Variance Reduced Value Iteration and Faster Algorithms for Solving Markov Decision Processes

Aaron Sidford
Stanford University
sidford@stanford.edu

Mengdi Wang
Princeton University
mengdiw@princeton.edu

Xian Wu
Stanford University
xwu20@stanford.edu

Yinyu Ye
Stanford University
yyye@stanford.edu

Abstract

In this paper we provide faster algorithms for approximately solving discounted Markov Decision Processes in multiple parameter regimes. Given a discounted Markov Decision Process (DMDP) with $|S|$ states, $|A|$ actions, discount factor $\gamma \in (0, 1)$, and rewards in the range $[-M, M]$, we show how to compute an ϵ-optimal policy, with probability $1 - \delta$ in time[1]

$$\tilde{O}\left(\left(|S|^2|A| + \frac{|S||A|}{(1-\gamma)^3}\right) \log\left(\frac{M}{\epsilon}\right) \log\left(\frac{1}{\delta}\right)\right) .$$

This contribution reflects the first nearly linear time, nearly linearly convergent algorithm for solving DMDPs for intermediate values of γ.

We also show how to obtain improved sublinear time algorithms provided we can sample from the transition function in $O(1)$ time. Under this assumption we provide an algorithm which computes an ϵ-optimal policy with probability $1 - \delta$ in time

$$\tilde{O}\left(\frac{|S||A|M^2}{(1-\gamma)^4\epsilon^2} \log\left(\frac{1}{\delta}\right)\right) .$$

Lastly, we extend both these algorithms to solve finite horizon MDPs. Our algorithms improve upon the previous best for approximately computing optimal policies for fixed-horizon MDPs in multiple parameter regimes.

Interestingly, we obtain our results by a careful modification of approximate value iteration. We show how to combine classic approximate value iteration analysis with new techniques in variance reduction. Our fastest algorithms leverage further insights to ensure that our algorithms make monotonic progress towards the optimal value. This paper is one of few instances in using sampling to obtain a linearly convergent linear programming algorithm and we hope that the analysis may be useful more broadly.

---

[1] We use $\tilde{O}$ to hide polylogarithmic factors in the input parameters, i.e. $\tilde{O}(f(x)) = O(f(x) \cdot \log(f(x))^{O(1)})$.

1   Introduction

Markov decision processes (MDPs) are a popular mathematical framework used to encapsulate sequential decision-making problems under uncertainty. MDPs are widely used to formulate problems that require planning to maximize rewards aggregated over a long time horizon. Rooted from classical physics, MDPs are the discrete analog of variational problems and continuous-state stochastic optimal control problems and are a fundamental computational model used in the study of control, dynamical systems, and artificial intelligence, with applications in many industries, including health care, finance and engineering. Moreover, recently MDPs have become increasingly important in reinforcement learning, a rapidly developing area of artificial intelligence that studies how an agent interacting with a poorly understood environment can learn over time to make optimal decisions.

   Although MDPs have been extensively studied across multiple disciplines since the 1950s, the development of efficient algorithms for large-scale MDPs remains challenging. More precisely, the best known algorithms in many parameter regimes scale super-linearly with the size of the input which can be prohibitively difficult, especially in large-scale applications. In this work, we provide the first nearly linear convergent, nearly linear time algorithms that can solve MDPs with high precision even when the discount factor depends polynomially (with a small exponent) on the number of states and actions. We also give sublinear algorithms with the fastest runtime dependencies on the parameters of the discounted MDP – the state and action space, and the discount factor. Our algorithms combine the classic value iteration algorithm with novel sampling and variance reduction techniques and our results provide new complexity benchmarks for solving discounted infinite-horizon MDPs.

   In this work, we focus on the discounted infinite-horizon Markov decision problem (DMDP). DMDPs are described by the tuple (S, A, P, r, γ), where S is the finite state space, A is the finite action space, and $\gamma \in (0, 1)$ is the discount factor. P is the collection of state-action-state transition probabilities, with each $p_a(i, j)$ specifying the probability of going to state j from state i when taking action a, r is the collection of rewards at different state-action pairs, i.e., we collect $r_a(i)$ if we are currently in state i and take action a.

   In a Markov Decision Process, at each time step t, a controller takes an available action $a \in A$ from their current state i and reaches the next state j with probability $p_a(i, j)$. For each action a taken, the decision maker earns an immediate reward, $r_a(i)$. A vector $\pi \in A^S$ that tells the actor which action to take from any state is called a stationary policy.

   The main goal in solving a DMDP is to find a stationary, deterministic policy π^* that maximizes the expected discounted cumulative reward. A stationary policy specifies actions to follow at each state, irregardless of time. A deterministic policy gives a single fixed prescribed action for each state. For all results and analyses regarding DMDPs in the rest of this paper, all our policies will be stationary and deterministic. The goal of maximizing the expected discounted cumulative reward can be formulated as

$$\max_{\pi \in A^S} v_\pi(i) := \mathbb{E}_\pi \left[\sum_{t=1}^\infty \gamma^t r_{a_t}(i_t) \mid i_0 = i \right],$$

where $\{i_0, a_0, i_1, a_1, \ldots, i_t, a_t, \ldots\}$ are state-action transitions generated by the MDP under the fixed policy π, i.e. $a_t = \pi_{i_t}$, and the expectation $\mathbb{E}_\pi[\cdot]$ is over the set of (i_t, a_t) trajectories.

   We refer to v_π as the expected value vector of policy π, which describes the expected discounted cumulative rewards corresponding to all possible initial states. We refer to v^* value vector associated with π^*, the policy that maximizes expected discounted cumulative reward for each initial state. We are interested in finding an ϵ-optimal policy π, under which the expected cumulative reward is ϵ-close to the maximal expected cumulative reward regardless of the initial state. More precisely, we say π is ϵ-optimal if $\|v^* - v_\pi\|_\infty \leq \epsilon$.

2

1.1 Our Results

In this paper we provide several randomized algorithms for approximately solving DMDPs, i.e. computing approximately optimal policies. To achieve our fastest running times we combine classic sampling techniques for value iteration with new methods associated with variance reduction to obtain faster running times. Our fastest algorithms leverage further insights to ensure that our algorithms make monotonic progress towards the optimal value. (See Section 1.2 for a more detailed overview of our approach.)

Our main results are an algorithm that computes an ϵ-approximate policy with probability $1 - \delta$ in time[2]

$$\tilde{O}\left(\left(|S|^2|A| + \frac{|S||A|}{(1-\gamma)^3}\right) \log\left(\frac{M}{\epsilon}\right) \log\left(\frac{1}{\delta}\right)\right) ,$$

and another algorithm which given a data structure from which we can sample state-action-state tuple according to its transition probability in expected $O(1)$ time,[3] computes an ϵ-approximate policy in time

$$\tilde{O}\left(\frac{|S||A|M^2}{(1-\gamma)^4\epsilon^2} \log\left(\frac{1}{\delta}\right)\right) .$$

The first algorithm is nearly linearly convergent, i.e., has an $\tilde{O}(\log(1/\epsilon))$ dependence on ϵ, and runs in nearly-linear time whenever $1/(1 - \gamma) = O(|S|^{1/3})$. This is the first algorithm for solving DMDPs that provably runs in nearly linear time even when the discount factor can be polynomial in the number of states and actions (albeit with a small exponent). Notably, it either matches or improves upon the performance of various forms of value iteration [Tse90, LDK95] in terms of the dependence on $|S|$, $|A|$, and γ.[4]

The second algorithm is sublinear in the $\Omega(|S|^2|A|)$ input size of the DMDP as long as $\frac{1}{\epsilon^2} = o(\frac{|S|(1-\gamma)^4}{M^2})$. It improves upon existing results on sampling-based value iteration (Q-learning) in its dependence on $1/(1 - \gamma)$ [KS99]. The algorithm matches the running time of [Wan17] under fewer assumptions (i.e. we make no assumptions regarding ergodicity).

We also extend these two algorithms to solve finite horizon MDPs. To the best of our knowledge, our algorithms are the first application of variance reduction for approximate value iteration to solve for approximately optimal policies for finite horizon MDPs. In Table 1 and Table 2 we provide a more comprehensive comparison of running time and in Section 2 we provide a more comprehensive discussion of previous work.

1.2 Approach Overview

We achieve our results by building on the classic *value iteration* algorithm. This algorithm simply computes a sequence of values $v_k \in \mathbb{R}^S$, $k = 0, 1, \ldots$, by applying the rule

$$[v_{k+1}]_i = \max_{a \in A}\left[r_a(i) + \gamma \cdot p_a(i)^\top v_k\right]$$

for all $i \in S$ and $k \geq 0$, and using the maximizing action as the current policy, where we denote by $p_a(i) \in \mathbb{R}^S$ the vector of transition probabilities given by $p_a(i) = [p_a(i, j)]_{j \in S}$. In our adaption, instead of

---

[2]We use $\tilde{O}$ to hide polylogarithmic factors in the input parameters, i.e. $\tilde{O}(f(x)) = O(f(x) \cdot \log(f(x))^{O(1)})$.

[3]The actual time it takes to sample by transition probabilities depends on the precise arithmetic model, assumptions about how the input is given, and what preprocessing is performed. For example with $\tilde{O}(|S|^2|A|)$ preprocessing, transitions can be stored in binary trees that allow sampling to be performed in $O(\log(|S|))$ time. However by using arrays and assuming indexing takes $O(1)$ time, faster sampling times can be achieved. We assume the sampling time is $O(1)$ to simplify notation. If sampling instead takes time $O(\alpha)$ this increases our sublinear running times by at most a multiplicative $O(\alpha)$.

[4]This algorithm also uses an oracle for sampling transition probabilities in expected $O(1)$ time. Since this can be implemented with $\tilde{O}(|S|^2|A|)$ preprocessing time, it does not affect the algorithm's asymptotic runtime and no sampling assumption is needed.

exactly computing $p_a(i)^\top v_k$, we approximate it by sampling. We analyze the performance using Hoeffding's inequality and well-known contraction properties of value iteration. While this is a fairly classic idea, we show how to improve it by using a type of *variance reduction*.

More formally, instead of sampling to compute $p_a(i)^\top v$ for current values $v \in \mathbb{R}^S$ and all actions $a \in A$ and $i \in S$ from scratch in every iteration, we show that we can decrease the number of samples needed by instead estimating how these quantities change over time. Our faster variance reduced algorithms first compute fairly precise estimates for $p_a(i)^\top v_0$ for initial values $v_0 \in \mathbb{R}^S$ and all actions $a \in A$ and $i \in S$ and then, when after k iterations the current value vector is v_k, sampling is used to estimate $p_a(i)^\top(v_k - v_0)$ for all actions $a \in A$ and $i \in S$. Adding these estimated changes to the previously estimated values of $p_a(i)^\top v$ is our approximate value iterate.

By sampling the change in $p_a(i)^\top v_k$ from some fixed $p_a(i)^\top v_0$, fewer samples are needed, compared to the sampling complexity of simply estimating $p_a(i)^\top v_k$, due to lower variance. We show that as with other applications of variance reduction, this scheme ties the incurred error to the current quality of the v_k. Exploiting this fact and carefully trading off how often the $p_a(i)^\top v_0$ are estimated and how the differences $p_a(i)^\top(v_k - v_0)$ are estimated, is what leads to our reductions in running time. Variance reduction has recently become a popular technique for obtaining faster algorithms for a wide range of optimization problems, and our application is the first for Markov Decision Processes [JZ13].

To further accelerate the sampling algorithm for computing ϵ-optimal policies, we modify our algorithm to start with an underestimate of the true values and increase them monotonically towards the optimum. We show that this allows us to tighten our runtime bounds and analyses in two ways. First, the fact that values are always increasing directly reduces the number of samples required. Secondly, it helps us maintain the invariant that the current proposed stationary policy π and values v satisfy the inequality $v \leq v_\pi$ entry-wise. This ensures that the induced policy has true values greater then v and thus allows us to convert from approximate values to approximate policies without a loss in approximation quality.

We hope these techniques will be useful in the development of even faster MDP algorithms in theory and practice. We are unaware of variance reduction as employed by this algorithm having been used previously to obtain linearly convergent algorithms for linear programming and thus we hope that the analysis of this paper may open the door for faster algorithms for a broader set of convex optimization problems.

2 Previous Work

The design and analysis of algorithms for solving MDPs have interested researchers across multiple disciplines since the 1950s. There are three major deterministic approaches: value iteration, policy iteration method, and linear programming, for finding the exact optimal stationary policy of an MDP [Ber95]. Later in 1990s, sampling-based methods for MDP began to gain traction, and lay the foundation for reinforcement learning algorithms today. Despite years of study, the complexity of MDP remains an open question and the best known running times are far from linear.

Deterministic Methods for MDP Bellman [Bel57] developed value iteration as a successive approximation method to solve nonlinear fixed-point equations. Its convergence and complexity have been thoroughly analyzed; see e.g. [Tse90, LDK95]. It is known that value iteration can compute an exact solution to a DMDP in time $O(|S|^2|A|L\frac{\log(1/(1-\gamma))}{1-\gamma})$, where L is a measure of complexity of the associated linear program that is at most the number of bits needed to represent the input. It is also well known that value iteration can find an approximate ϵ-approximate solution in time $O(|S|^2|A|\frac{\log(1/\epsilon(1-\gamma))}{1-\gamma})$.

Howard introduced policy iteration shortly thereafter [How60], and its complexity has also been analyzed extensively; see e.g. [MS99, Ye11, Sch13]. Not long after the development of value iteration and policy iteration, [d'E63] and [DG60] discovered that the Bellman equation can be formulated into an equivalent linear program, allowing a rich suite of tools developed for LP, such as interior point methods and the simplex method by Dantzig [Dan16], to solve MDPs exactly. This connection also led to the insight that the simplex method, when applied to solving DMDPs, is the simple policy iteration method.

[FH14] showed that value iteration is not strongly polynomial for DMDP. However, Ye [Ye11] showed that policy iteration (which is a variant of the general simplex method for linear programming) and the simplex method are strongly polynomial for DMDP and terminates in $O(\frac{|S|^2|A|}{1-\gamma}\log(\frac{|S|}{1-\gamma}))$ number of iterations. [HMZ13] and [Sch13] improved the iteration bound to $O(\frac{|S||A|}{(1-\gamma)}\log(\frac{1}{(1-\gamma)}))$ for Howard's policy iteration method. [Ye05] also designed a combinatorial interior-point algorithm (CIPA) that solves the DMDP in strongly polynomial time.

Recent developments [LS14, LS15a] showed that linear programs can be solved in $\tilde{O}(\sqrt{\text{rank}(A)})$ number of linear system solves, which, applied to DMDP, leads to a running time of $\tilde{O}(|S|^{2.5}|A|L)$ and $\tilde{O}(|S|^{2.5}|A|\log(M/((1-\gamma)\epsilon)))$ (see Appendix B for a derivation).

We also note that while there are many methods for approximate linear programming, the error they incur for solving DMDPs is unclear as care needs to be taken in converting between ϵ-approximate linear programming solutions and ϵ-approximate values and policies (note that ϵ-approximate values does not necessarily lead to ϵ-approximate policies). For an illustrative example, see Appendix B where we show how convert a particular type of approximate linear programming solution to an approximate policy and use interior point methods to compute approximate policies efficiently.

For more detailed surveys on MDP and its solution methods, we refer the readers to the textbooks [Ber95, BT95, Put14, Ber13] and the references therein. In particular, our treatment of the Bellman operator and its properties follows the style set by [Ber95, Ber13] for deterministic value iteration.

Value Iteration	$|S|^2|A|L\frac{\log(1/(1-\gamma))}{1-\gamma}$	[Tse90, LDK95]
Policy Iteration (Block Simplex)	$\frac{|S|^4|A|^2}{1-\gamma}\log(\frac{1}{1-\gamma})$	[Ye11],[Sch13]
Recent Interior Point Methods	$\tilde{O}(|S|^{2.5}|A|L)$	[LS14]
Combinatorial Interior Point Algorithm	$|S|^4|A|^4\log\frac{|S|}{1-\gamma}$	[Ye05]

Table 1: **Running Times to Solve DMDPs Exactly**: In this table, $|S|$ is the number of states, $|A|$ is the number of actions per state, $\gamma \in (0,1)$ is the discount factor, and L is a complexity measure of the linear program formulation that is an upper bound on the total bit size to present the DMDP input.

Sampling-Based Methods for MDP   Q-learning was the first sampling-based method for MDP, initially developed for reinforcement learning. Q-learning methods are essentially sampling-based variants of value iteration. [KS99] proved that phased Q-learning takes $\tilde{O}(\frac{|S||A|}{\epsilon^2})$ sample transitions to compute an ϵ-optimal policy, where the dependence on γ is left unspecified. No runtime analysis is given explicitly for phased Q-learning in [KS99], because in reinforcement learning settings, samples are observed experiences, and so the cost of sampling is not easily quantified.

There is a large body of work on sampling-methods for MDPs in the literature of reinforcement learning, see e.g., [SLW+06, SLL09, LH12, AMK12, AMK13] and many others. These works study learning algorithms that update parameters by drawing information from some oracle, where the sampling oracles and

Value Iteration	$|S|^2|A|\frac{\log(1/(1-\gamma)\epsilon)}{1-\gamma}$	[Tse90, LDK95]
Recent Interior Point Methods	$\tilde{O}\left(|S|^{2.5}|A|\log\left(\frac{M}{(1-\gamma)\epsilon}\right)\right)$	[LS14] (Appendix B)
Randomized Primal-Dual Method	$\tilde{O}(C\frac{|S||A|M^2}{(1-\gamma)^4\epsilon^2})$	[Wan17]
Empirical QVI	$\tilde{O}\left(\frac{|S||A|M^2}{(1-\gamma)^3\epsilon^2}\right)$ if $\epsilon = \tilde{O}\left(\frac{1}{\sqrt{(1-\gamma)|S|}}\right)$	[AMK13]
High Precision Randomized Value Iteration	$O\left(\left(|S|^2|A| + \frac{|S||A|}{(1-\gamma)^3}\right)\log\left(\frac{M}{\epsilon\delta}\right)\right)$	This Paper
Sublinear Randomized Value Iteration	$\tilde{O}\left(\frac{|S||A|M^2}{(1-\gamma)^4\epsilon^2}\right)$	This Paper

Table 2: **Running Times to Compute ϵ-Approximate Policies in DMDPs with High Probability**: In this table, $|S|$ is the number of states, $|A|$ is the number of actions per state, $\gamma \in (0,1)$ is the discount factor, M is an upper bound on the absolute value of any reward, and C is an upper bound on the ergodicity.

modeling assumptions vary. The focus of this research is usually the sample complexity of learning; they are typically not concerned with the explicit runtime complexity, since the cost of sampling and processing a sample transition is nebulous in applied reinforcement learning settings. Our randomized algorithm is related to reinforcement learning under a generative model that allows the agent to sample transitions conditioned on specified state-action pairs, for examples, see [AMK12, KMN02, AMK13].

One notable contribution is the work of [AMK13], which introduces policy iteration and value iteration algorithms for generative models that achieve the optimal sample complexity for finding ϵ-optimal value functions, rather than ϵ-optimal policies, as well as the matching lower bound. Their algorithms also achieve optimal sample complexity and optimal runtime for finding ϵ-optimal policies in a restricted ϵ regime.

A recent related work [Wan17] proposed a randomized mirror-prox method with adaptive transition sampling, which applies to a special saddle point formulation of the Bellman equation. They achieve a total runtime of $\tilde{O}(\frac{|S|^3|A|M^2}{(1-\gamma)^6\epsilon^2})$ for the general DMDP and $\tilde{O}(C\frac{|S||A|}{(1-\gamma)^4\epsilon^2})$ for DMDPs that are ergodic under all possible policies, where C is a DMDP-specific ergodicity measure.

Summary   Table 1 and Table 2 summarize the best-known running-time complexity of solution methods for DMDP. The running-time complexity is in terms of the total number of arithmetic operations. Compared to deterministic methods and sampling-based methods, our main result has the sharpest dependence on the input dimensions $|S|$ and $|A|$ and discount factor γ. In terms of lower bounds for the DMDP, [CW17] recently showed that the runtime complexity for any randomized algorithm is $\Omega(|S|^2|A|)$. In the case where each transition can be sampled in $\tilde{O}(1)$ time, [CW17] showed that any randomized algorithm needs $\Omega(\frac{|S||A|}{\epsilon})$ runtime to produce an ϵ-optimal policy with high probability. [AMK13] also shows a sample complexity lower bound of $\Omega(\frac{|S||A|}{(1-\gamma)^3\epsilon^2})$ for finding optimal policies under the generative model. In both the general and restricted cases, our main result nearly matches the lower bounds in its dependence on $|S|$ and $|A|$.

3   Preliminaries

We describe a DMDP by the tuple (S, A, P, r, γ), where S is the finite state space, A is the finite action space, P is the collection of state-action-state transition probabilities, r is the collection of state-action rewards, and $\gamma \in (0,1)$ is a discount factor. We use $p_a(i,j)$ to denote the probability of going to state j

from state i when taking action a and define $p_a(i) \in \mathbb{R}^S$ with $p_a(i)_j \stackrel{\text{def}}{=} p_a(i, j)$ for all $j \in S$. We use $r_a(i)$ to denote the reward obtained from taking action $a \in A$ at state $i \in S$ and assume that for some known $M > 0$ it is the case that all $r_a(i) \in [-M, M]$.

We make the assumption throughout that for any state $i \in S$ and action $a \in A$ we can sample $j \in S$ independently at random so that $\Pr[j = k] = p_a(i, k)$ in expected $O(1)$ time. This is a natural assumption under standard arithmetic models of computation that can be satisfied by preprocessing the DMDP in linear, i.e. $O(|S|^2|A|)$ time.[5] For further discussion of sampling schemes, see [Wan17].

We also use $\vec{1}$ and $\vec{0}$ to denote the all ones and all zeros vectors, respectively.

In the remainder of this section we give definitions for several fundamental concepts in DMDP analysis that we use throughout the paper.

Definition 3.1 (Value Operator). For a given DMDP the *value operator* $T : \mathbb{R}^S \mapsto \mathbb{R}^S$ is defined for all $u \in \mathbb{R}^S$ and $i \in S$ by

$$T(u)_i = \max_{a \in A} \left[r_a(i) + \gamma \cdot p_a(i)^\top u \right] , \tag{3.1}$$

and we let v^* denote the *value of the optimal policy* π^*, which is the unique vector such that $T(v^*) = v^*$.

Definition 3.2 (Policy). We call any vector $\pi \in A^S$ a *policy* and say that the action prescribed by policy π to be taken at state $i \in S$ is π_i. We let T_π denote the *value operator associated with* π defined for all $u \in \mathbb{R}^S$ and $i \in S$ by

$$[T_\pi(u)]_i = r_{\pi_i}(i) + \gamma \cdot p_{\pi_i}(i)^\top u ,$$

and we let v_π denote the *values of policy* π, which is the unique vector such that $T_\pi(v_\pi) = v_\pi$.

Note that T_π can be viewed as the value operator for the modified MDP where the only available action from each state is given by the policy π. Note that this modified MDP is essentially just an uncontrolled Markov Chain.

Definition 3.3 (ϵ-optimality). We say values $u \in \mathbb{R}^S$ are ϵ-*optimal* if $\|v^* - u\|_\infty \leq \epsilon$ and we say a policy $\pi \in A^S$ is ϵ-*optimal* if $\|v^* - v_\pi\|_\infty \leq \epsilon$, i.e. the values of the policy are ϵ-optimal.

3.1 Value Iteration Facts

We review basic facts regarding value iteration that we use in our analysis. These are well established in the literature and for more details, please see Appendix A.

The first fact is that the value operator T is a contraction mapping, meaning that the operator brings value vectors closer together. This is key to establishing the convergence and correctness of value iteration:

Lemma 3.4 (Contraction Mapping). *For all values $u, v \in \mathbb{R}^S$ we have that $\|T(u) - T(v)\|_\infty \leq \gamma \|u - v\|_\infty$ and consequently $\|T(u) - v^*\|_\infty \leq \gamma \|u - v^*\|_\infty$, where v^* is the optimal value vector.*

The second fact is that we can bound how close values are to being the value of a policy by how much the value operator for that policy moves the values:

Lemma 3.5. *For any policy $\pi \in A^S$ and values $u \in \mathbb{R}^S$ it holds that $\|T_\pi(u) - v_\pi\|_\infty \leq \frac{\gamma}{1-\gamma}\|T_\pi(u) - u\|_\infty$, where v_π is the exact value vector for policy π.*

---

[5] As discussed in the introduction if instead sampling required $O(\log |S|)$ time, which is easily achieved with $\tilde{O}(|S|^2|A|)$ preprocessing, this would increase our running times by only a multiplicative $O(\log |S|)$ which would be hidden by the $\tilde{O}(\cdot)$ notation, leaving running times unaffected.

The last fact is that the value operator is monotonic in the sense that it preserves the property that one value vector may be larger than another entry-wise.

Lemma 3.6. *If values $u, v \in \mathbb{R}^S$ satisfy $u \leq v$ entry-wise, then $T(u) \leq T(v)$ entry-wise.*

4 DMDP Algorithms

In this section, we present our algorithms and the corresponding analysis for solving DMDPs. We split the presentation of our algorithms and its analysis into multiple pieces as follows:

- Section 4.1: we present ApxVal, our randomized, approximate value iteration sub-routine.

- Section 4.2: we introduce a simple randomized value iteration scheme, RandomizedVI, which approximates the value operator using ApxVal for each iteration. We analyze its convergence and correctness guarantees.

- Section 4.3: we use RandomizedVI to create a high precision randomized value iteration algorithm, HighPrecisionRandomVI, which returns an ϵ-optimal value vector in nearly linear time.

- Section 4.4: we present a sublinear time randomized value iteration algorithm, SublinearRandomVI, which returns an ϵ-optimal value vector in sublinear time.

- Section 4.5: we show how to compute an ϵ-optimal policy using our ϵ-optimal value vectors.

- Section 4.6: we present a monotonic value operator, ApxMonVal, that further improves the runtime of our sublinear randomized value iteration algorithm for computing ϵ-approximate policies.

4.1 Approximate Value Operator

Here we introduce our main sub-routine for performing randomized value iterations, ApxVal (See Algorithm 2). The routine ApxVal approximates the value operator by sampling. Instead of computing the value operator exactly, i.e.:

$$T(u)_i = \max_{a \in A} \left[r_a(i) + \gamma \cdot p_a(i)^\top u \right] \tag{4.1}$$

ApxVal approximates $T(u)_i$ by estimating $p_a(i)^\top u$ via sampling and maximizing over the action space based on these estimates.

The sampling procedure we use, ApxTrans is given in Algorithm 1. Its concentration guarantees are standard, but for completeness we give the analysis and the statement of Hoeffding's inequality below.

Algorithm 1 Approximate Transition: $\texttt{ApxTrans}(u, M, i, a, \epsilon, \delta)$

Input: Values $u \in \mathbb{R}^S$ and scalar $M \geq 0$ such that $\|u\|_\infty \leq M$
Input: State $i \in S$ and action $a \in A$
Input: Target accuracy $\epsilon \in (0, 1)$ and failure probability $\delta \in (0, 1)$
1: set $m := \lceil \frac{2M^2}{\epsilon^2} \ln(\frac{2}{\delta}) \rceil$
2: **for** each $k \in [m]$ **do**
3:    choose $i_k \in S$ independently with $\Pr[i_k = j] = p_a(i, j)$.
4: **return** sample average $S = \frac{1}{m} \sum_{k \in [m]} u_{i_k}$

Theorem 4.1 (Hoeffding's Inequality ([Hoe63] Theorem 2)). *Let $X_1, ..., X_m$ be independent real valued random variables with $X_i \in [a_i, b_i]$ for all $i \in [m]$ and let $Y = \frac{1}{m} \sum_{i \in [m]} X_i$. For all $t \geq 0$,*

$$\mathbb{P}[|Y - \mathbb{E}[Y]| \geq t] \leq 2 \exp \left(\frac{-2m^2 t^2}{\sum_{i \in [m]} (b_i - a_i)^2} \right).$$

Lemma 4.2 (Sample Concentration). *`ApxTrans` (See Algorithm 1) can be implemented to run in time $O(M^2 \epsilon^{-2} \log(1/\delta))$ and outputs $Y \in \mathbb{R}$ such that $|Y - p_a(i)^\top u| \leq \epsilon$ with probability $1 - \delta$.*

Proof. Note that Y and u_{i_k} were chosen so that $\mathbb{E}[Y] = p_a(i)^\top u$. Furthermore, since $u_{i_k} \in [-M, M]$ by the assumption that $\|u\|_\infty \leq M$ we have by Hoeffding's Inequality, Theorem 4.1, and choice of m that

$$\mathbb{P}\left[|Y - p_a(i)^\top u| \geq \epsilon\right] = \mathbb{P}[|Y - \mathbb{E}Y| \geq \epsilon] \leq 2 \cdot \exp \left(\frac{-2m^2 \epsilon^2}{m(2M)^2} \right) \leq \delta.$$

Since the algorithm simply takes $O(m)$ samples and outputs their average the running time is $O(m)$. $\square$

Using `ApxTrans` (Algorithm 1) our approximate value operator `ApxVal` is given in Algorithm 2. This algorithm works as previously described, using sampling to approximate $p_a(i)^\top u$ and then using these samples to approximately maximize over the space of actions.

Algorithm 2 Approximate Value Operator: `ApxVal`$(u, v_0, x, \epsilon, \delta)$

Input: Current values, $u \in \mathbb{R}^S$, and initial value: $v_0 \in \mathbb{R}^S$
Input: Precomputed offsets: $x \in \mathbb{R}^{S \times A}$ with $|x_a(i) - p_a(i)^\top v_0| \leq \epsilon$ for all $i \in S$ and $a \in A$.
Input: Target accuracy $\epsilon \in (0, 1)$ and failure probability $\delta \in (0, 1)$
1: compute $M = \|u - v_0\|_\infty$
2: **for** each state $i \in S$ **do**
3:      **for** each action $a \in A$ **do**
4:          set $\tilde{S}_a(i) = x_a(i) + $ `ApxTrans`$(u - v_0, M, i, a, \epsilon, \delta/(|S||A|))$
5:          set $\tilde{Q}_a(i) = r_a(i) + \gamma \cdot \tilde{S}_a(i)$
6:      set $\tilde{v}_i = \max_{a \in A} \tilde{Q}_a(i)$ and $\pi_i \in \text{argmax}_{a \in A} \tilde{Q}_a(i)$
7: **return** $(\tilde{v}, \pi)$
8: ($\tilde{v} \in \mathbb{R}^S$ *is the result of an approximate value iteration and $\pi \in A^S$ is the corresponding policy*).

`ApxVal` is designed to enable variance reduction. Instead of sampling values of u directly and using the sample average, `ApxVal` samples the difference from some fixed value vector input $v_0 \in \mathbb{R}^S$ and assumes that the value of $p_a(i)^\top v_0$ is approximately given through input $x_{i,a}$. In particular, `ApxVal` samples from $u - v_0$ and computes their average plus the offset $x_{i,a}$ which we denote by $S_a(i)$. Note that if we take v_0 to be the 0 vector, this is just standard naive sampling. However, by invoking `ApxVal` with intelligently chosen v_0 and x we are able to take less samples and still obtain precise estimates of $S_a(i)$. The fact we ensure v_0 is closer to v and therefore the variance in our sampling is smaller and less samples are needed, is what we refer to as variance reduction and is crucial in achieving our best runtime results.

In the following lemma we analyze `ApxVal` (Algorithm 2) and show that it approximates the value operator. The lemma is used repeatedly in later analysis.

Lemma 4.3 (Approximate Value Operator). *`ApxVal` (Algorithm 2) can be implemented to run in time*

$$O\left(|S||A|\left\lceil \|u - v_0\|_\infty^2 \epsilon^{-2} \ln(|S||A|/\delta) \right\rceil\right)$$

9

and with probability $1 - \delta$ returns values $\tilde{v} \in \mathbb{R}^S$ and policy $\pi \in A^S$ such that $\|\tilde{v} - T_\pi(u)\|_\infty \leq 2\gamma\epsilon$ and $\|\tilde{v} - T(u)\|_\infty \leq 2\gamma\epsilon$.

Proof. The running time follows from the running time bound for `ApxTrans` given by Lemma 4.2 and the facts that we can compute $\|u - v_0\|_\infty$ in $O(|S|)$ time and perform the remaining arithmetic and maximization operations in $O(|S||A|)$ time.

By applying union bound to $|S||A|$ applications of `ApxTrans` as analyzed by Lemma 4.2 we know that with probability at least $1 - \delta$ for all $i \in S$ and $a \in A$ we have that

$$|\tilde{S}_a(i) - x_a(i) - p_a(i)^\top (u - v_0)| \leq \epsilon .$$

By triangle inequality, in this case we have that for all $i \in S$ and $a \in A$

$$|\tilde{S}_a(i) - p_a(i)^\top u| \leq |\tilde{S}_a(i) - x_a(i) - p_a(i)^\top (u - v_0)| + |x_a(i) - p_a(i)^\top v_0| \leq \epsilon + \epsilon \leq 2\epsilon . \qquad (4.2)$$

Now for all $i \in S$ and $a \in A$ let

$$Q_a(i) \overset{\text{def}}{=} r_a(i) + \gamma \cdot p_a(i)^\top u \text{ and } v_i \overset{\text{def}}{=} \max_{a \in A}[Q_a(i)] .$$

Note that $[T(u)]_i = v_i$ and $[T_\pi(u)]_i = Q_{\pi_i}(i)$. Now assuming (4.2) holds we know that

$$|Q_a(i) - \tilde{Q}_a(i)| \leq 2\gamma\epsilon$$

for all $a \in A$, $i \in S$. This implies

$$\|\tilde{v} - T_\pi(u)\|_\infty = \max_{i \in S} |\tilde{Q}_{\pi_i}(i) - Q_{\pi_i}(i)| \leq 2\gamma\epsilon .$$

Furthermore, this bound implies that for all $i \in S$

$$\tilde{v}_i = \max_{a \in A} \tilde{Q}_a(i) \leq \max_{a \in A} Q_a(i) + 2\gamma\epsilon = [T(u)]_i + 2\gamma\epsilon$$

and similarly, $\tilde{v}_i \geq [T(u)]_i - 2\gamma\epsilon$. Consequently, $|\tilde{v}_i - [T(u)]_i| \leq 2\gamma\epsilon$ and $\|\tilde{v} - T(u)\|_\infty \leq 2\gamma\epsilon$. $\qquad \square$

In the following lemma we show that the guarantees of Lemma 4.3 suffice to prove that `ApxVal` is approximately contractive. This connects the analysis of `ApxVal` to standard value iteration and is key for establishing convergence and correctness for our algorithms.

Lemma 4.4 (Approximate Contraction). *If values $u, v \in \mathbb{R}^S$ satisfy $\|v - T(u)\|_\infty \leq \alpha$ then*

$$\|v - v^*\|_\infty \leq \alpha + \gamma\|u - v^*\|_\infty .$$

Proof. By triangle inequality and the optimality of v^* we know that

$$\|v - v^*\|_\infty = \|v - T(v^*)\|_\infty \leq \|v - T(u)\|_\infty + \|T(u) - T(v^*)\|_\infty .$$

Now, $\|v - T(u)\|_\infty \leq \alpha$ by assumption, and $\|T(u) - T(v^*)\|_\infty \leq \gamma\|u - v^*\|_\infty$ by the fact that T is a γ-contraction map (See Lemma 3.4). Combining yields the desired result. $\qquad \square$

Algorithm 3 Randomized VI: `RandomizedVI(`v_0, L, ϵ, δ`)`

Input: Initial values $v_0 \in \mathbb{R}^S$ and number of iterations $L > 0$

Input: Target accuracy $\epsilon \geq 0$ and failure probability $\delta \in (0, 1)$

1: `Compute` $x \in \mathbb{R}^{S \times A}$ `such that` $x_a(i) = p_a(i)^\top v_0$ `for all` $i \in S$ `and` $a \in A$

2: **for** `each iteration` $l \in [L]$ **do**

3: $\quad (v_l, \pi_l) = $ `ApxVal(`$v_{l-1}, v_0, x, \epsilon, \delta/L$`)`

4: **return** (v_K, π_K)

4.2 Basic Approximate VI Analysis

Here we introduce a simple randomized value iteration scheme, `RandomizedVI`, which approximates the value operator using `ApxVal` in each iteration. We analyze its convergence and correctness guarantees here.

Lemma 4.5 (Quality of Randomized VI). *With probability* $1 - \delta$*, an invocation of* `RandomizedVI` *(Algorithm 3) produces* $v_K \in \mathbb{R}^S$ *and* $\pi_K \in A^S$ *such that*

$$\|v_L - v^*\|_\infty \leq \frac{2\epsilon\gamma}{1 - \gamma} + \exp(-L(1 - \gamma)) \cdot \|v_0 - v^*\|_\infty .$$

Consequently, if $L \geq \lceil \frac{1}{1-\gamma} \log(\|v_0 - v^*\|_\infty/(2\epsilon\gamma)) \rceil$ *then* $\|v_L - v^*\|_\infty \leq \frac{4\epsilon\gamma}{1-\gamma}$.

Proof. By the analysis of `ApxVal`, Lemma 4.3, and union bounding over its L invocation, we have that with probability $1 - \delta$, $\|v_l - T(v_{l-1})\|_\infty \leq 2\gamma\epsilon$ for all $l \in [L]$. In this case, by Lemma 4.4, we have that each iteration of `RandomizedVI` is approximately contractive with $\alpha = 2\epsilon\gamma$, i.e.

$$\|v_l - v^*\|_\infty \leq 2\epsilon\gamma + \gamma\|v_{l-1} - v^*\|_\infty$$

for all $l \in [L]$. Consequently, for all $l \geq 0$ we have

$$\|v_l - v^*\|_\infty \leq \sum_{l \in [L]} \gamma^l \cdot 2\epsilon\gamma + \gamma^l \cdot \|v_0 - v^*\|_\infty \leq \frac{2\epsilon\gamma}{1 - \gamma} + \exp(-l(1 - \gamma)) \cdot \|v_0 - v^*\|_\infty ,$$

where we used that $\sum_{l \in [L]} \gamma^l \leq \sum_{l=1}^{\infty} \gamma^l = \frac{1}{1-\gamma}$ and $\gamma = (1 - (1 - \gamma)) \leq \exp(-(1 - \gamma))$.

The final claim follows from the fact that if $L \geq \frac{1}{1-\gamma} \cdot \log(\|v_0 - v^*\|_\infty/2\epsilon\gamma)$ then

$$\frac{1}{1 - \gamma} \cdot 2\epsilon\gamma + \exp(-L(1 - \gamma)) \cdot \|v_0 - v^*\|_\infty \leq \frac{1}{1 - \gamma} \cdot 2\epsilon\gamma + 2\epsilon\gamma \leq \frac{4\epsilon\gamma}{1 - \gamma} .$$

$\qquad\square$

Lemma 4.6 (Randomized VI Runtime). `RandomizedVI` *(Algorithm 3) can be implemented to run in time*

$$O\left(|S|^2|A| + L|S||A| \left[\frac{\|v_0 - v^*\|_\infty^2}{\epsilon^2} + \frac{1}{(1 - \gamma)^2} \right] \log\left(\frac{|S||A|L}{\delta} \right) \right) .$$

Proof. Note that x can be computed naively in time $O(|S|^2|A|)$. To bound the remaining running time we appeal to Lemma 4.4, with $\alpha = 2\epsilon\gamma$, as justified by Lemma 4.3. We have $\|v_l - v^*\|_\infty \leq \frac{2\epsilon\gamma}{1-\gamma} + \|v_0 - v^*\|_\infty$

for all $l \in [0, L]$. Consequently by triangle inequality and $(a + b)^2 \le 2a^2 + 2b^2$ for all $a, b \in \mathbb{R}$:

$$\|v_l - v_0\|_\infty^2 \le \|v_l - v^* + v^* - v_0\|_\infty^2 \le (\|v_l - v^*\|_\infty + \|v^* - v_0\|_\infty)^2 \le 2\|v_k - v^*\|_\infty^2 + 2\|v^* - v_0\|_\infty^2$$

$$\le 2\left(\frac{2\epsilon}{1 - \gamma}\right)^2 + 2\left(2\|v_0 - v^*\|\right)^2 = \frac{8\epsilon^2}{(1 - \gamma)^2} + 8\|v_0 - v^*\|_\infty^2$$

The result then follows from the running time of `ApxVal` given by Lemma 4.3. $\square$

Note that this analysis applies immediately to classic approximate value iteration algorithms. In particular, this analysis can extend the one provided by [KS99] for phased Q-learning, to provide a total runtime guarantee in the restricted setting where (as assumed) taking a sample costs $O(1)$ time in expectation.

Theorem 4.7 (Simple Randomized VI). *With probability $1 - \delta$ for $\epsilon < \frac{M}{1-\gamma}$[6] an invocation of*

$$\texttt{RandomizedVI}\left(\vec{0}, \left\lceil \frac{1}{1 - \gamma} \log\left(\frac{2M}{(1 - \gamma)^2 \epsilon}\right)\right\rceil, \frac{(1 - \gamma)\epsilon}{4\gamma}, \delta\right)$$

(See Algorithm 3), produces $v_L \in \mathbb{R}^S$ such that $\|v_L - v^*\|_\infty \le \epsilon$ and can be implemented to run in time

$$\tilde{O}\left(\frac{|S||A|M^2}{(1 - \gamma)^5 \epsilon^2} \log\left(\frac{1}{\epsilon}\right)\right).$$

Proof. In this case $x = \vec{0}$ and therefore computing it does not contribute to the running time, i.e. we do not have an additive $|S|^2|A|$ term (also by our assumption we can sample in $O(1)$ expected time from the transition function). To bound the rest of the running time, note that $\|v^*\|_\infty \le \frac{M}{1-\gamma}$ and therefore $\|\vec{0} - v^*\|_\infty \le \frac{M}{1-\gamma}$. Consequently, the result follows from Lemma 4.5 and Lemma 4.6. $\square$

In the next several sections we show how to improve upon this result.

4.3 High Precision Randomized VI in Nearly Linear Time

In this section we improve on `RandomizedVI` to obtain our best running times for computing an ϵ-optimal value vector for small values of ϵ. This improvement stems from the insight that the distance between the current value vector and the optimal value v^* as `RandomizedVI` progresses can be bounded by the approximate contraction property of `ApxVal`, the subroutine that `RandomizedVI` uses for approximate value iteration. Hence, in later iterations when the value vector is quite close to the optimal value, a more efficient sample complexity is sufficient to achieve similar concentration guarantees.

Our algorithm, `HighPrecisionRandomVI`, is quite simple. In each iteration the algorithm calls `RandomizedVI` for a decreasing error requirement, ϵ_k. In the first iteration when the distance between the current value vector and the optimal value is high, `HighPrecisionRandomVI` simply decreases the distance to optimal value vector by a factor of $\frac{1}{2}$. Then `HighPrecisionRandomVI` appeals to `RandomizedVI` again using the new value vector as the initial vector input, knowing that because this new vector is much closer to v^*, subsequent approximate value iterations will require fewer samples to achieve an error $\frac{1}{4}$ of the original. This process is repeated so that the cost of each iteration is less than the same upper bound but so that after k iterations the error is a $1/2^k$ fraction of the original. Ultimately this allows `RandomizedVI` to produce an ϵ-optimal value vector in nearly linear time.

---

[6]Note that if ϵ is larger than $\frac{M}{1-\gamma}$, a naive solution value vector $\vec{0}$ would suffice, since $\|\vec{0} - v^*\|_\infty \le \epsilon$

Algorithm 4 High Precision Randomized VI: `HighPrecisionRandomVI(ϵ, δ)`

Input: Target precision ϵ and failure probability $\delta \in (0, 1)$

  1: Let $\ K = \lceil \log_2(\frac{M}{\epsilon(1-\gamma)}) \rceil \ $ and $\ L = \lceil \frac{1}{1-\gamma} \log(\frac{4}{1-\gamma}) \rceil$

  2: Let $\ v_0 = \vec{0} \ $ and $\ \epsilon_0 = \frac{M}{1-\gamma}$

  3: **for** each iteration $k \in [K]$ **do**

  4:      $\epsilon_k = \frac{1}{2}\epsilon_{k-1} = \frac{M}{2^k(1-\gamma)}$

  5:      $(v_k, \pi_k) = \texttt{RandomizedVI}(v_{k-1}, L, (1-\gamma)\epsilon_k/(4\gamma), \delta/K)$

  6: **return** (v_K, π_K)

Lemma 4.8 (Quality and Runtime of High Precision Randomized VI). *With probability $1 - \delta$ in an invocation of* `HighPrecisionRandomVI` *(See Algorithm 4) we have that* $\|v_k - v^*\|_\infty \le \epsilon_k$ *for all* $k \in [0, K]$ *and therefore* v_K *is an ϵ-optimal value vector.*

Proof. We prove by induction on k for that all $k \in [0, K]$ that $\|v_k - v^*\|_\infty \le \epsilon_k$ with $\epsilon_0 = \frac{M}{2^0(1-\gamma)}$ provided that each iteration of `RandomizedVI` meets the criteria of Lemma 4.5. Since by union bound this happens with probability $1 - \delta$ and since $\epsilon_k \le \epsilon$ this suffices to prove the claim.

Note that clearly $v_0 = \vec{0}$ and therefore $\|v_0 - v^*\|_\infty \le \frac{M}{1-\gamma} = \epsilon_0$ and therefore the base case holds. Now suppose that $\|v_{k-1} - v^*\|_\infty \le \epsilon_{k-1}$ for some $k \in [K]$. By Lemma 4.5 and our assumption that the call to `RandomizedVI` succeeds we have that so long as

$$L \ge \left\lceil \frac{1}{1-\gamma} \log \left(\frac{\|v_{k-1} - v^*\|_\infty}{(2[(1-\gamma)\epsilon_k/(4\gamma)]\gamma)} \right) \right\rceil = \left\lceil \frac{1}{1-\gamma} \log \left(\frac{2\|v_{k-1} - v^*\|_\infty}{(1-\gamma)\epsilon_k} \right) \right\rceil$$

which it is as by the assumption that $\|v_{k-1} - v^*\|_\infty \le \epsilon_{k-1} = 2\epsilon_k$, then

$$\|v_k - v^*\|_\infty \le \frac{4[(1-\gamma)\epsilon_k/(4\gamma)]\gamma}{1-\gamma} = \epsilon_k .$$

Consequently, the result follows by induction. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

Lemma 4.9 (Runtime of High Precision Randomized VI). `HighPrecisionRandomVI` *(See Algorithm 4) can be implemented so that with probability $1 - \delta$ it runs in time*

$$\tilde{O}\left(\left(|S|^2|A| + \frac{|S||A|}{(1-\gamma)^3} \right) \log \left(\frac{M}{\epsilon} \right) \log \left(\frac{1}{\delta} \right) \right) .$$

Proof. Each iteration k calls `RandomizedVI`$(v_{k-1}, L, (1-\gamma)\epsilon_k/(4\gamma), \delta/K)$. By Lemma 4.6 we note that the running time of the kth call is

$$O\left(|S|^2|A| + L|S||A| \left[\frac{\|v_{k-1} - v^*\|_\infty}{[(1-\gamma)\epsilon_k/(4\gamma)]^2} + \frac{1}{(1-\gamma)^2} \right] \log \left(\frac{|S||A|L}{\delta} \right) \right) .$$

However, by Lemma 4.6 we know that with probability $1 - \delta$, $\|v_{k-1} - v^*\|_\infty \le \epsilon_{k-1} \le \epsilon_k$ for all k and consequently, the cost of each iteration k is

$$O\left(|S|^2|A| + \frac{|S||A|}{(1-\gamma)^3} \log \left(\frac{1}{1-\gamma} \right) \log \left(\frac{|S||A| \log(\frac{1}{1-\gamma}) \log(\frac{M}{\epsilon(1-\gamma)})}{(1-\gamma)\delta} \right) \right) .$$

Aggregating over the $K = \lceil \log_2(\frac{M}{\epsilon(1-\gamma)}) \rceil$ iterations yields the desired running time. $\qquad\qquad\square$

13

Note that this proof used that the running time of `RandomizedVI` in `HighPrecisionRandomVI` depends on $\|v_{k-1} - v^*\|_\infty^2 / \epsilon_k^2$. By decreasing both $\|v_{k-1} - v^*\|_\infty^2$ and ϵ_k^2 at geometric rates we ensured the cost of `RandomizedVI` remained the same, even though the accuracy it yielded improved. In the next section we show how such variance reduction can be applied to obtain faster sublinear time algorithms.

4.4 Randomized VI in Sublinear Time

Here we present an algorithm, `SampledRandomizedVI` for computing an ϵ-optimal value vector that runs in sublinear time whenever $\frac{1}{\epsilon^2} = \tilde{o}(\frac{|S|(1-\gamma)^4}{M^2})$. Our algorithm is similar to `HighPrecisionRandomVI`; the primary difference is that instead of computing the initial offset x in `ApxVal` exactly in $O(|S|^2|A|)$ time, we compute an an approximation, $\tilde{x}$, to ϵ-accuracy of x in ℓ_∞, ie $\|x - \tilde{x}\|_\infty \le \epsilon$, by sampling in $O(|S||A|\|v_0\|_\infty^2 / \epsilon^2)$ time. Variance reduction is then performed as in `HighPrecisionRandomVI`, to decrease the number of samples need to run the rest of the algorithm.

Algorithm 5 Sampled Randomized VI: `SampledRandomizedVI`$(v_0, L, \epsilon, \delta)$

Input: Initial values $v_0 \in \mathbb{R}^S$ and number of iterations $L > 0$
Input: Target accuracy $\epsilon > 0$ and failure probability $\delta \in (0, 1)$
  1: Sample to obtain approximate offsets: $\tilde{x} \in \mathbb{R}^{S \times A}$ with $|\tilde{x}_a(i) - p_a(i)^\top v_0| \le \epsilon$ for all $i \in S$ and $a \in A$
  2: **for** each round $l \in [L]$ **do**
  3:   $(v_l, \pi_l) = $ `ApxVal`$(v_{l-1}, v_0, \tilde{x}, \epsilon, \delta/L)$
  4: **return** (v_L, π_L)

The analysis for `SampledRandomizedVI` follows exactly from Lemmas 4.5 and 4.6, since `ApxVal` was designed and analyzed for the case where $|\tilde{x}_a(i) - p_a(i)^\top v_0| \le \epsilon$. This condition was trivially satisfied for `RandomizedVI` when we used $x_a(i) = p_a(i)^\top v_0$.

Algorithm 6 Sublinear Time Randomized VI: `SublinearRandomVI`(ϵ, δ)

Input: Target precision ϵ and failure probability $\delta \in (0, 1)$
  1: Let $K = \lceil \log_2(\frac{M}{\epsilon(1-\gamma)}) \rceil$ and $L = \lceil \frac{1}{1-\gamma} \log(\frac{4}{1-\gamma}) \rceil$
  2: Let $v_0 = \vec{0}$ and $\epsilon_0 = \frac{M}{1-\gamma}$
  3: **for** each iteration $k \in [K]$ **do**
  4:   $\epsilon_k = \frac{1}{2}\epsilon_{k-1} = \frac{M}{2^k(1-\gamma)}$
  5:   $(v_k, \pi_k) = $ `SampledRandomizedVI`$(v_{k-1}, L, (1-\gamma)\epsilon_k/(4\gamma), \delta/K)$
  6: **return** (v_K, π_K)

Lemma 4.10 (Quality of Sublinear Randomized VI). *In an invocation of `SublinearRandomVI` (See Algorithm 6) with probability $1 - \delta$ we have that $\|v_k - v^*\|_\infty \le \epsilon_k$ for all $k \in [0, K]$ and therefore v_K is an ϵ-optimal value vector.*

Proof. We can analyze `SampledRandomizedVI` identically to how we analyzed `RandomizedVI`. Therefore we can analyze `SublinearRandomVI` similarly as we analyzed `HighPrecisionRandomVI` in Lemma 4.8 and the result follows. $\square$

We now turn our attention to the runtime of `SublinearRandomVI`.

Lemma 4.11 (Runtime of Sampled Randomized VI). *An invocation of* `SampledRandomizedVI` *(See Algorithm 5) can be implemented to run in time*

$$O\left(|S||A|\left[\frac{M^2}{(1-\gamma)^2\epsilon^2} + L\left(\frac{\|v_0-v^*\|_\infty^2}{\epsilon^2} + \frac{1}{(1-\gamma)^2}\right)\right]\log\left(\frac{|S||A|L}{\delta}\right)\right).$$

Proof. We first claim that an invocation of `SampledRandomizedVI` (See Algorithm 5) can be implemented to run in time

$$O\left(\left[\frac{|S||A|\|v_0\|_\infty^2}{\epsilon^2} + L|S||A|\left(\frac{\|v_0-v^*\|_\infty^2}{\epsilon^2} + \frac{1}{(1-\gamma)^2}\right)\right]\log\left(\frac{|S||A|L}{\delta}\right)\right).$$

This claims largely follows the runtime analysis of `RandomizedVI`, see Lemma 4.6. The only difference in the runtime comes from the fact that instead of computing x exactly (which previously took $O(|S|^2|A|)$ time), here we only need to compute an approximation, $\tilde{x}$, to ϵ-accuracy of x in ℓ_∞, ie $\|x-\tilde{x}\|_\infty \leq \epsilon$ without increasing the failure probability. As we showed in Section 4.1 this can be done by sampling (See Lemma 4.2) in $O(|S||A|\|v_0\|_\infty^2 \log(|S||A|/\delta)/\epsilon^2)$ time. The result then follows as

$$\|v_0\|_\infty^2 \leq (\|v_0-v^*\|_\infty + \|v^*\|_\infty)^2 \leq 2\|v_0-v^*\|_\infty^2 + 2\|v^*\|_\infty^2$$

$$\leq 2\|v_0-v^*\|_\infty^2 + 2\frac{M^2}{(1-\gamma)^2}.$$

where we used that $\|v_*\|_\infty \leq \frac{M}{1-\gamma}$ and $(x+y)^2 \leq 2x^2 + 2y^2$ for all $x,y \in \mathbb{R}$. $\qquad\square$

Lemma 4.12 (Runtime of Sublinear Randomized VI). `SublinearRandomVI` *(See Algorithm 6) can be implemented to run in time*

$$\tilde{O}\left(|S||A|\left(\frac{M^2}{(1-\gamma)^4\epsilon^2} + \frac{1}{(1-\gamma)^3}\right)\log\left(\frac{1}{\delta}\right)\right)$$

Proof. The runtime analysis for `SublinearRandomVI` follows the skeleton of the runtime analysis of `HighPrecisionRandomVI` in Lemma 4.9. Each iteration k calls `SampledRandomizedVI`$(v_{k-1}, L, (1-\gamma)\epsilon_k/4\gamma, \delta/K)$. Note that $L = \lceil\frac{1}{1-\gamma}\log(\frac{4}{1-\gamma})\rceil$, $\epsilon = (1-\gamma)\epsilon_k/4\gamma$ and by Lemma 4.11 with probability $1-\delta$ we have that $\|v_{k-1}-v^*\|_\infty \leq \epsilon_{k-1} \leq 2\epsilon_k$. Substituting these terms into the runtime analysis prescribed by Lemma 4.11 yields that each iteration k costs

$$\tilde{O}\left(|S||A|\left(\frac{M^2}{(1-\gamma)^4\epsilon_k^2} + \frac{1}{(1-\gamma)^3}\right)\log\left(\frac{1}{\delta}\right)\right)$$

Aggregating over $K = \lceil\log_2(\frac{M}{\epsilon(1-\gamma)})\rceil$ iterations and using that $k \leq \lceil\log_2(\frac{M}{\epsilon(1-\gamma)})\rceil$ and therefore $\epsilon_k = \Omega(\epsilon)$ for all $k \in [K]$ yields the desired running time. $\qquad\square$

Remark 4.13. It is unclear which term $\frac{M^2}{(1-\gamma)^4\epsilon^2}$ or $\frac{1}{(1-\gamma)^3}$ will dominate this runtime. Depending on the problem and the value of M, ϵ, and $1-\gamma$ either can be larger. However, we will later show how to modify our algorithms so that our values increase monotonically so that the $1/(1-\gamma)^3$ term can be removed.

4.5 Obtaining a Policy

In this section, we discuss how to leverage the analysis in the previous section to compute approximately optimal policies for DMDPs. We provide fairly general techniques to turn the $O(\epsilon/(1-\gamma))$-approximate value vectors computed by `RandomizedVI` and `SampledRandomizedVI`) into $O(\epsilon/(1-\gamma)^2)$-approximate policy vectors. This yields our fastest nearly linear convergent algorithms for computing ϵ-approximate policies. In the next section we show how to improve upon these techniques and obtain even faster sublinear time algorithms.

We start with the following lemma that essentiall follows from Proposition 2.1.4 in [Ber13].

Lemma 4.14. `RandomizedVI` and `SampledRandomizedVI` with $L \geq 1 + \frac{1}{1-\gamma} \cdot \log(\|v_0 - v^*\|_\infty/2\epsilon\gamma)$ produce a policy π_L that is $16\epsilon/(1-\gamma)^2$-optimal with probability $1 - \delta$.

Proof. By Lemma 4.5 we know that both

$$\|v_L - v^*\|_\infty \leq \frac{4\epsilon\gamma}{1-\gamma} \text{ and } \|v_{L-1} - v^*\|_\infty \leq \frac{4\epsilon\gamma}{1-\gamma} \ .$$

Furthermore, by Lemma 4.3 we know that

$$\|v_L - T_{\pi_L}(v_{L-1})\|_\infty \leq 2\gamma\epsilon \ .$$

Combining these facts and invoking Lemma 3.5 gives

$$\|v_{\pi_L} - v^*\|_\infty \leq \|v_{\pi_L} - T_{\pi_L}(v_{L-1})\|_\infty + \|T_{\pi_L}(v_{L-1}) - v_L\|_\infty + \|v_L - v^*\|_\infty$$

$$\leq \frac{\gamma}{1-\gamma}\|v_{L-1} - T_{\pi_L}(v_{L-1})\|_\infty + 2\gamma\epsilon + \frac{4\epsilon\gamma}{1-\gamma} \ .$$

Furthermore, we have that

$$\|v_{L-1} - T_{\pi_L}(v_{L-1})\|_\infty \leq \|v_{L-1} - v^*\|_\infty + \|v^* - v_L\|_\infty + \|v_L - T_{\pi_L}(v_{L-1})\|_\infty$$

$$\leq \frac{4\epsilon\gamma}{1-\gamma} + \frac{4\epsilon\gamma}{1-\gamma} + 2\gamma\epsilon$$

Combining and using the fact that $\gamma \in (0, 1)$ and $1/(1-\gamma) \geq 1$ yields the result. $\square$

This lemma allows us to immediately claim running times for computing ϵ-approximate policies simply be computing $O(\epsilon(1-\gamma))$-approximate values as before. The proofs of the following two corollaries are immediate from our previous analysis.

Corollary 4.15 (High Precision Approximate Policy Computation). `HighPrecisionRandomVI` *(See Algorithm 4) can be implemented so that with probability $1 - \delta$ it computes an ϵ-approximate policy in time*

$$\tilde{O}\left(\left(|S|^2|A| + \frac{|S||A|}{(1-\gamma)^3}\right)\log\left(\frac{M}{\epsilon}\right)\log\left(\frac{1}{\delta}\right)\right) \ .$$

Corollary 4.16 (Sublinear Time Approximate Policy Computation). `SublinearRandomVI` *(See Algorithm 6) can be implemented so that with probability $1 - \delta$ it computes an ϵ-approximate policy in time*

$$\tilde{O}\left(|S||A|\left(\frac{M^2}{(1-\gamma)^6\epsilon^2} + \frac{1}{(1-\gamma)^3}\right)\log\left(\frac{1}{\delta}\right)\right)$$

In the next section we show how to greatly improve this sublinear running time for computing an ϵ-approximate policy by presenting a modified version of `SublinearRandomVI` that uses a monotonic variant of `ApxVal`, called `ApxMonVal`, that avoids this $1/(1-\gamma)^2$ loss (and the additive $1/(1-\gamma)^3$ term).

4.6 Improved Monotonic Algorithm

Here we discuss how to improve upon the running times for obtaining policies achieved in Section 4.4. We show how to obtain ϵ-approximate policies in the same time our algorithms obtained $O(\epsilon)$-approximate values, i.e. we show how to avoid the loss of $1/(1-\gamma)$ factors in running times that occurred in Section 4.4.

Our key insight to obtain this improvements is given in Lemma 4.17. This lemma shows that if we obtain values $v \in \mathbb{R}^S$ and a policy $\pi \in A^S$ such that $T_\pi(v) \geq v$ entry-wise, then $v \leq v_\pi \leq v^*$ entry-wise. In other words, if we maintain $T_\pi(v) \geq v$ and obtain values v that are ϵ-optimal then the corresponding policy π is ϵ-optimal. Consequently, to avoid loss in optimality when converting approximate values to approximate policies we simply need to ensure that $T_\pi(v) \geq v$ entrywise.

In this section we show how to maintain such values and policies by modifying our algorithms to be *monotonic*. That is we modify our routines to start with an under-estimate of the optimal values and only increase them monotonically during the course of the algorithm. This modification allows us to not only maintain a policy with the desired properties, but also to remove the extra factor of $1/(1-\gamma)^2$ in the running time of `RandomizedVI` (see Lemma 4.6).

In the remainder of this section we prove the basic properties of the value operator we use, Lemma 4.17, show how to achieve our monotonic version of `ApxVal`, which we call `ApxMonVal` (Algorithm 7) and analyze it in Lemma 4.18, Lemma 4.19, and Lemma 4.20. We conclude by giving our fastest known running times for computing ϵ-approximate policies in Theorem 4.21. This theorem analyzes the algorithm, `SublinearRandomMonVI` (see Algorithm 5), which is a monotonic version of `SublinearRandomVI`. This algorithm invokes `SampledRandomizedMonVI` (see Algorithm 8) which is a monotonic version of the algorithm, `SampledRandomizedVI`.

First we present the lemma which proves the key properties of the value this operator that we leverage to obtain higher quality policies. This lemma essentially follows from Proposition 2.2.1 in [Ber13].

Lemma 4.17. *If $v \leq \tilde{v} \leq T_{\tilde{\pi}}(v)$ entry-wise for $v, \tilde{v} \in \mathbb{R}^S$ and $\tilde{\pi} \in A^S$, then $v \leq \tilde{v} \leq v_{\tilde{\pi}} \leq v^*$ entry-wise.*

Proof. By Lemma 3.6 and the fact that $v \leq \tilde{v}$ we know that $T_{\tilde{\pi}}(v) \leq T_{\tilde{\pi}}(\tilde{v})$. Furthermore since $\tilde{v} \leq T_{\tilde{\pi}}(v)$ by assumption, we know that $\tilde{v} \leq T_{\tilde{\pi}}(\tilde{v})$. Applying $T_{\tilde{\pi}}$ preserves the inequality so we have $T_{\tilde{\pi}}(\tilde{v}) \leq T_{\tilde{\pi}}^2(\tilde{v})$ and by induction obtain that entry-wise,

$$\tilde{v} \leq T_{\tilde{\pi}}(\tilde{v}) \leq T_{\tilde{\pi}}^2(\tilde{v}) \ldots \leq T_{\tilde{\pi}}^\infty(\tilde{v}) = v_{\tilde{\pi}}$$

That $v_{\tilde{\pi}} \leq v^*$ entry-wise follows trivially from that $\tilde{\pi}$ can yield values better than the optimum policy. $\square$

Next, we present our monotonic approximate value operator, `ApxMonVal` in Algorithm 7. `ApxMonVal` modifies the output of `ApxVal` to obtain the invariants of Lemma 4.17 - provided they hold for the input. The modifications are straightforward. First we subtract a small amount from the output of `ApxVal` to obtain underestimates for the value operator with high probability. If an underestimate is higher then the current value estimate for a particular state, then we update the policy to the new action for that state and the new value, otherwise we maintain the same action for this state. This ensures that our values are always underestimates of the true values, the values are always increasing, and the invariants to invoke Lemma 4.17 are maintained. Moreover, this does not significantly change the running time or the quality of the output in terms of how well it approximates the value operator. We prove these facts in the following lemmas.

Algorithm 7 Monotonic Random Value Operator: `ApxMonVal`$(u, \pi, v_0, x, \epsilon, \delta)$

Input: Current values, $u \in \mathbb{R}^S$, current policy, $\pi \in A^S$ with $T_\pi(u) \geq u$, and initial value, $v_0 \in \mathbb{R}^S$

Input: Precomputed offsets: $x \in \mathbb{R}^{S \times A}$ with $|x_a(i) - p_a(i)^\top v_0| \leq \epsilon$ for all $i \in S$ and $a \in A$.

Input: Target accuracy $\epsilon \in (0, 1)$ and failure probability $\delta \in (0, 1)$

  1: $(q, w) := $ `ApxVal`$(u, v_0, x, \epsilon, \delta)$

  2: **for** each $i \in S$ **do**

  3:      **if** $q_i - 2\gamma\epsilon > u_i$ **then**

  4:          $\tilde{v}_i = q_i - 2\gamma\epsilon$

  5:          $\tilde{\pi}_i = w_i$

  6:      **else**

  7:          $\tilde{v}_i = u_i$

  8:          $\tilde{\pi}_i = \pi_i$

  9: **return** $(\tilde{v}, \tilde{\pi}) = $ `ApxMonVal`$(u, v_0, x, \epsilon, \delta)$

10: *(Note: $\tilde{v} \in \mathbb{R}^S$ is the result of an approximate value iteration that chooses between the best under estimated action and the previous best underestimate, and $\tilde{\pi} \in A^S$ is the corresponding policy).*

We now show that `ApxMonVal` satisfies the monotonicity properties of Lemma 4.17.

Lemma 4.18 (`ApxMonVal` Monotonicity Properties). *With probability* $1-\delta$, *an invocation of* `ApxMonVal` *(Algorithm 7) returns* $\tilde{v} \in \mathbb{R}^S$ *and* $\tilde{\pi} \in A^S$ *such that* $u \leq \tilde{v} \leq T_{\tilde{\pi}}(u) \leq T_{\tilde{\pi}}(\tilde{v})$ *entry-wise.*

Proof. By design $v \leq \tilde{v}$ entry-wise trivially. To show that $\tilde{v}_i \leq T_{\tilde{\pi}}(\tilde{v})_i$ for all $i \in S$ note that by Lemma 4.3 with probability $1 - \delta$ we have $\|q - T_w(u)\|_\infty \leq 2\gamma\epsilon$ and therefore $-2\gamma\epsilon \leq T_w(u)_i - q_i \leq 2\gamma\epsilon$ for all $i \in S$. Consequently, in the case where $q_i - 2\gamma\epsilon > u_i$, we have:

$$\tilde{v}_i = q_i - 2\gamma\epsilon \leq T_w(u)_i = T_{\tilde{\pi}}(u)_i$$

and in the case where $q_i - 2\gamma\epsilon \leq u_i$

$$\tilde{v}_i = u_i \leq T_\pi(u)_i = T_{\tilde{\pi}}(u)_i$$

by assumptions. In either case we have that $\tilde{v}_i \leq T_{\tilde{\pi}}(u)_i$. However, since $u \leq \tilde{v}$ entry-wise, by Lemma 3.6 we have $T_{\tilde{\pi}}(u)_i \leq T_{\tilde{\pi}}(\tilde{v})_i$ as desired. $\qquad\square$

Lemma 4.19 (`ApxMonVal` Approximates Value Operator). *With probability* $1-\delta$, *an invocation of* `ApxMonVal` *(Algorithm 7), returns* $\tilde{v} \in \mathbb{R}^S$ *and* $\tilde{\pi} \in A^S$ *with* $\|\tilde{v} - T_{\tilde{\pi}}(u)\|_\infty \leq 4\gamma\epsilon$ *and* $\|\tilde{v} - T(u)\|_\infty \leq 4\gamma\epsilon$.

Proof. By Lemma 4.3 with probability $1 - \delta$ we have $\|q - T_w(u)\|_\infty \leq 2\gamma\epsilon$ and $\|q - T(u)\|_\infty \leq 2\gamma\epsilon$.

Now if for any $i \in S$ we have $q_i - 2\gamma\epsilon > u_i$ then

$$|\tilde{v}_i - T_{\tilde{\pi}}(u)_i| = |q_i - 2\gamma\epsilon - T_w(u)_i| \leq \|q - T_w(u)\|_\infty + 2\gamma\epsilon = 4\gamma\epsilon \ .$$

and similarly

$$|\tilde{v}_i - T(u)_i| = |q_i - 2\gamma\epsilon - T(u)_i| \leq \|q - T(u)\|_\infty + 2\gamma\epsilon = 4\gamma\epsilon \ .$$

On the other hand, if for any $i \in S$ we have $q_i - 2\gamma\epsilon < u_i$, we note that since $\|q - T(u)\|_\infty \leq 2\gamma\epsilon$ it follows that:

$$T_\pi(u)_i \leq T(u)_i \leq q_i + 2\gamma\epsilon \leq u_i + 4\gamma\epsilon \ .$$

Since $u_i \leq T_\pi(u)_i$ by assumption, this implies that $|\tilde{v}_i - T_{\tilde{\pi}}(u)_i| \leq 4\gamma\epsilon$ and $|\tilde{v}_i - T(u)_i| \leq 4\gamma\epsilon$. $\qquad\square$

Lemma 4.20 (Runtime of `ApxMonVal`). *`ApxMonVal` (Algorithm 7) can be implemented to run in time*

$$O\left(|S||A|\left\lceil\frac{\|v-v_0\|_\infty^2}{\epsilon^2}\ln\left(\frac{|S||A|}{\delta}\right)\right\rceil\right).$$

Proof. The routine `ApxMonVal` can be implemented in the time needed to invoke `ApxVal` plus an additional $O(|S|)$ work. The running time therefore follows from the analysis of `ApxVal` in Lemma 4.3. $\square$

We now have everything we need to obtain our improved sublinear time algorithm for computing approximate policies. In the remainder of this section we provide the algorithms that achieve this and analyze them to prove Theorem 4.21.

Algorithm 8 Monotonic Sampled Randomized VI: `SampledRandomizedMonVI`$(v_0, \pi_0, T, \epsilon, \delta)$

Input: Initial values $v_0 \in \mathbb{R}^S$, initial policy, $\pi_0 \in A^S$ with $T_{\pi_0}(v_0) \geq v_0$, and number of iterations $T > 0$
Input: Target accuracy $\epsilon > 0$ and failure probability $\delta \in (0,1)$
1: Compute approximate offsets: $\tilde{x} \in \mathbb{R}^{S \times A}$ with $|\tilde{x}_a(i) - p_a(i)^\top v_0| \leq \epsilon$ for all $i \in S$ and $a \in A$
2: **for** each round $t \in [T]$ **do**
3: $\quad (v_t, \pi_t) = \texttt{ApxMonVal}(v_{t-1}, \pi_{t-1}, v_0, \tilde{x}, \epsilon/2, \delta/T)$
4: **return** (v_T, π_T)

Algorithm 9 Monotonic Sublinear Time Randomized VI: `SublinearRandomMonVI`(ϵ, δ)

Input: Target precision ϵ and failure probability $\delta \in (0,1)$
1: Let $K = \lceil \log_2(\frac{M}{\epsilon(1-\gamma)}) \rceil$ and $T = \lceil \frac{1}{1-\gamma} \log(\frac{4}{1-\gamma}) \rceil$
2: Let $v_0 = \vec{0}$, $\pi_0 \in A^S$ arbitrary, and $\epsilon_0 = \frac{M}{1-\gamma}$
3: **for** each iteration $k \in [K]$ **do**
4: $\quad \epsilon_k = \frac{1}{2}\epsilon_{k-1} = \frac{M}{2^k(1-\gamma)}$
5: $\quad (v_k, \pi_k) = \texttt{SampledRandomizedMonVI}(v_{k-1}, \pi_{k-1}, T, (1-\gamma)\epsilon_k/(4\gamma), \delta/K)$
6: **return** (v_K, π_K)

Theorem 4.21 (Sublinear Time Approximate Policy Computation). *`SublinearRandomMonVI` (see Algorithm 9) can be implemented so that with probability $1 - \delta$ it yields an ϵ-approximate policy in time*

$$\tilde{O}\left(\frac{|S||A|M^2}{(1-\gamma)^4\epsilon^2}\log\left(\frac{1}{\delta}\right)\right).$$

Proof. The algorithm `SublinearRandomMonVI` is the same as the algorithm `SublinearRandomVI`, with the exception that $\pi_k \in A^S$ are maintained and `SampledRandomizedMonVI` is used instead of `SampledRandomizedVI`. Furthermore, the algorithm `SampledRandomizedMonVI` is the same as the algorithm `SampledRandomizedVI` with the exception that $\pi_k \in A^S$ are maintained, `ApxMonVal` is used instead of `ApxVal`, and the value of ϵ used is decreased by a factor of two (to account for the slightly larger error of `ApxMonVal` in Lemma 4.19).

Furthermore, since for $v_0 = \frac{-M}{1-\gamma}\vec{1}$, any policy $\pi \in A^S$, and any $i \in S$ we have

$$[T_\pi(v_0)]_i = r_{\pi_i}(i) + \gamma \cdot p_{\pi_i}(i)^\top v_0 \geq -M + \gamma \cdot \frac{-M}{1-\gamma} = \frac{-M}{1-\gamma}$$

19

Consequently, for $v_0 \leq T_{\pi_0}(v_0)$ entry-wise for the v_0 and π_0 in `SublinearRandomMonVI` and therefore we have that $T_{\pi_k}(v_k) \geq v_k$ is maintained throughout these algorithms, with probability $1 - \delta$.

This implies that these algorithms can be analyzed exactly as they were previously (Lemma 4.12 and Lemma 4.10), though now the v_k increase monotonically throughout the algorithm and $T_{\pi_k}(v_k) \geq v_k$ is maintained. Consequently, the additive $1/(1 - \gamma)^2$ term in the previous analysis (Lemma 4.6 and Lemma 4.11) does not occur as $\|v_k - v^*\|_\infty$ decreases monotonically with k and we have that the policy returned is of the same quality as the value return with probability $1 - \delta$. $\qquad\square$

5 Finite Horizon Markov Decision Process

In this section we show how to use the ideas and techniques developed in Section 4 to obtain faster runtimes for solving finite horizon Markov Decision Processes, a close relative of the infinite horizon DMDP.

In the finite horizon Markov Decision problem, we are given a tuple (S, A, P, r, H), where S is the finite state space, A is the finite action space, P is the collection of state-action-state transition probabilities, r is the collection of state-action rewards, and $H \in \mathbb{Z}^+$ is the discrete time horizon of the problem. We use $p_a(i, j)$ to denote the probability of going to state j from state i when taking action a and define $p_a(i) \in \mathbb{R}^S$ with $p_a(i)_j \stackrel{\text{def}}{=} p_a(i, j)$ for all $j \in S$. We use $r_a(i)$ to denote the reward obtained from taking action $a \in A$ at state $i \in S$ and assume that for some known $M > 0$ it is the case that all $r_a(i) \in [-M, M]$.

The key difference between the finite horizon MDP and the infinite horizon discounted MDP is that in the former, a specified horizon H is given, and in the latter, the given discount factor indirectly adds a time dimension to the problem (i.e. a smaller discount factor, meaning that future rewards are heavily discounted, suggests a shorter relevant time horizon). Rewards in the finite horizon model are typically undiscounted, however all results easily extend to the discounted case.

As with our results for DMDP, we make the assumption throughout that for any state $i \in S$ and action $a \in A$ we can sample $j \in S$ independently at random so that $\Pr[j = k] = p_a(i, k)$ in expected $O(1)$ time. (See Section 3 for further discussion).

The goal in solving a finite horizon MDP is to compute a non-stationary policy $\pi(i, h)$, for every $i \in S$ and $h \in \{0, 1, 2, \ldots, H - 1\}$ that tells an agent which action to choose at state i at time h to maximize total expected reward over the entire time horizon H.

In the remainder of this section, we give 2 algorithms for computing ϵ-optimal policies. The first algorithm is an adaptation of our simple randomized value iteration scheme, `RandomizedVI`, and the second one is an adaptation of our high precision randomized value iteration `HighPrecisionRandomVI`. The first algorithm is sublinear for large values of ϵ. The high precision algorithm is competitive for smaller values of ϵ, in the regime where the first algorithm would effectively have superlinear running time.

5.1 Randomized Value Iteration for Finite Horizon MDP

The basic ideas in our simple randomized value iteration scheme, `RandomizedVI`, can used to generate an ϵ-optimal policy for finite horizon MDPs. We make a few tweaks in our finite horizon extension, `RandomizedFiniteHorizonVI`. The first is that we output the policy for every time step, since the optimal policy for a finite horizon MDP is non-stationary. The second is that the error ϵ-input to `RandomizedFiniteHorizonVI` is the desired cumulative error guarantee for our policy. Also, we use descending iteration counts to be consistent with backwards induction, and without loss of generality,

we start the backwards induction with an initial value vector $\vec{0}$, so the termination reward for ending at any state i is 0.

Algorithm 10 Randomized Finite Horizon VI: `RandomizedFiniteHorizonVI(v₀, H, ε, δ)`

Input: Initial values $v_0 \in \mathbb{R}^S$ and number of iterations $H > 0$
Input: Target accuracy $\epsilon \geq 0$ and failure probability $\delta \in (0,1)$
 1: Compute $x \in \mathbb{R}^{S \times A}$ such that $x_a(i) = p_a(i)^\top v_0$ for all $i \in S$ and $a \in A$
 2: Let $h = H$
 3: **while** $h > 0$ **do**
 4:     $(v_h, \pi_h) = \texttt{ApxVal}(v_{h+1}, v_0, x, \frac{\epsilon}{2H}, \delta/H)$
 5:     Decrement h by 1
 6: **return** (v_h, π_h) for each $h \in [H]$

To start the backwards induction with initial value vector $\vec{0}$, we simply use $v_0 = \vec{0}$ as the input to `RandomizedFiniteHorizonVI`.

Lemma 5.1 (Quality). `RandomizedFiniteHorizonVI`$(\vec{0}, H, \epsilon, \delta)$ *returns an ϵ-optimal policy with probability $1 - \delta$.*

Proof. Since the backwards induction of exact value iteration directly finds the optimal policy, the quality guarantee of `RandomizedFiniteHorizonVI` comes directly from the guarantees of `ApxVal`, see Lemma 4.3. At each iteration h, we allow an additive error of at most $\frac{\epsilon}{H}$ with failure probability $\frac{\delta}{H}$. The error for the non-stationary policy compounds additively in the finite horizon case, so summing up this additive error over H iterations and taking a union bound yields an ϵ-optimal policy with probability $1 - \delta$. □

Lemma 5.2 (Runtime). `RandomizedFiniteHorizonVI`$(\vec{0}, H, \epsilon, \delta)$ *runs in time*

$$\tilde{O}\left(|S||A|\frac{M^2 H^5}{\epsilon^2}\ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

Proof. Since rewards are undiscounted (or equivalently, $\gamma = 1$), the range of the values in the vector for the approximate value iteration increases by a factor of M in every iteration. More formally, using the results of Lemma 4.3, at iteration h, and letting $t = H - h$, the invocation of `ApxVal` runs in time

$$\tilde{O}\left(|S||A|\frac{t^2 M^2 H^2}{\epsilon^2}\ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

Summing over H iterations yields a total runtime of

$$\tilde{O}\left(|S||A|\frac{M^2 H^5}{\epsilon^2}\ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

□

5.2   Variance Reduced Value Iteration for Finite Horizon MDP

In this section we show how to apply variance reduction to more efficiently find an ϵ-optimal non-stationary policy for the finite horizon MDP problem by using `RandomizedFiniteHorizonVI` as a building

block. The idea is to compute $p_a^T(i)v_0$ for a fixed v_0 and then approximate $p_a^T(i)(v_k - v_0)$ to save on sample complexity. Our variance reduced algorithm `VarianceReducedFiniteHorizonVI` invokes `RandomizedFiniteHorizonVI` multiple times using different proximal v_0 over horizon H.

Algorithm 11 Variance Reduced Finite Horizon VI: `VarianceReducedFiniteHorizonVI(H, L, ε, δ)`

Input: Number of iterations $H > 0$, recompute threshold $L > 0$
Input: Target accuracy $\epsilon \geq 0$ and failure probability $\delta \in (0, 1)$
  1: $v_0 = \vec{0}$
  2: $t = [H/L]$
  3: **while** $t > 0$ **do**
  4:     $(v_{tL+0}, \pi_{tL+0}), \dots, (v_{tL+L-1}, \pi_{tL+L-1}) = $ `RandomizedFiniteHorizonVI`$(v_0, L, \frac{L\epsilon}{H}, \frac{L\delta}{H})$
  5:     $v_0 = v_{tL+0}$
  6:     Decrement t by 1
  7: **return** (v_h, π_h) for each $h \in [H]$.

Lemma 5.3 (Quality of `VarianceReducedFiniteHorizonVI`). *With probability $1 - \delta$ Algorithm 11 (`VarianceReducedFiniteHorizonVI`) produces an ϵ-optimal policy.*

Proof. This policy error follows from the guarantees of `RandomizedFiniteHorizonVI` given by Lemma 5.1. With failure probability $\frac{L\delta}{H}$, each invocation of `RandomizedFiniteHorizonVI` returns an $\frac{L\epsilon}{H}$-optimal policy for the L-horizon problem using the last value vector from the previous invocation of `RandomizedFiniteHorizonVI` as the termination value vector for the L-step subproblem. Taking union bound over the H/L iterations yields an ϵ-optimal policy with probability $1 - \delta$. □

Lemma 5.4 (Runtime of `VarianceReducedFiniteHorizonVI`). *The running time of Algorithm 11 (`VarianceReducedFiniteHorizonVI`) is*

$$\tilde{O}\left(|S|^2|A|\frac{H}{L} + |S||A|\frac{L^2 M^2 H^3}{\epsilon^2} \ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

Proof. `VarianceReducedFiniteHorizonVI` invokes `RandomizedFiniteHorizonVI` $\frac{H}{L}$ times. Each invocation of `RandomizedFiniteHorizonVI`, by Lemma 5.2 takes $O(|S|^2|A|)$ time to compute the $x \in \mathbb{R}^{S \times A}$ such that $x_a(i) = p_a(i)^\top v_0$. The advantage that this affords is that the maximum range of the sampled random variables is $[-LM, LM]$ in each invocation of `RandomizedFiniteHorizonVI`. Therefore, each iteration within `RandomizedFiniteHorizonVI`, by appealing to Lemma 4.3 takes total time

$$\tilde{O}\left(|S||A|\frac{L^2 M^2 H^2}{\epsilon^2} \ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

Aggregating over L iterations in each invocation of `RandomizedFiniteHorizonVI` yields a runtime of

$$\tilde{O}\left(|S||A|\frac{L^3 M^2 H^2}{\epsilon^2} \ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

for each invocation of `RandomizedFiniteHorizonVI`. Now, `RandomizedFiniteHorizonVI` is invoked exactly $\frac{H}{L}$ times, so that the running time of `VarianceReducedFiniteHorizonVI`, without the cost of computing the offset vectors $x_a(i)$, is:

$$\tilde{O}\left(|S||A|\frac{L^2 M^2 H^3}{\epsilon^2} \ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

22

When we account for the cost of computing the offset vectors $x_a(i)$, we conclude that the total runtime of `VarianceReducedFiniteHorizonVI` is

$$\tilde{O}\left(|S|^2|A|\frac{H}{L} + |S||A|\frac{L^2M^2H^3}{\epsilon^2}\ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

$\square$

It is important to pick a good choice for L. First, note that `VarianceReducedFiniteHorizonVI` is an improvement over `RandomizedFiniteHorizonVI` in the regime where the running time of `RandomizedFiniteHorizonVI` is super linear in $|S|^2|A|$, since the recomputations in the variance reduced version, `VarianceReducedFiniteHorizonVI`, incur a $|S|^2|A|\frac{H}{L}$ pre-processing runtime. So if ϵ is large enough that `RandomizedFiniteHorizonVI` is sublinear in $|S|^2|A|$, it is better to choose $L = 1$ (which essentially degenerates to `RandomizedFiniteHorizonVI`). When ϵ is very small, then when invoking `VarianceReducedFiniteHorizonVI`, we can pick L to equilibrate between the two terms in the runtime. Equating the two expressions yields that $L^3 = \frac{\epsilon^2|S|}{H^2M^2}$. Therefore we conclude with the following theorem.

Theorem 5.5. *`VarianceReducedFiniteHorizonVI`, Algorithm 11 can return an ϵ-optimal policy with probability $1 - \delta$ in running time*

$$\tilde{O}\left(\frac{|S|^{5/3}|A|H^{5/3}M^{2/3}}{\epsilon^{2/3}}\ln\left(\frac{|S||A|H}{\delta}\right)\right) .$$

6 Summary and Remarks

In this paper we developed a class of new value iteration-based algorithms that employ the variance reduction to achieve improved running times for solving DMDPs. These algorithms compute approximately optimal policies in nearly linear and even (sometimes) sublinear running times and improve upon the previous best known randomized algorithms for solving DMDP in terms of the dependence on $\gamma, |S|$ and $|A|$. We also extend our ideas to finite horizon MDP. For future research we hope to improve our algorithms' dependence on the discount factor. We also believe that our method can be generalized to settings when different structural knowledge is available, e.g., when there is a known upper bound on the diameter of the process, and when the process is known to be ergodic. We also plan to investigate the use of variance reduction techniques to speed up the policy iteration method and primal-dual methods.

7 Acknowledgements

References

[AMK12] Mohammad Gheshlaghi Azar, Rémi Munos, and Bert Kappen. On the sample complexity of reinforcement learning with a generative model. *arXiv preprint arXiv:1206.6461*, 2012.

[AMK13]   Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J Kappen. Minimax pac bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning*, 91(3):325–349, 2013.

[Bel57]   Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[Ber95]   Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific, Belmont, MA, 1995.

[Ber13]   Dimitri P Bertsekas. *Abstract dynamic programming*. Athena Scientific, Belmont, MA, 2013.

[BT95]   Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of the 34th IEEE Conference on Decision and Control*, volume 1, pages 560–564. IEEE, 1995.

[CW17]   Yichen Chen and Mengdi Wang. Lower bound on the computational complexity of discounted markov decision problems. *arXiv preprint arXiv:1705.07312*, 2017.

[Dan16]   George Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 2016.

[d'E63]   F d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[DG60]   Guy De Ghellinck. Les problemes de decisions sequentielles. *Cahiers du Centre dEtudes de Recherche Opérationnelle*, 2(2):161–179, 1960.

[FH14]   Eugene A Feinberg and Jefferson Huang. The value iteration algorithm is not strongly polynomial for discounted dynamic programming. *Operations Research Letters*, 42(2):130–131, 2014.

[HMZ13]   Thomas Dueholm Hansen, Peter Bro Miltersen, and Uri Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *J. ACM*, 60(1):1:1–1:16, February 2013.

[Hoe63]   Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[How60]   Ronald A. Howard. *Dynamic programming and Markov processes*. The MIT press, Cambridge, MA, 1960.

[JZ13]   Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 315–323, 2013.

[KMN02]   Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[KS99]   Michael J Kearns and Satinder P Singh. Finite-sample convergence rates for q-learning and indirect algorithms. In *Advances in neural information processing systems*, pages 996–1002, 1999.

[LDK95]   Michael L Littman, Thomas L Dean, and Leslie Pack Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc., 1995.

[LH12]   Tor Lattimore and Marcus Hutter. Pac bounds for discounted mdps. In *International Conference on Algorithmic Learning Theory*, pages 320–334. Springer, 2012.

[LS14]   Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 424–433. IEEE, 2014.

[LS15a]   Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 230–249. IEEE, 2015.

[LS15b]   Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. *CoRR*, abs/1503.01752, 2015.

[MS99]   Yishay Mansour and Satinder Singh. On the complexity of policy iteration. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 401–408. Morgan Kaufmann Publishers Inc., 1999.

[Put14]   Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Sch13]   Bruno Scherrer. Improved and generalized upper bounds on the complexity of policy iteration. In *Advances in Neural Information Processing Systems*, pages 386–394, 2013.

[SLL09]   Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444, 2009.

[SLW+06] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888. ACM, 2006.

[Tse90]   Paul Tseng. Solving h-horizon, stationary markov decision problems in time proportional to log (h). *Operations Research Letters*, 9(5):287–297, 1990.

[Wan17]   Mengdi Wang. Randomized linear programming solves the discounted markov decision problem in nearly-linear running time. *arXiv preprint arXiv:1704.01869*, 2017.

[Ye05]   Yinyu Ye. A new complexity result on solving the markov decision problem. *Mathematics of Operations Research*, 30(3):733–749, 2005.

[Ye11]   Yinyu Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36(4):593–603, 2011.

A Value Iteration Facts

Here we review some basic facts about value iteration we use in our analysis. These are well established in the MDP literature but we include it here for completeness.

Lemma 3.4 (Contraction Mapping). *For all values $u, v \in \mathbb{R}^S$ we have that $\|T(u) - T(v)\|_\infty \leq \gamma \|u - v\|_\infty$ and consequently $\|T(u) - v^*\|_\infty \leq \gamma \|u - v^*\|_\infty$, where v^* is the optimal value vector.*

Please see [Ber13] and [Put14] for more details on basic facts for DMDP.

Lemma 3.5. *For any policy $\pi \in A^S$ and values $u \in \mathbb{R}^S$ it holds that $\|T_\pi(u) - v_\pi\|_\infty \leq \frac{\gamma}{1-\gamma}\|T_\pi(u) - u\|_\infty$, where v_π is the exact value vector for policy π.*

Proof. By the fact that $T_\pi(v_\pi) = v_\pi$ we have:

$$
\begin{aligned}
\|T_\pi(u) - v_\pi\|_\infty &= \|T_\pi(u) - T_\pi(v_\pi)\|_\infty \\
&= \|T_\pi(u) - T_\pi(T_\pi(u)) + T_\pi(T_\pi(u)) - T_\pi(v_\pi)\|_\infty \\
&\leq \|T_\pi(u) - T_\pi(T_\pi(u))\|_\infty + \|T_\pi(T_\pi(u)) - T_\pi(v_\pi)\|_\infty \\
&\leq \gamma \|T_\pi(u) - u\|_\infty + \gamma \|T_\pi(u) - v_\pi\|_\infty \ .
\end{aligned}
$$

Therefore, $(1 - \gamma)\|T_\pi(u) - v_\pi\|_\infty \leq \gamma\|T_\pi(u) - u\|_\infty$ and so we conclude that

$$
\|T_\pi(u) - v_\pi\|_\infty \leq \frac{\gamma}{1 - \gamma}\|T_\pi(u) - u\|_\infty \ .
$$

$\square$

Lemma 3.6. *If values $u, v \in \mathbb{R}^S$ satisfy $u \leq v$ entry-wise, then $T(u) \leq T(v)$ entry-wise.*

Proof. Note that

$$
T(u)_i = \max_{a \in A}\left[r_a(i) + \gamma \cdot p_a(i)^\top u \right] \leq \max_{a \in A}\left[r_a(i) + \gamma \cdot p_a(i)^\top v \right] = T(v)_i \ ,
$$

where the inequality holds because $u \leq v$, and γ and $p_a(i)$ are non-negative for all $a \in A, i \in S$. $\square$

B Solving DMDPs With Interior Point Methods

In this section, we show how to solve DMDPs using standard linear programming machinery, i.e. interior point methods. First, we provide a fairly standard formulation of a DMDP as a linear program and show that from an approximate solutions to the linear program an approximately optimal policy can be obtained. (See Definition B.1 and Lemma B.2). While there is some loss in quality in converting an approximate DMDP LP solutions into policies, we show how to bound the loss so that it only only affects logarithmic factors in the final running time.

We then show how to use interior point methods to solve this linear program. To make it easier to apply the interior point methods we reduce approximately solving this linear program to approximately solving ℓ_1-regression, using a technique from [LS15b]. The fastest known interior point methods for solving ℓ_1 regression, i.e. [LS15b], gives the running time for solving DMDPS that we provided in Section 2 (See Theorem B.8)

We start by providing and analyzing our linear program formulation of the DMDP.

Definition B.1 (DMDP Linear Program). We call the following linear program the *DMDP LP*

$$
\begin{aligned}
\text{minimize} \quad & v^\top \vec{1} \\
\text{subject to} \quad & \mathbf{A}v \geq r
\end{aligned}
\tag{B.1}
$$

where $r \in \mathbb{R}^{(S \times A)}$ is the vector of rewards, i.e. $r_{i,a} = r_a(i)$ for all $i \in S$ and $a \in A$ and $\mathbf{A} = \mathbf{E} - \gamma \mathbf{P}$ where $\mathbf{E} \in \mathbb{R}^{(S \times A) \times S}$ is the matrix where for all $i, j \in S$ and $a \in A$ we have that the j-th entry of row (i, a) of $\mathbf{E}$ is 1 if $i = j$ and 0 otherwise, and $\mathbf{P} \in \mathbb{R}^{(S \times A) \times S}$ is a matrix where for all $i \in S$ and $a \in A$ we have that row (i, a) of $\mathbf{P}$ is $p_a(i)$.

We call a vector $v \in \mathbb{R}^S$ an ϵ-*approximate DMDP LP solution* if $\mathbf{A}v \geq r - \epsilon\vec{1}$ and $v^\top\vec{1} \leq OPT + \epsilon$ where OPT is the optimal value of the DMDP LP, (B.1).

This DMDP LP is a standard formulation of the DMDP as a linear program. In the next lemma we prove that solving the DMDP LP is equivalent to solving a DMDP and we show that approximately optimizing the objective of the DMDP LP while meeting the constraints corresponds to values that are approximately optimal in terms of local optimality of the value operator.

Lemma B.2 (DMDP LP Properties). *In the DMDP LP a vector $v \in \mathbb{R}^S$ satisfies $\mathbf{A}v \geq r$ if and only if $v \geq T(v)$ entry-wise. If additionally v satisfies $v^\top\vec{1} \leq OPT + \epsilon$ then $T(v) \leq v \leq T(v) + \epsilon\vec{1}$. Consequently, v^* the solution to the DMDP is the unique minimizer of the DMDP LP.*

Proof. Note that $\mathbf{A}v \geq r$ is equivalent to $v_i - \gamma \cdot p(i, a)^\top v \geq r_i(a)$ or $v_i \geq r_i(a) + \gamma \cdot p(i, a)^\top v$ for all $i \in S$ and $a \in A$. Since $[T(v)]_i = \max_{a \in A}[r_i(a) + \gamma \cdot p(i, a)^\top v]$ we have $\mathbf{A}v \geq r$ if and only if $T(v) \leq v$.

Now suppose v additionally satisfies $v^\top\vec{1} \leq OPT + \epsilon$. Let $i \in [n]$ be a coordinate such that $[T(v) - v]_i$ is maximal. Let $\alpha = [v - T(v)]_i$ and let w be the same vector as v where coordinate i is decreased by α. Since $v \geq T(v)$ we know α is positive and see that $T(w) \geq w$ as well by our choice of α (since T is monotonic in v). Consequently $\mathbf{A}w \geq r$ and $w^\top\vec{1} = v^\top\vec{1} - \alpha \leq OPT + \epsilon - \alpha$. By the definition of OPT we therefore have that $\alpha \leq \epsilon$ and $v \leq T(v) + \epsilon\vec{1}$.

Consequently, any solution v_{lp} to the DMDP LP must satisfy $v_{lp} = T(v_{lp})$ and therefore be the unique optimal values of the DMDP. $\square$

Using Lemma B.2 we show that given any ϵ-approximate solution to the DMDP LP we can quickly compute an $O(\epsilon|S|(1 - \gamma)^{-2})$-approximate policy simply by computing the best actions with respect to these approximate values.

Lemma B.3 (Approximate Policies from Approximate LP Solution). *If v is an ϵ-approximate DMDP LP solution and if $\pi \in A^S$ is defined with $\pi_i = \operatorname{argmax}_{a \in A} r_a + \gamma \cdot p(i, a)^\top v$ for all $i \in S$ then π is an $8\epsilon|S|(1 - \gamma)^{-2}$-optimal policy.*

Proof. Note that $\mathbf{A}\vec{1} = (1 - \gamma)\vec{1}$ and therefore $w \overset{\text{def}}{=} v + \frac{\epsilon}{1-\gamma}\vec{1}$ satisfies $\mathbf{A}w \geq r$. Since

$$
w^\top\vec{1} = v^\top\vec{1} + \frac{\epsilon|S|}{(1 - \gamma)} \leq OPT + \frac{2\epsilon|S|}{(1 - \gamma)}
$$

Lemma B.2 yields $T(w) \leq w \leq T(w) + 2C\vec{1}$ for $C \overset{\text{def}}{=} \epsilon|S|(1-\gamma)^{-1}$ and $\|w - T(w)\|_\infty \leq 2C$. Since clearly $\|v - w\|_\infty \leq C$ and $\|T(v) - T(w)\|_\infty \leq C$ then by triangle inequality we have that $\|v - T(v)\|_\infty \leq 4C$.

Next, triangle inequality yields that $\|v - v^*\| \leq \|T(v) - v\| + \|T(v) - v^*\|$ and since $T(v^*) = v^*$ by the contraction property we have $\|T(v) - v^*\|_\infty \leq \gamma\|v - v^*\|_\infty$ and combining these facts yields that

$\|v - v^*\| \leq 4C\gamma(1-\gamma)^{-1}$. However, clearly by the definition of π it is the case that $T(v) = T_\pi(v)$ and by Lemma 3.5 we have $\|T(v) - v_\pi\| \leq \gamma(1-\gamma)^{-1} \cdot 4C$. Consequently,

$$\|v_\pi - v^*\|_\infty \leq \|T(v) - v_\pi\|_\infty + \|T(v) - v^*\|_\infty \leq 8C\gamma(1-\gamma)^{-1}$$

where we used that $\|T(v) - v^*\|_\infty \leq \gamma\|v - v^*\|_\infty \leq 4C\gamma(1-\gamma)^{-1}$. $\qquad\square$

With Lemma B.3 established, the goal in the rest of this section is to show how to produce an approximate DMDP LP solution quickly. While it may be possible to adapt the fastest generic linear programming algorithms (e.g. [LS15a]) directly to the DMDP LP formulation, for a simpler proof we reduce the original linear program to computing ϵ-approximate ℓ_1 regression. We do this so that we can easily invoke the following result of [LS15a] to obtain our running times.

Corollary B.4 (ℓ_1 Regression Running Time - Corollary 24 of [LS15a], Adapted)**.** *Given $\boldsymbol{M} \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$ then there is an algorithm which with high probability in time $\tilde{O}(nd^{1.5}\log(1/\epsilon))$ finds $x \in \mathbb{R}^d$ with*

$$\|\boldsymbol{M}x - b\|_1 \leq \min_{y \in \mathbb{R}^d} \|\boldsymbol{M}y - b\|_1 + \epsilon\|b\|_1 \ .$$

To convert our DMDP LP to such an optimization problem we turn our constraint set into a symmetric one. First we modify the DMDP LP to have two sided constraints. We know that v^* satisfies $\|v^*\|_\infty \leq \frac{M}{1-\gamma}$ and therefore for all $i \in S$ and $a \in A$ we have

$$[\mathbf{A}v^*]_{(i,a)} = r_{(i,a)} + \gamma \cdot p_{(i,a)}^\top v^* \leq M + \gamma \cdot \frac{M}{1-\gamma} \leq \frac{2M}{1-\gamma} \ .$$

Furthermore, since we know that v^* is the optimizer of the DMDP LP by Lemma B.2 we therefore see that solving the DMDP LP is equivalent to minimizing $v^\top \vec{1}$ under the constraint that $v \in P$ where

$$P = \left\{ v \in \mathbb{R}^S : r \leq \mathbf{A}v \leq \frac{2M}{1-\gamma}\vec{1} \right\} \ .$$

Moreover, we can center this feasible region; for $s = \frac{1}{2}(\frac{2M}{1-\gamma}\vec{1} - r)$ and $b = \frac{1}{2}(\frac{2M}{1-\gamma}\vec{1} + r)$ we have

$$P = \{ v \in \mathbb{R}^S : -s \leq \mathbf{A}v - b \leq s \} \ .$$

Furthermore, if we let $\mathbf{S} = \mathrm{diag}(s)$, i.e. the diagonal matrix with s on the diagonal then

$$P = \left\{ v \in \mathbb{R}^S : \|\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b\|_\infty \leq 1 \right\}$$

Consequently, we wish to minimize $v^\top \vec{1}$ under the constraint that $\|\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b\|_\infty \leq 1$. To turn this into a ℓ_1 regression problem we use a technique from [LS15a] that ℓ_∞ constraints can be turned into ℓ_1 objectives through the following simple lemma.

Lemma B.5. *For all vectors $x \in R$ we have*

$$|x-1| + |x+1| = 2 \cdot \max\{|x|, 1\}$$

Proof. If $x \in [0,1)$ then

$$|x-1| + |x+1| = 1 - x + 1 + x = 1$$

and if $x \notin [1, \infty)$ then

$$|x-1| + |x+1| = |x| - 1 + |x| + 1 = 2|x|$$

The result follows by symmetry for the case when $x < 0$. $\qquad\square$

From this lemma we see that to penalize infeasibility, ie v such that $\|\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b\|_\infty > 1$, we can simply add a term of the form $\|\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b - \vec{1}\|_1 + \|\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b + \vec{1}\|_1$ to the objective. We give the following ℓ_1 regression problem:

Definition B.6 (DMDP ℓ_1 Regression). For a given DMDP and a parameter α (to be set later) we call the following ℓ_1 regression problem the *DMDP ℓ_1 problem*[7]

$$\min_v f(v) = \left| \alpha \left(\frac{|S|M}{1-\gamma} + \vec{1}^\top v \right) \right| + \left\| \mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b - \vec{1} \right\|_1 + \left\| \mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b + \vec{1} \right\|_1 \tag{B.2}$$

$$= \left| \alpha \left(\frac{|S|M}{1-\gamma} - \vec{1}^\top v \right) \right| + 2 \sum_{i \in S} \max \left\{ |[\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b]_i|, 1 \right\} \tag{B.3}$$

where $s = \frac{1}{2}(\frac{2M}{1-\gamma}\vec{1} - r)$ and $b = \frac{1}{2}(\frac{2M}{1-\gamma}\vec{1} + r)$, and $\mathbf{S} = \mathrm{diag}(s)$. We let v_f^* denote the optimal solution to this ℓ_1 regression problem and we call v an ϵ-optimal solution to f if $f(v) \leq f(v_f^*) + \epsilon$.

Note that our ϵ-approximate solution to (B.2) may be infeasible for the original linear program, (B.1), and $2 \sum_i \max \left\{ |[\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b]_i|, 1 \right\}$ characterizes the magnitude of our constraint violation for the original MDP problem, (B.1). However, by carefully choosing α we can show that we can trade-off violating these constraints with lack of optimality in the objective and therefore ϵ-approximate solutions to (B.2) still give approximate DMDP-LP solutions. We quanitfy this below.

Lemma B.7. *Suppose that v is an ϵ-approximate solution to the DMDP ℓ_1 problem then v is an ϵ'-approximate DMDP LP solution for*

$$\epsilon' \leq \max \left\{ \frac{\epsilon}{\alpha}, \frac{2\alpha|S|M^2}{(1-\gamma)^2} + \frac{\epsilon M}{(1-\gamma)} \right\} .$$

Proof. Recall that $|[\mathbf{S}^{-1}\mathbf{A}v^* - \mathbf{S}^{-1}b]_i| \leq 1$ for all $i \in S$ and $\vec{1}^\top v^* \in [\frac{-|S|M}{1-\gamma}, \frac{|S|M}{1-\gamma}]$. Consequently,

$$f(v_f^*) \leq f(v^*) = \alpha \left(\frac{|S|M}{1-\gamma} + \vec{1}^\top v^* \right) + 2|S| \leq \frac{2\alpha|S|M}{1-\gamma} + 2|S|$$

Since, $f(v) \leq f(v_f^*) + \epsilon$ by assumption we have that

$$\alpha \left(\frac{|S|M}{1-\gamma} + \vec{1}^\top v \right) + 2|S| \leq f(v) \leq \alpha \left(\frac{|S|M}{1-\gamma} + \vec{1}^\top v^* \right) + 2|S| + \epsilon$$

and therefore $\vec{1}^\top v \leq \vec{1}^\top v^* + \frac{\epsilon}{\alpha}$. Furthermore this implies,

$$2 \sum_{i \in S} \max \left\{ |[\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b]_i|, 1 \right\} \leq f(v) \leq \frac{2\alpha|S|M}{1-\gamma} + 2|S| + \epsilon$$

Now letting $\delta_1 = \epsilon + \frac{2\alpha|S|M}{1-\gamma}$ we see this in turn this implies that $\|\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b\|_\infty \leq \frac{\delta_1}{2} + 1$. Consequently, $\mathbf{S}^{-1}\mathbf{A}v - \mathbf{S}^{-1}b \geq -(\frac{\delta_1}{2} + 1)\vec{1}$ and

$$\mathbf{A}v \geq b - s - \frac{\delta_1}{2}s \geq r - \frac{\delta_1}{4}r - \frac{\delta_1}{2} \left(\frac{M}{1-\gamma}\vec{1} \right) \geq r - \frac{\delta_1 M}{(1-\gamma)}\vec{1} .$$

Considering the maximum of $\frac{\epsilon}{\alpha}$ and $\frac{\delta_1 M}{1-\gamma}$ yields the result. □

---

[7]This equality below is an immediate consequence of Lemma B.5

We now have all the pieces we need to prove our desired result.

Theorem B.8 (Interior Point Based DMDP Running Time). *Given a DMDP we can use [LS15a] to compute an ϵ-approximate policy with high probability in time*

$$\tilde{O}\left(|S|^{2.5}|A|\log\left(\frac{M}{(1-\gamma)\epsilon}\right)\right)$$

Proof. The result follows from using Corollary B.4 to solve the DMDP ℓ_1 problem. By Lemma B.3 it suffices to have an $\frac{\epsilon(1-\gamma)^2}{8|S|}$ approximate solution to the ℓ_1 problem. Therefore, we choose $\alpha = \frac{\epsilon(1-\gamma)^4}{32M^2|S|^2}$ and $\epsilon' = \max\{\frac{\alpha\epsilon(1-\gamma)^2}{8|S|}, \frac{\epsilon(1-\gamma)^3}{16M|S|}\}$ and use Corollary B.4 to compute an ϵ'-optimal solution to the DMDP ℓ_1 problem. This can be used to compute an ϵ-approximate policy through Lemma B.3. The running time follows from Corollary B.4, as $\mathbf{M} = \mathbf{S}^{-1}\mathbf{A} \in \mathbb{R}^{(S\times A)\times S}$ and we solve to error $O(\frac{\epsilon'(1-\gamma)}{|S|M})$ as for b in the corollary $\|b\|_1 = O(|S|M/(1-\gamma))$. □