa}(\cdot)$*[1].

Proof: We notice that $\mathbf{E}_{s'\sim P_{sa}(\cdot)}[V^*(s')]$ is the true mean of the random variable $V^*(s_i)$ when s_i is drawn from $P_{sa}(\cdot)$. Therefore, from Hoeffding's Inequality, we have:

$$\Pr\left[|Q^*(s,a) - U^*(s,a)| \leq \lambda\right] = \Pr\left[\gamma\left|\mathbf{E}_{s'\sim P_{sa}(\cdot)}[V^*(s')] - \frac{1}{C}\sum_i V^*(s_i)\right| \leq \lambda\right] \tag{3}$$

$$\geq 1 - 2e^{-\lambda^2 C/\gamma^2 V_{max}^2} \tag{4}$$

$$\geq 1 - 2e^{-\lambda^2 C/V_{max}^2} \tag{5}$$

The last inequality holds because $\gamma < 1$. This lemma quantifies the error due to finite sampling.

Lemma 2. *We first denote the output of $EstimateV(n, C, \gamma, G, s)$ and $EstimateQ(n, C, \gamma, G, s)$ to be $V^n(s)$ and $Q^n(s,a)$, respectively. Then, we define α_n recursively from $\alpha_0 = V_{max}$, $\alpha_{n+1} = \gamma(\lambda + \alpha_n)$. With probability at least $1 - (kC)^n e^{-\lambda^2 C/V_{max}^2}$, we have $|Q^*(s,a) - Q^n(s,a)| \leq \alpha_n$.*

Proof: Based on the above definitions of $V^n(s)$ and $Q^n(s,a)$, we have:

$$Q^n(s,a) = R_{sa} + \gamma \frac{1}{C}\sum_{i=1}^{C} V^{n-1}(s_i), \tag{6}$$

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

[1]This result is slightly different from the original lemma 3 in [3] where the error probability is $e^{-\lambda^2 C/V_{max}^2}$ instead of twice of it. Since we need to bound the absolute error from two sides, we believe missing this factor 2 is a typo in the original paper.

We first prove the following result:

$$\Pr[|Q^*(s,a) - Q^n(s,a)| \leq \epsilon] \geq 1-\delta \implies \Pr[|V^*(s) - V^n(s)| \leq \epsilon] \geq 1-\delta. \tag{7}$$

This is because $|V^*(s) - V^n(s)| = |max_a Q^*(s,a) - max_a Q^n(s,a)|$. If $max_a Q^*(s,a) > max_a Q^n(s,a)$, we let $a^* = \text{argmax}_a Q^n(s,a)$, then we have:

$$|V^*(s) - V^n(s)| = Q^*(s,a^*) - Q^n(s,a^*) + Q^n(s,a^*) - Q^n(s,a) \leq \epsilon + 0 = \epsilon \tag{8}$$

Similarly, if $max_a Q^*(s,a) < max_a Q^n(s,a)$, we let $a^* = \text{argmax}_a Q^*(s,a)$, and then we have:

$$|V^*(s) - V^n(s)| = Q^n(s,a^*) - Q^*(s,a^*) + Q^*(s,a^*) - Q^*(s,a) \leq \epsilon + 0 = \epsilon \tag{9}$$

We then prove this above lemma by induction on n. When $n = 0$, we have:

$$\Pr[|Q^*(s,a) - Q^0(s,a)| \leq \alpha_0] = \Pr[|Q^*(s,a) - 0| \leq V_{max}] = 1 \geq 1 - e^{-\lambda^2 C/V_{max}^2}. \tag{10}$$

Now, suppose the lemma holds for $n = h-1$, that is $\Pr\left[|Q^*(s,a) - Q^{h-1}(s,a)| > \alpha_{h-1}\right] < (kC)^{h-1} e^{-\lambda^2 C/V_{max}^2}$, then for $n = h$ we have the following:

$$|Q^*(s,a) - Q^h(s,a)| = \gamma \left| \mathbf{E}_{s' \sim P_{sa}(\cdot)}[V^*(s')] - \frac{1}{C}\sum_{i=1}^{C} V^*(s_i) + \frac{1}{C}\sum_{i=1}^{C} V^*(s_i) - \frac{1}{C}\sum_{i=1}^{C} V^{h-1}(s_i) \right| \tag{11}$$

$$\leq \gamma \left(\left| \mathbf{E}_{s' \sim P_{sa}(\cdot)}[V^*(s')] - \frac{1}{C}\sum_{i=1}^{C} V^*(s_i) \right| + \left| \frac{1}{C}\sum_{i=1}^{C} V^*(s_i) - \frac{1}{C}\sum_{i=1}^{C} V^{h-1}(s_i) \right| \right) \tag{12}$$

Therefore, we have:

$$\Pr\left[|Q^*(s,a) - Q^h(s,a)| > \alpha_h\right] = \Pr\left[|Q^*(s,a) - Q^h(s,a)| > \gamma(\lambda + \alpha_{h-1})\right] \tag{13}$$

$$\leq \Pr\left[\gamma\left(\left|\mathbf{E}_{s' \sim P_{sa}(\cdot)}[V^*(s')] - \frac{1}{C}\sum_{i=1}^{C} V^*(s_i)\right|\right) > \gamma\lambda\right] \tag{14}$$

$$+ \Pr\left[\gamma\left(\left|\frac{1}{C}\sum_{i=1}^{C} V^*(s_i) - \frac{1}{C}\sum_{i=1}^{C} V^{h-1}(s_i)\right|\right) > \gamma\alpha_{h-1}\right] \tag{15}$$

$$\leq 2e^{-\lambda^2 C/V_{max}^2} + C \cdot (kC)^{h-1} e^{-\lambda^2 C/V_{max}^2} \tag{16}$$

$$\leq (kC)^h e^{-\lambda^2 C/V_{max}^2} \tag{17}$$

The first inequality comes from the following two facts: (1) If $0 < a \leq b, 0 < c$, then $\Pr[a > c] \leq \Pr[b > c]$ and (2) $\Pr[a + b > c_1 + c_2] < \Pr[a > c_1 \cup b > c_2] < \Pr[a > c_1] + \Pr[b > c_2]$. The second inequality comes from lemma 1 and our assumption that lemma 2 holds for $n = h-1$. The last inequality holds as we can simply assume $2 < C^h(k^h - k^{h-1})$.

Colloary 1. *For $\delta = \frac{\lambda}{R_{max}}$, $H = log_\gamma(\frac{\lambda}{V_{max}})$, $C = \frac{V_{max}^2}{\lambda^2}(2Hlog\frac{kHV_{max}^2}{\lambda^2} + log\frac{1}{\delta})$, we have* $\Pr[|Q^*(s,a) - Q^H(s,a)| \leq \frac{2\lambda}{1-\gamma}] \geq 1-\delta$.

From the recursive definition of α, we have:

$$\alpha_H = \left(\sum_{i=1}^{H} \gamma^i \lambda\right) + \gamma^H V_{max} \leq \frac{\lambda}{1-\gamma} + \gamma^H V_{max} \leq \frac{2\lambda}{1-\gamma}. \tag{18}$$

Then we verify that for the given C and H, we have $(kC)^H e^{-\lambda^2 C/V_{max}^2} \leq \delta$. Specifically, we have:

$$(kC)^H e^{-\lambda^2 C/V_{max}^2} \leq \delta \impliedby Hlog(kC) - \left(2Hlog\frac{kHV_{max}^2}{\lambda^2} + log\frac{1}{\delta}\right) \leq log\delta \tag{19}$$

$$\impliedby log(kC) \leq 2log(\frac{kHV_{max}^2}{\lambda^2}) \impliedby C \leq \frac{kH^2V_{max}^4}{\lambda^4} \tag{20}$$

$$\impliedby 2Hlog\frac{kHV_{max}^2}{\lambda^2} + log\frac{1}{\delta} \leq \frac{kH^2V_{max}^2}{\lambda^2} \tag{21}$$

$$\impliedby 2Hlog\frac{kHV_{max}^2}{\lambda^2} + log\frac{kHV_{max}^2}{\lambda^2} \leq \frac{kH^2V_{max}^2}{\lambda^2} \tag{22}$$

$$\impliedby 3log\frac{kHV_{max}^2}{\lambda^2} \leq \frac{kHV_{max}^2}{\lambda^2} \tag{23}$$

Line 19 is obtained by replacing C in $e^{-\lambda^2 C/V_{max}^2}$. Line 21 is obtained by further replacing C in the left hand side of line 20. Line 22 holds because $\frac{1}{\delta} = \frac{V_{max}(1-\gamma)}{\lambda} < \frac{kHV_{max}^2}{\lambda^2}$. You can further verify this inequality by noticing $\gamma < 1$, $k > 1$, $H > 1$, and $\gamma^H = \frac{\lambda}{V_{max}}$. Finally, we can see that if λ is small enough, line 23 holds because $\frac{kHV_{max}^2}{\lambda^2} > 3$. Since λ is the error measure, it should be small, and thus we will have $(kC)^H e^{-\lambda^2 C/V_{max}^2} \leq \delta$.[2]

The lemma shows algorithm $\mathcal{A}$ computes good estimates of $Q^*(s_0, a)$ for all actions a. Following we relate this result to the expected value of a stochastic policy implemented by algorithm $\mathcal{A}$.

Lemma 3. *Given a stochastic policy π, if for each state s, with probability at least $1-\delta$, we have $Q^*(s, \pi^*(s)) - Q(s, \pi(s)) < \lambda$, then for any state s we have $V^*(s) - V^\pi(s) \leq (\lambda + 2\delta V_{max})/(1-\gamma)$.*

As π is a stochastic policy, $\pi(s)$ is a random variable. Furthermore, we have assumed the rewards are bounded by R_{max} and thus $|Q^*(s,a)| \leq V_{max}$ for any (s,a) pair. We have:

$$\begin{aligned}
\mathbf{E}[Q^\pi(s, \pi(s))] &\geq (1-\delta)(Q^*(s, \pi^*(s)) - \lambda) - \delta V_{max} && (24)\\
&= Q^*(s, \pi^*(s)) - \lambda - \delta Q^*(s, \pi^*(s)) + \delta\lambda - \delta V_{max} && (25)\\
&\geq Q^*(s, \pi^*(s)) - \lambda - 2\delta V_{max} && (26)
\end{aligned}$$

We then prove the lemma following the procedures in [4]. Specifically, we define the loss:

$$L_{V^\pi}(s) = V^*(s) - V^\pi(s) = Q^*(s, \pi^*(s)) - \mathbf{E}[Q^\pi(s, \pi(s))] \quad (27)$$

Suppose state z achieves maximum loss. That is $L_{V^\pi}(z) \geq L_{V^\pi}(s)$ for any $s \in S$. Let $a = \pi^*(z)$ and $b \sim \pi(z)$. From inequality 26 we have

$$R(z,a) + \gamma \sum_y P_{za}(y) V^*(y) - \lambda - 2\delta V_{max} \leq \mathbf{E}_{b\sim\pi(z)}[R(z,b) + \gamma \sum_y P_{zb}(y) V^\pi(y)] \quad (28)$$

Because $V^*(y) \geq V^\pi(y)$ for any state y, we can replace $V^\pi(y)$ in the right hand size of above inequality and get the following result:

$$R(z,a) - \mathbf{E}_{b\sim\pi(z)}[R(z,b)] \leq \lambda + 2\delta V_{max} + \gamma \mathbf{E}_{b\sim\pi(z)}[\sum_y P_{zb}(y) V^*(y)] - \gamma \sum_y P_{za}(y) V^*(y). \quad (29)$$

Furthermore, we have:

$$\begin{aligned}
L_{V^\pi}(z) = V^*(z) - V^\pi(z) &= Q^*(z,a) - \mathbf{E}_{b\sim\pi(z)}[Q^\pi(z,b)] && (30)\\
&= \big(R(z,a) - \mathbf{E}_{b\sim\pi(z)}[R(z,b)]\big) + \left(\gamma \sum_y P_{za}(y) V^*(y) - \gamma \mathbf{E}_{b\sim\pi(z)}[\sum_y P_{zb}(y) V^\pi(y)]\right) && (31)\\
&\leq \lambda + 2\delta V_{max} + \gamma \mathbf{E}_{b\sim\pi(z)}[\sum_y P_{zb}(y) V^*(y)] - \gamma \sum_y P_{za}(y) V^*(y) && (32)\\
&\quad + \gamma \sum_y P_{za}(y) V^*(y) - \gamma \mathbf{E}_{b\sim\pi(z)}[\sum_y P_{zb}(y) V^\pi(y)] && (33)\\
&= \lambda + 2\delta V_{max} + \gamma \mathbf{E}_{b\sim\pi(z)}\left[\sum_y P_{zb}(y) V^*(y) - \sum_y P_{zb}(y) V^\pi(y)\right] && (34)\\
&= \lambda + 2\delta V_{max} + \gamma \mathbf{E}_{b\sim\pi(z)}\left[\sum_y P_{zb}(y)(V^*(y) - V^\pi(y))\right] && (35)\\
&= \lambda + 2\delta V_{max} + \gamma L_{V^\pi}(y) && (36)\\
&\leq \lambda + 2\delta V_{max} + \gamma L_{V^\pi}(z) && (37)
\end{aligned}$$

inequality 32 comes from inequality 38. inequality 37 holds from the definition of state z that it achieves the maximum loss. Finally, from the definition L_{V^π} and inequality 37 we can get:

$$V^*(s) - V^\pi(s) \leq \frac{\lambda + 2\delta V_{max}}{1-\gamma}. \quad (38)$$

[2] There is no detailed proof of this inequality in the original paper. I suspect the authors know some more advanced inequalities that will make this proof "obvious".

Finally, we combine the above three lemmas and show the near-optimality of Algorithm $\mathcal{A}$. From Colloary 1, we know that with probability $1-\delta$, the estimation error of Q^* is at most $\frac{2\lambda}{1-\gamma}$. By Lemma 3, this implies that such a policy π has the following property that from every state s,

$$V^*(s) - V^\pi(s) \leq \frac{2\lambda}{(1-\gamma)^2} + \frac{2\delta V_{max}}{1-\gamma}. \tag{39}$$

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

$$V^*(s) - V^\pi(s) \leq \frac{4\lambda}{(1-\gamma)^2} = \epsilon. \tag{40}$$

4.2 Proof of Theorem 2

Theorem 2 states that any planning algorithm with access only to a generative model, and which implements a ϵ-optimal policy in a general MDP, must have running time at least exponential in the ϵ-horizon time H. We prove this by constructing a very special MDP that requires the planning algorithm to call the generative model $\Omega(2^H)$ times.

Specifically, we design a MDP M using a binary tree T of depth H. Each node in T represents a state in M. The actions of M are $\{0, 1\}$. When we are in state s and perform an action b, we move to state s_b deterministically, where s_b is the b-th child of s in T. If s is a leaf node in T, we move to an absorbing state. We randomly choose a leaf node v in the tree and let $R(v, b) = R_{max}$ for any $b \in \{0, 1\}$ and $R(v', b) = 0$ for any state $v' \neq v, b \in \{0, 1\}$. Given the state s_0 corresponding to the root node of T, any algorithm $\mathcal{A}$ must perform at least $\Omega(2^H)$ calls to the generative model to find that leaf node v for computing a near optimal policy.

5 Discussion

5.1 Relation to Tabular Certainty-Equivalence

In the note [2] Section 2.3, we show an analysis of certainty-equivalence algorithm that has no explicit dependency on the state space size $|S|$. However, the certainty-equivalence algorithm is a model-based RL algorithm and requires n samples per (s, a) pair, which causes implicit dependency on $|S|$. In comparison, the sparse sampling algorithm presented in this report is a value-based method and focuses on returning the action for one single state s_0. Therefore, it leverages the generative model to obtain samples only of the state-action pairs that matter for the calculation of $Q^*(s_0, \cdot)$. Since the number of state-action pairs that contributes to $Q^*(s_0, \cdot)$ is limited and independent of $|S|$, the final algorithm has no dependency of $|S|$.

5.2 Relation to Monto Carlo Tree Search

Monte Carlo Tree Search (MCTS) was introduced in [1] as a building block for the Go playing engine CrazyStone. Later, researchers and engineers in Google DeepMind developed variants of MCTS algorithm for building AlphaGo/Zero that outperforms professional human Go players. Here, we compare the sparse sampling algorithm with the vanilla MCTS algorithm in [1].

The ultimate goal of MCTS is to output the most promising move/action given a game state, which is exactly the same as the purpose of the sparse sampling algorithm in this report. Furthermore, MCTS will also represent a game using a tree structure where each node is a game state and each edge indicates a possible move/action. There are two main differences between MCTS and the sparse sampling algorithm. First, the sparse sampling algorithm constructs a sparse look-ahead tree (c.f. Fig. 2) in a breadth-first fashion. For each non-leaf node, we will expand it into exactly kC children nodes and the maximum expansion depth is a fixed number H. In comparison, MCTS (dynamically) constructs the game tree in a depth-first fashion. It starts from a node that is not fully expanded and follows a rollout policy to expand it until a terminal node (where the final reward can be calculated) is reached. Second, the sparse sampling algorithm can obtain rewards for every state-action pair from the generative model. On the other hand, MCTS (in its vanilla form) works in a sparse reward setting where the reward is obtained only when the game ends. Such final reward is then backpropagated to all intermediate states. Finally, despite those dissimilarities, these two algorithms still share the same philosophy – using sampling to remove dependency on large state space size.

5.3 Further Improvements

There are several improvements that can help to reduce the running time in practice. First, we can do different amounts of sampling at each level of the tree. The intuition is that the further away we are from the root, the less influence our estimates will have on the Q values at the root due to the discounting. Second, we can change the value estimates in leaf nodes from zero (in current algorithm) to the values returned by an approximate value function at those states. Finally, we can use memorization in the calculation of $EstimateV$ and cache estimated $\hat{V}^h(s)$ to save the running time.

6 Conclusion

In this report, we present a randomized algorithm $\mathcal{A}$ that achieves online planning in large MDPs. We show detailed proofs of two main theorems, some of which are not presented in the original paper. Finally, we discuss the implications of this algorithm and compare it with several related methods.

References

[1] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, 2006.

[2] N. Jiang. Notes on tabular methods, `http://nanjiang.cs.illinois.edu/files/cs598/note3.pdf`, 2019.

[3] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49:193–208, 1999.

[4] S. P. Singh and R. C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.