

PAC Reinforcement Learning Bounds for RTDP and Rand-RTDP

Alexander L. Strehl
Computer Science Dept.
Rutgers University
Piscataway, NJ 08854 USA
strehl@cs.rutgers.edu

Lihong Li
Computer Science Dept.
Rutgers University
Piscataway, NJ 08854 USA
lihong@cs.rutgers.edu

Michael L. Littman
Computer Science Dept.
Rutgers University
Piscataway, NJ 08854 USA
mlittman@cs.rutgers.edu

Abstract

Real-time Dynamic Programming (RTDP) is a popular algorithm for planning in a Markov Decision Process (MDP). It can also be viewed as a learning algorithm, where the agent improves the value function and policy while acting in an MDP. It has been empirically observed that an RTDP agent generally performs well when viewed this way, but past theoretical results have been limited to asymptotic convergence proofs. We show that, like true learning algorithms E^3 and R-MAX, a slightly modified version of RTDP satisfies a Probably Approximately Correct (PAC) condition (with better sample complexity bounds). In other words, we show that the number of timesteps in an infinite-length run in which the RTDP agent acts according to a non- ϵ -optimal policy from its current state is less than some polynomial (in the size of the MDP), with high probability. We also show that a randomized version of RTDP is PAC with asymptotically equal sample complexity bounds, but has much less per-step computational cost, $O(\ln(S) \ln(SA) + \ln(A))$, rather than $O(S + \ln(A))$ when we consider only the dependence on S , the number of states and A , the number of actions of the MDP.

Introduction

The Real-Time Dynamic Programming (RTDP) algorithm was introduced by Barto et al. (1995). In that work, it was shown that RTDP generalizes a popular search algorithm, Learning Real Time A* (LRTA*) (Korf, 1990). Thus, RTDP is important to the reinforcement learning (RL) community as well as the heuristic search community. New analyses and variations of RTDP, such as Labeled RTDP and Bounded RTDP, continue to be studied (Bonet & Geffner, 2003; McMahan et al., 2005).

With respect to standard RL terminology, RTDP is, by definition, strictly a “planning” algorithm because it receives as input the MDP (rewards and transitions) it will act upon. A “learning” algorithm, as typically defined, doesn’t receive these quantities as input. However, as noted by Barto et al. (1995), RTDP can be thought of as a learning algorithm. The form of RTDP is similar to many learning algorithms in that an agent following RTDP travels through the

MDP in one long trajectory and updates its action-values (Q -value estimates) for those state-actions that it actually visits. It also displays complex behavior that could arguably be considered “learning”¹. Traditional planning algorithms, such as value iteration, tend to solve an MDP by iterating through all the state-action pairs many times, rather than by following a valid trajectory through the MDP. Hence, RTDP can be considered something in between a learning algorithm and a planning algorithm, and it represents an important connection between the learning and planning communities in RL.

The first result of our paper is that RTDP has very nice theoretical properties when analyzed in a PAC-like framework (as defined in a later section). We prove that a slightly modified version of RTDP, when run forever in any MDP, will make only a polynomial number of mistakes (non- ϵ -optimal choices) with high probability. We formalize this notion of *sample complexity* below. This result doesn’t imply convergence of its action-value estimates².

To the best of our knowledge, the only related theoretical result is that the action-value estimates, $Q(s, a)$, computed by RTDP converge to $Q^*(s, a)$ in the limit (Barto et al., 1995; Bertsekas & Tsitsiklis, 1996). Instead, we would like a guarantee on RTDP’s performance after only finitely many interactions with the environment. Furthermore, we are not as interested in the convergence of RTDP’s action-value estimates as we are in the performance of its behavior (the value of the agent’s policy). For instance, in the work of Bonet and Geffner (2003), it is empirically shown that RTDP’s action-value estimates may take a long time to converge to their optimal values. This finding is due to the fact that some states may be difficult to reach and RTDP only updates the value estimates for states as they are reached by the agent. The authors create a new algorithm, Labeled RTDP, that exhibits a better convergence rate (empirically). However, when the actual performances of the algorithms were compared as measured by average cost to the goal (Figure 3 of Bonet and Geffner (2003)), RTDP’s performance was at least as good as Labeled RTDP’s. This observation suggests

¹See Section 3.4 of Barto et al. (1995) for a related discussion.

²In fact, the action-value estimates of our modified version of RTDP won’t converge to the optimal action-values Q^* . We leave it as an important open problem whether the original RTDP algorithm is PAC-MDP or not.

that the slow convergence rate of RTDP may not negatively impact its resulting behavior. Therefore, we have chosen to directly analyze the sample complexity of RTDP rather than the convergence rate of its action-value estimates.

One of RTDP’s attractive features is that by only updating value estimates for the states reached by an agent in the MDP, it focuses computation on relevant states. In fact, each step (action choice) of the agent requires only a single Bellman backup, a computation that, in the worst-case, scales linearly with the number of states. Although this method is a significant improvement over more expensive operations such as value iteration, it can still be costly in large environments. Our second result is to provide an algorithm, called *Rand-RTDP* that has similar PAC-like learning guarantees, but whose computation per-step scales at a rate of $O(\ln^2(S))$, when we consider its dependence on S . The key insight is that the single full-update of RTDP can be closely approximated by an average of a few randomly chosen sample updates. This idea has been mentioned in Bertsekas and Tsitsiklis (1996) and also analyzed in Kearns et al. (1999).

Definitions and Notation

This section introduces the Markov Decision Process notation used throughout the paper; see Sutton and Barto (1998) for an introduction. An MDP M is a five tuple $\langle S, A, T, R, \gamma \rangle$, where S is the state space, A is the action space, $T : S \times A \times S \rightarrow [0, 1]$ is a transition function, $R : S \times A \rightarrow [0, 1]$ is a reward function, and $0 \leq \gamma < 1$ is a discount factor on the summed sequence of rewards. We also let S and A denote the number of states and the number of actions, respectively. From state s under action a , the agent receives a random reward r , which has expectation $R(s, a)$, and is transported to state s' with probability $T(s'|s, a)$. A policy is a strategy for choosing actions. We assume (unless otherwise noted) that rewards all lie between 0 and 1. A stationary policy is one that produces an action based on only the current state. For any policy π , let $V_M^\pi(s)$ ($Q_M^\pi(s, a)$) denote the discounted, infinite-horizon value (action-value) function for π in M (which may be omitted from the notation) from state s . If T is a positive integer, let $V_M^\pi(s, T)$ denote the T -step value function of policy π . Specifically, $V_M^\pi(s) = E[\sum_{j=1}^{\infty} \gamma^{j-1} r_j]$ and $V_M^\pi(s, T) = E[\sum_{j=1}^T \gamma^{j-1} r_j]$ where $[r_1, r_2, \dots]$ is the reward sequence generated by following policy π from state s . These expectations are taken over all possible paths the agent might follow. The optimal policy is denoted π^* and has value functions $V_M^*(s)$ and $Q_M^*(s, a)$. Note that a policy cannot have a value greater than $1/(1 - \gamma)$.

PAC-MDP

In this section, we provide the formal definition we use to characterize a learning algorithm as *efficient*.

Any algorithm \mathcal{A} that guides an agent to act based on only its history (trajectory through the MDP), such as Q-learning (Watkins & Dayan, 1992) and R-MAX (Brafman & Tennenholtz, 2002), can be viewed as simply some non-stationary policy. Thus, each action the agent takes is the result of

executing some policy. Suppose we stop the algorithm after its $(t - 1)$ st action, and let us denote the agent’s policy by \mathcal{A}_t . This policy has a well-defined value function $V^{\mathcal{A}_t}(s_t)$, where s_t is the current (t th) state reached by the agent. We say it is ϵ -optimal, for any ϵ , from its current state, if $V^*(s_t) - V^{\mathcal{A}_t}(s_t) \leq \epsilon$.

Now, suppose that the algorithm \mathcal{A} is run for an infinite-length trajectory (run forever in the MDP, starting from an arbitrary initial state). Kakade (2003) defines the **sample complexity of exploration** (sample complexity, for short) of \mathcal{A} to be the number of timesteps t such that the non-stationary policy at time t , \mathcal{A}_t , is not ϵ -optimal from the current state, s_t , at time t (formally $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$). We believe this definition captures the essence of measuring learning. An algorithm \mathcal{A} is then said to be an **efficient PAC-MDP** (Probably Approximately Correct in Markov Decision Processes) algorithm if, for any ϵ and δ , the per-step computational complexity and the sample complexity of \mathcal{A} is less than some polynomial in the relevant quantities $(S, A, 1/\epsilon, 1/\delta, 1/(1 - \gamma))$, with probability at least $1 - \delta$. It is simply **PAC-MDP** if we relax the definition to have no computational complexity requirement. The terminology, PAC, is borrowed from Valiant (1984), a classic paper dealing with classification.

Two Algorithms

In this section, we present two learning/planning algorithms. Both require as input a model, including the transition and reward functions, of an MDP. In addition, we assume that the learner receives S , A , and γ as input. The following experiment is then conducted. The agent always occupies a single state s of the MDP M . The algorithm is told this state and must compute an action a . The agent receives a reward r (which our algorithms completely ignore) and is then transported to another state s' according to the reward and transition functions. This procedure then repeats forever. The first state occupied by the agent may be chosen arbitrarily. We define a *timestep* to be a single interaction with the environment, as described above.

Since we are interested in algorithms that satisfy the PAC-MDP guarantee, we also allow the learning algorithm to receive two additional inputs, ϵ and δ , both real numbers between zero and one, exclusively.

Both algorithms maintain action-value estimates, $Q(s, a)$ for each state-action pair (s, a) . At time $t = 1, 2, \dots$, let $Q_t(s, a)$ denote the algorithm’s current action-value estimate for (s, a) and let $V_t(s)$ denote $\max_{a \in A} Q_t(s, a)$. The learner always acts greedily with respect to its estimates, meaning that if s_t is the t th state reached, $a' := \operatorname{argmax}_{a \in A} Q_t(s_t, a)$ is the next action chosen. Additionally, both algorithms make use of “optimistic initialization”, that is, $Q_1(s, a) = 1/(1 - \gamma)$ for all (s, a) . The main difference between the two algorithms is how the action-values are updated on each timestep.

The RTDP Algorithm

Real-time dynamic programming (RTDP) is an algorithm for acting in a known MDP that was first studied by Barto et al. (1995).

Suppose that at time $t \geq 1$, action a is performed from state s . Then, the following update is performed:

$$Q_{t+1}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V_t(s'). \quad (1)$$

We analyze a modified version of RTDP, which has an additional free parameter, $\epsilon_1 \in (0, 1)$. In the analysis section (Proposition 2), we will provide a precise value for ϵ_1 in terms of the other inputs (S, A, ϵ, δ , and γ) that guarantees the resulting algorithm is PAC-MDP. Our modification is that we allow the update, Equation 1, to take place only if the new action-value results in a decrease of at least ϵ_1 . In other words, the following equation must be satisfied for an update to occur:

$$Q_t(s, a) - \left(R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V_t(s') \right) \geq \epsilon_1. \quad (2)$$

Otherwise, no change is made and $Q_{t+1}(s, a) = Q_t(s, a)$. For all other state-action pairs $(s', a') \neq (s, a)$, the action-value estimate is left unchanged, that is, $Q_{t+1}(s', a') = Q_t(s', a')$.

The only difference between our modified version of RTDP and the standard version is that we enforce an update condition (Equation 2). The main purpose of this change is to bound the number of updates performed by RTDP, which is required for our analysis that RTDP is PAC-MDP. It also has the interesting side effect of reducing the overall computational cost of running RTDP. Without the update condition, RTDP will continually refine its value-function estimates, which involves infinitely many Bellman backups (computations of Equation 1). With the update condition, RTDP stops updating its estimates once they are “close” to optimal. By “close” we mean sufficient to obtain an ϵ -optimal policy. Hence, the modified version will perform only a finite number of Bellman backups. A similar modification is described by Bonet and Geffner (2003).

The Rand-RTDP Algorithm

We now introduce a randomized version of RTDP, called Rand-RTDP. In addition to action-value estimates, Rand-RTDP maintains a quantity $\text{LAU}(s, a)$, for each (s, a) , which stands for “last attempted update”. Let $\text{LAU}_t(s, a)$ denote the value of $\text{LAU}(s, a)$ at time t (meaning immediately before the t th action is chosen). This quantity stores the timestep when (s, a) was last encountered and an “attempted update” occurred (as defined below). The algorithm also has two free parameters, $\epsilon_1 \in (0, 1)$ and a positive integer m . In the analysis section (Proposition 3) we provide precise values for these parameters in terms of the other inputs (S, A, ϵ, δ , and γ) that guarantee the resulting algorithm is PAC-MDP.

Suppose that at time $t \geq 1$, action a is performed from state s , and $\text{LAU}_t(s, a)$ is less than or equal to the last timestep for which any action-value estimate was updated (meaning some action-value estimate has changed on or since the last attempted update of (s, a)). We call such a timestep an *attempted update*. Consider the following up-

date:

$$Q_{t+1}(s, a) = \frac{1}{m} \sum_{i=1}^m (r_i + \gamma V_t(s_i)) + \epsilon_1, \quad (3)$$

where s_1, \dots, s_m are m random draws from $T(\cdot|s, a)$, the transition distribution for (s, a) and r_1, \dots, r_m are m random draws from the reward distribution for (s, a) (with mean $R(s, a)$). We allow the update, Equation 3, to take place only if the new action-value results in a decrease of at least ϵ_1 . In other words, the following equation must be satisfied for a successful update to occur:

$$Q_t(s, a) - \frac{1}{m} \sum_{i=1}^m (r_i + \gamma V_t(s_i)) \geq 2\epsilon_1. \quad (4)$$

Otherwise, no change is made, an *unsuccessful update* occurs, and $Q_{t+1}(s, a) = Q_t(s, a)$. For all other state-action pairs $(s', a') \neq (s, a)$, the action-value estimate is left unchanged, that is, $Q_{t+1}(s', a') = Q_t(s', a')$. In addition, if the condition on $\text{LAU}_t(s, a)$ above doesn’t hold, then no update is made.

The main difference between Rand-RTDP and RTDP is that instead of a full Bellman backup (Equation 1), Rand-RTDP performs a multi-sample backup (Equation 3). In addition, the update of Rand-RTDP involves an additive bonus of ϵ_1 (a small positive number). The purpose of this bonus is to ensure that, with high probability, the action-value estimates, $Q(s, a)$, are optimistic, which means that they always upper bound the optimal action-values, $Q^*(s, a)$. The bonus is needed because even if the previous value estimates, $V_t(\cdot)$, are optimistic, the random sampling procedure could result in a choice of next-states and rewards so that the average $(\frac{1}{m} \sum_{i=1}^m (r_i + \gamma V_t(s_i)))$ of these terms is not optimistic anymore.

The purpose of the $\text{LAU}(s, a)$ values (and the condition on them required for an attempted update) is to limit the number of attempted updates performed by the algorithm. This limit allows us to prove that, with high probability, every randomized update will result in a new action-value estimate that is not too far from what would have resulted if a full backup was performed. They also limit the total amount of computation. Specifically, they prevent an attempted update for (s, a) if the last attempted update for (s, a) was unsuccessful and no other action-value estimates have changed since the last attempted update of (s, a) .

Rand-RTDP’s Generative Model Note that Rand-RTDP updates its action-value estimates by obtaining random samples of next states s' and immediate rewards r for its current state-action pair. This sampling is the only time Rand-RTDP uses its knowledge of M . In other words, Rand-RTDP requires only a *generative model* of the underlying MDP³, rather than an actual representation of the transition and reward functions. Next, we show that the agent can “simulate”

³A generative model of an MDP is a model that outputs a random next state and immediate reward pair given input of a state-action pair (s, a) . Each call to the model should produce independent draws distributed according to the dynamics of the MDP (Kearns et al., 1999).

a generative model when it is given the true dynamics of the underlying MDP. The reverse, obtaining the true dynamics given a generative model, is in general impossible given only finite time.

If the algorithm is given an explicit representation of the transition function, $T(\cdot|s, a)$, in standard form (a list, for each (s, a) , of possible next-states s' and their corresponding probabilities $T(s'|s, a)$), then we can modify this data structure, by adding to each transition probability in the list the sum of the transition probabilities of all next-states appearing earlier in the list, in the same amount of time it takes just to read the original data structure (transition function) as input. Our modified data structure combined with a random number generator clearly allows us to draw random next-states in time logarithmically (by binary search) in the number of next states (at most S) (Knuth, 1969). Thus, we can simulate a generative model in logarithmic time with a constant-time setup cost. Knuth (1969) also describes a more sophisticated method that allows constant-time queries, but requires sorting the input lists in the pre-processing stage.

Implementation of Rand-RTDP An implementation of Rand-RTDP is provided in Algorithm 1.

Algorithm 1 Rand-RTDP

```

0: Inputs:  $\gamma, S, A, m, \epsilon_1$ 
1: for all  $(s, a)$  do
2:    $Q(s, a) \leftarrow 1/(1 - \gamma)$  // action-value estimates
3:    $\text{LAU}(s, a) \leftarrow 0$  // time of last attempted update
4: end for
5:  $t^* \leftarrow 0$  // time of most recent Q-value change
6: for  $t = 1, 2, 3, \dots$  do
7:   Let  $s$  denote the state at time  $t$ .
8:   Choose action  $a := \arg\max_{a' \in A} Q(s, a')$ .
9:   if  $\text{LAU}(s, a) \leq t^*$  then
10:    Draw  $m$  random next states,  $s_1, \dots, s_m$  and  $m$  random immediate rewards  $r_1, \dots, r_m$  from the generative model for  $(s, a)$ .
11:     $q \leftarrow \frac{1}{m} \sum_{i=1}^m (r_i + \gamma \max_{a' \in A} Q(s_i, a'))$ 
12:    if  $Q(s, a) - q \geq 2\epsilon_1$  then
13:       $Q(s, a) \leftarrow q + \epsilon_1$ 
14:       $t^* \leftarrow t$ 
15:    end if
16:     $\text{LAU}(s, a) \leftarrow t$ 
17:  end if
18: end for

```

Analysis and Comparison

Computational Complexity

In this section, we analyze the per-step computational complexity of the two algorithms. We measure complexity assuming individual numbers require unit storage and can be manipulated arithmetically in unit time⁴.

The per-step computation cost of RTDP is $O(K + \ln(A))$, (where K is the number of states that can be reached in

⁴We have tried not to abuse this assumption.

one step with positive probability), which is highly detrimental in domains with a large number of next states (in the worst case, $K = S$). The per-step computational cost of Rand-RTDP is $O(m \cdot G + \ln(A))$ where G is the cost of drawing a single sample from the generative model. In the analysis section we provide a value for m that is sufficient for Rand-RTDP to be PAC-MDP. Using this value of m (see Proposition 3) and assuming that $G = O(\ln(K))$, the worst-case per-step computational cost of Rand-RTDP is $O(m \cdot G + \ln(A)) =$

$$O\left(\frac{\ln(SA/(\epsilon\delta(1-\gamma)))}{\epsilon^2(1-\gamma)^4} \ln(K) + \ln(A)\right),$$

which is a vast improvement over RTDP when only the dependence on S is considered. Specifically, when ϵ, δ, A , and γ are taken as constants, the computational cost is $O(\ln(S) \ln(K)) = O(\ln^2(S))$. Note that the computational requirements of both algorithms involve an additive factor of $O(\ln(A))$, due to storing and accessing the maximum action-value estimate for the current state after an update occurs via a priority queue.

General Framework

The algorithms we consider have the following form:

Definition 1 Suppose an RL algorithm \mathcal{A} maintains a value, denoted $Q(s, a)$, for each state-action pair (s, a) with $s \in S$ and $a \in A$. Let $Q_t(s, a)$ denote the estimate for (s, a) immediately before the t th action of the agent. We say that \mathcal{A} is a **greedy algorithm** if the t th action of \mathcal{A} , a_t , is $a_t := \arg\max_{a \in A} Q_t(s_t, a)$, where s_t is the t th state reached by the agent.

The following is a definition of a new MDP that will be useful in our analysis.

Definition 2 (Known State-Action MDP) For an MDP $M = \langle S, A, T, R, \gamma \rangle$, a given set of action-values, $Q(s, a)$ for each state-action pair (s, a) , and a set K of state-action pairs, we define the **known state-action MDP** $M_K = \langle S \cup \{s_0\}, A, T_K, R_K, \gamma \rangle$ as follows. Let s_0 be an additional state added to the state space of M . Under all actions from s_0 , the agent is returned to s_0 with probability 1. The reward for taking any action from s_0 is 0. For all $(s, a) \in K$, $R_K(s, a) = R(s, a)$ and $T_K(\cdot|s, a) = T(\cdot|s, a)$. For all $(s, a) \notin K$, $R_K(s, a) = Q(s, a)$ and $T(s_0|s, a) = 1$.

The known state-action MDP is a generalization of the standard notions of a “known state MDP” of Kearns and Singh (2002) and Kakade (2003). It is an MDP whose dynamics (reward and transition functions) are equal to the true dynamics of M for a subset of the state-action pairs (specifically those in K). For all other state-action pairs, the value of taking those state-action pairs in M_K (and following any policy from that point on) is equal to the current action-value estimates $Q(s, a)$. We intuitively view K as a set of state-action pairs for which the agent has sufficiently accurate estimates of their dynamics.

Definition 3 (Escape Event) Suppose that for some algorithm there is a set of state-action pairs K_t defined during

each timestep t . Let A_K be defined as the event, called the **escape event**, that some state-action pair (s, a) is experienced by the agent at time t such that $(s, a) \notin K_t$.

Note that all learning algorithms we consider take ϵ and δ as input. We let $\mathcal{A}(\epsilon, \delta)$ denote the version of algorithm \mathcal{A} parameterized with ϵ and δ . The following proposition is taken from Strehl et al. (2006).

Proposition 1 *Let $\mathcal{A}(\epsilon, \delta)$ be a greedy learning algorithm, such that for every timestep t , there exists a set K_t of state-action pairs. We assume that $K_t = K_{t+1}$ unless during timestep t , an update to some action-value occurs or the event A_K happens. Let M_{K_t} be the known state-action MDP and π_t be the current greedy policy, that is, for all states s , $\pi_t(s) = \operatorname{argmax}_a Q_t(s, a)$. Suppose that for any inputs ϵ and δ , with probability at least $1 - \delta$, the following conditions hold: (1) $Q_t(s, a) \geq Q^*(s, a) - \epsilon$ (optimism), (2) $V_t(s) - V_{M_{K_t}}^{\pi_t}(s) \leq \epsilon$ (accuracy), and (3) the total number of updates of action-value estimates plus the number of times the escape event from K_t , A_K , can occur is bounded by $\zeta(\epsilon, \delta)$ (learning complexity). Then, when $\mathcal{A}(\epsilon, \delta)$ is executed on any MDP M , it will follow a 4ϵ -optimal policy from its current state on all but*

$$O\left(\frac{\zeta(\epsilon, \delta)}{\epsilon(1-\gamma)^2} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps, with probability at least $1 - 2\delta$.

Our proofs work by the following scheme (for both RTDP and Rand-RTDP): (1) Define a set of known-state actions for each timestep t . (2) Show that these satisfy the conditions of Proposition 1.

Sample Complexity Bound for RTDP

In this section, we prove that RTDP is PAC-MDP with a sample complexity bound of

$$O\left(\frac{SA}{\epsilon^2(1-\gamma)^4} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right). \quad (5)$$

First, we show that RTDP has the property of ‘‘optimism’’.

Lemma 1 *During execution of RTDP, $Q_t(s, a) \geq Q^*(s, a)$ holds for all time steps t and state-action pairs (s, a) .*

Proof: The proof is by induction on the timestep t . For the base case, note that $Q_1(s, a) = 1/(1-\gamma) \geq Q^*(s, a)$ for all (s, a) . Now, suppose the claim holds for all timesteps less than or equal to t . Thus, we have that $Q_t(s, a) \geq Q^*(s, a)$ for all (s, a) . We also have that $V_t(s) = \max_{a \in A} Q_t(s, a) \geq Q_t(s, \operatorname{argmax}_a Q^*(s, a)) \geq Q^*(s, \operatorname{argmax}_a Q^*(s, a)) = V^*(s)$. Suppose s is the t th state reached and a is the action taken at time t . Of the action-values maintained by RTDP, only $Q(s, a)$ can change. Specifically, if it does get updated, then we have that $Q_{t+1}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_t(s') \geq R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') = Q^*(s, a)$. \square

Proposition 2 *The RTDP algorithm with $\epsilon_1 = \epsilon(1-\gamma)$ is PAC-MDP.*

Proof: We apply Proposition 1. By Lemma 1, Condition (1) is satisfied. During timestep t of the execution of RTDP, we define K_t to be the set of all state-action pairs (s, a) such that:

$$Q_t(s, a) - \left(R(s, a) + \gamma \sum_{s'} T(s'|s, a) V_t(s') \right) \leq \epsilon_1. \quad (6)$$

We claim that $V_t(s) - V_{M_{K_t}}^{\pi_t}(s) \leq \epsilon_1/(1-\gamma)$ always holds. To verify, note that $V_{M_{K_t}}^{\pi_t}$ is the solution to the following set of equations:

$$\begin{aligned} V_{M_{K_t}}^{\pi_t}(s) &= R(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s'|s, \pi_t(s)) V_{M_{K_t}}^{\pi_t}(s'), & \text{if } (s, \pi_t(s)) \in K_t, \\ V_{M_{K_t}}^{\pi_t}(s) &= Q_t(s, \pi_t(s)), & \text{if } (s, \pi_t(s)) \notin K_t. \end{aligned}$$

The vector V_t is the solution to a similar set of equations except with some additional positive reward terms, each bounded by ϵ_1 (see Equation 6). It follows that $V_t(s) - V_{M_{K_t}}^{\pi_t}(s) \leq \epsilon_1/(1-\gamma)$. Thus, by letting $\epsilon_1 = \epsilon(1-\gamma)$, we satisfy $V_t(s) - V_{M_{K_t}}^{\pi_t}(s) \leq \epsilon$, as desired (to fulfill Condition (2) of Proposition 1). Finally, note that by definition, the algorithm performs updates if and only if the event A_K occurs. Hence, it is sufficient to bound the number of times that A_K occurs. Every successful update of $Q(s, a)$ decreases its value by at least ϵ_1 . Since its value is initialized to $1/(1-\gamma)$, we have that each state-action pair can be updated at most $1/(\epsilon_1(1-\gamma))$ times, for a total of at most $SA/(\epsilon_1(1-\gamma)) = SA/(\epsilon(1-\gamma)^2)$ timesteps t such that A_K can occur. Ignoring log factors, this analysis leads to a total sample complexity bound of

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^4}\right). \quad (7)$$

\square

Sample Complexity Bound for Rand-RTDP

In this section, we prove that Rand-RTDP is PAC-MDP also with a sample complexity bound of

$$O\left(\frac{SA}{\epsilon^2(1-\gamma)^4} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right). \quad (8)$$

For Rand-RTDP, we define K_t , during timestep t of the execution of Rand-RTDP, to be the set of all state-action pairs (s, a) such that:

$$Q_t(s, a) - \left(R(s, a) + \gamma \sum_{s'} T(s'|s, a) V_t(s') \right) \leq 3\epsilon_1. \quad (9)$$

We will show, in the proof of the following lemma, that Rand-RTDP will perform at most κ attempted updates during any given infinite-length run through any MDP, where

$$\kappa := SA \left(1 + \frac{SA}{\epsilon_1(1-\gamma)} \right)$$

Lemma 2 *During execution of Rand-RTDP, if $m \geq \ln(2\kappa/\delta)/(2\epsilon_1^2(1-\gamma)^2)$, then on every timestep t such that (escape event) A_K occurs, a successful update also occurs, with probability at least $1 - \delta/2$.*

Proof: First, we bound the number of updates and attempted updates. Note that each time a successful update occurs for (s, a) , this event results in a decrease to the current action-value estimate, $Q(s, a)$, of at least ϵ_1 . As $Q(s, a)$ is initialized to $1/(1-\gamma)$ and it cannot fall below zero, it can be successfully updated at most $1/(\epsilon_1(1-\gamma))$ times. Hence, the total number of updates is bounded by $SA/(\epsilon_1(1-\gamma))$. Now, note that when a state-action pair, (s, a) , is first experienced, it will always perform an attempted update. Next, another attempted update of (s, a) will occur only when at least some action-value has been successfully updated (due to Line 9 in Algorithm 1) since the last attempted update (of (s, a)). Thus, the total number of attempted updates is bounded by $SA(1 + SA/(\epsilon_1(1-\gamma))) = \kappa$.

Suppose at time t , (s, a) is experienced and results in an attempted update. The Rand-RTDP agent draws m next states s_1, \dots, s_m and m immediate rewards r_1, \dots, r_m from its generative model. Define the random variables $X_i := r_i + \gamma V_t(s_i)$ for $i = 1, \dots, m$. Note that the X_i are independently and identically distributed with mean $E[X_i] = R(s, a) + \gamma \sum_{s'} T(s'|s, a) V_t(s')$. Hence, by the Hoeffding bound we have that

$$\Pr \left[\frac{1}{m} \sum_{i=1}^m X_i - E[X_1] \geq \epsilon_1 \right] \leq e^{-2\epsilon_1^2 m (1-\gamma)^2}.$$

Thus, by our choice of m we have that

$$\frac{1}{m} \sum_{i=1}^m X_i - \epsilon_1 < E[X_1] \quad (10)$$

will hold with probability at least $1 - \delta/(2\kappa)$. We claim that if $(s, a) \notin K_t$ (so that the event A_K occurred at time t), then this attempted update will be successful if Equation 10 holds. To see this fact, note that the difference between the old action-value estimate ($Q_t(s, a)$) and the candidate for the new action-value estimate is at least ϵ_1 :

$$\begin{aligned} & Q_t(s, a) - \left(\frac{1}{m} \sum_{i=1}^m (r_i + \gamma V_t(s_i)) + \epsilon_1 \right) \\ &= Q_t(s, a) - (1/m) \sum_{i=1}^m (X_i) - \epsilon_1 \\ &> Q_t(s, a) - E[X_1] - 2\epsilon_1 > 3\epsilon_1 - 2\epsilon_1 = \epsilon_1. \end{aligned}$$

The first step follows from the definition of X_i . The second step follows from Equation 10. The third step follows from the fact that $(s, a) \notin K_t$ (see Equation 9). Since there are at most κ attempted updates, the chance that the update is not successful (which is at most the probability that Condition 10 does not hold) is at most $\delta/2$, for any timestep that satisfies the above restrictions (an attempted update occurs for $(s, a) \notin K_t$ at time t), by the union bound.

What we have shown is that, with high probability, every attempted update of (s, a) at times t such that $(s, a) \notin K_t$

will be successful. It can be shown that if this statement holds for execution of the algorithm up to some time t , and $(s, a) \notin K_t$ is the state-action pair at time t , then an attempted update will occur for (s, a) . It is not hard to prove but is a bit tedious, so we omit it here. \square

Next, we show that Rand-RTDP has the property of ‘‘optimism’’ with high probability.

Lemma 3 *During execution of Rand-RTDP, if $m \geq \ln(2\kappa/\delta)/(2\epsilon_1^2(1-\gamma)^2)$, then $Q_t(s, a) \geq Q^*(s, a)$ holds for all time steps t and state-action pairs (s, a) , with probability at least $1 - \delta/2$.*

Proof: The proof is by induction on the timestep t . When $t = 1$, no actions have been taken by the agent yet, and we have that $Q_1(s, a) = 1/(1-\gamma) \geq Q^*(s, a)$ for all (s, a) , due to optimistic initialization. Now, suppose that $Q_t(s, a) \geq Q^*(s, a)$ holds for all (s, a) at some timestep t . Suppose that a is the t th action, and it was taken from state s (s was the t th state reached by the agent). If this step does not result in an attempted update, then no action-value estimates change, and we are done. Thus, suppose that an attempted update does occur. The Rand-RTDP agent draws m next states s_1, \dots, s_m and m immediate rewards r_1, \dots, r_m from its generative model (to be used in the attempted update). By Equation 3, if the update is successful, then $Q_{t+1}(s, a) = (1/m) \sum_{i=1}^m (r_i + \gamma V_t(s_i)) + \epsilon_1$ (if it’s not successful then it still upper bounds $Q^*(s, a)$, by the inductive assumption). However, we have that $(1/m) \sum_{i=1}^m (r_i + \gamma V_t(s_i)) + \epsilon_1 \geq (1/m) \sum_{i=1}^m (r_i + \gamma V^*(s_i)) + \epsilon_1$, by the induction hypothesis. Applying the Hoeffding bound (with random variables $Y_i := r_i + \gamma V^*(s_i)$ for $i = 1, \dots, m$) and the union bound as in the proof of Lemma 2, we then have that $Q_{t+1}(s, a) \geq Q^*(s, a)$ holds, with probability at least $1 - \delta/2$, over all attempted updates of any state-action pair. \square

Proposition 3 *The Rand-RTDP algorithm with $m = \frac{1}{2\epsilon_1^2(1-\gamma)^2} \ln \left(\frac{2\kappa}{\delta} \right) = O \left(\frac{1}{\epsilon^2(1-\gamma)^4} \ln \left(\frac{SA}{\epsilon(1-\gamma)\delta} \right) \right)$ and $\epsilon_1 = \epsilon(1-\gamma)/3$ is PAC-MDP.*

Proof sketch: The proof is very similar to that of Proposition 2, and leads to a total sample complexity bound of

$$\tilde{O} \left(\frac{SA}{\epsilon^2(1-\gamma)^4} \right). \quad (11)$$

\square

Note that the PAC sample complexity bound we’ve proved for RTDP (Equation 7) is asymptotically the same as that for Rand-RTDP (see Equation 11). This result suggests that it may be possible to use Rand-RTDP instead of RTDP on large-scale problems where computation cost is critical, without sacrificing much in terms of convergence performance.

Experiments

For each experiment, we recorded the cumulative reward obtained (to stand in for sample complexity)⁵ by the algorithm

⁵The true sample complexity of the algorithm is expensive, and often impossible, to compute in practice.

for a fixed number of timesteps (50,000). We also measured the number of backups (specifically, the number of times the algorithm computes the value of some reachable next-state, $V(s')$ computed by the algorithm (to measure computational complexity). Our experiments were designed to supplement the theoretical analysis and are not meant as a thorough evaluation of the algorithms.

The algorithms were tested on random MDPs generated as follows, for parameters $S = 500$, $A = 2$, and $\gamma = 0.95$. To guarantee that every state was reachable from every other state, random Hamiltonian circuits (to ensure connectivity) connecting every state were constructed under each action, and a probability of 0.1 was assigned to each of the corresponding transitions. Then, for each state-action pair, the transition function was completed by assigning random probabilities (summing up to 0.9) to 99 randomly selected next states. Therefore, for each state-action pair, there were at most 100 next-states. To make the MDP more interesting, we constructed the reward function so that $R(s, a)$ tended to be large if the index of s was large; specifically, the mean reward, $R(s, a)$, was randomly selected with mean proportional to the index of s .⁶ Each experiment was repeated 100 times and the results were averaged.

The results of RTDP are summarized below:

Param	Reward	Backups
$\epsilon_1 = 0.1$	$25,248 \pm 13$	$4,476,325 \pm 589$
$\epsilon_1 = 0.2$	$25,235 \pm 12$	$4,396,626 \pm 889$
$\epsilon_1 = 0.3$	$25,208 \pm 15$	$4,288,353 \pm 4,505$
$\epsilon_1 = 0.4$	$25,197 \pm 16$	$4,005,369 \pm 31,161$

The results of Rand-RTDP are summarized below:

Param	Reward	Backups
$(\epsilon_1 = 0.1, m = 30)$	$25,124 \pm 14$	$1,469,122 \pm 184$
$(\epsilon_1 = 0.2, m = 30)$	$25,138 \pm 12$	$1,450,281 \pm 227$
$(\epsilon_1 = 0.3, m = 30)$	$25,126 \pm 15$	$1,435,197 \pm 269$
$(\epsilon_1 = 0.4, m = 30)$	$25,104 \pm 15$	$1,420,445 \pm 343$
$(\epsilon_1 = 0.1, m = 50)$	$25,159 \pm 13$	$2,448,039 \pm 308$
$(\epsilon_1 = 0.2, m = 50)$	$25,150 \pm 15$	$2,415,387 \pm 411$
$(\epsilon_1 = 0.3, m = 50)$	$25,148 \pm 14$	$2,388,299 \pm 456$
$(\epsilon_1 = 0.4, m = 50)$	$25,139 \pm 13$	$2,361,596 \pm 596$

As a comparison, we also ran the optimal policy, which achieved $25,873 \pm 13$ cumulative reward, and the policy that chooses actions uniformly at random, which achieved $24,891 \pm 15$ cumulative reward. We see from the results that Rand-RTDP did not require as much computation as RTDP, but also didn't achieve quite as much reward.

Conclusion

In this paper, we have provided a theoretical analysis of the sample complexity of RTDP and a randomized variant of it, Rand-RTDP. When viewed as learning algorithms, both algorithms are shown to be efficient in the PAC-MDP framework. In particular, their sample complexity bounds have a sub-quadratic dependence on the sizes of the state and action

⁶We found that if the mean rewards are chosen uniformly at random, then the optimal policy almost always picks the action that maximizes the immediate reward, which is not interesting in the context of sequential decision making.

spaces, which are arguably the most important factors. To the best of our knowledge, this is the first sample complexity result for RTDP, although people have investigated its limit behavior (Barto et al., 1995; Bonet & Geffner, 2003). Furthermore, the Rand-RTDP algorithm improves the per-step computational cost over RTDP without an asymptotic increase in its sample complexity.

Acknowledgments

Thanks to the National Science Foundation (IIS-0325281). We also thank John Langford and Eric Wiewiora for discussions.

References

- Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72, 81–138.
- Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.
- Bonet, B., & Geffner, H. (2003). Labeled RTDP: Improving the convergence of real-time dynamic programming. *Proc. 13th International Conf. on Automated Planning and Scheduling* (pp. 12–21). Trento, Italy: AAAI Press.
- Brafman, R. I., & Tenenbholz, M. (2002). R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3, 213–231.
- Kakade, S. M. (2003). *On the sample complexity of reinforcement learning*. Doctoral dissertation, Gatsby Computational Neuroscience Unit, University College London.
- Kearns, M., Mansour, Y., & Ng, A. Y. (1999). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)* (pp. 1324–1331).
- Kearns, M. J., & Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49, 209–232.
- Knuth, D. (1969). *The art of computer programming: Vol 2 / seminumerical algorithms, chapter 3: Random numbers*. Addison-Wesley.
- Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence*, 42, 189–211.
- McMahan, H. B., Likhachev, M., & Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. *Proceedings of the Twenty-second International Conference on Machine Learning (ICML-05)* (pp. 569–576).
- Strehl, A. L., Li, L., & Littman, M. L. (2006). Incremental model-based learners with formal learning-time guarantees. To appear in *Proceedings of the Twenty-second Conference on Uncertainty in Artificial Intelligence (UAI-06)*.
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. The MIT Press.
- Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27, 1134–1142.
- Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8, 279–292.