PROBABLY APPROXIMATELY CORRECT (PAC) EXPLORATION IN REINFORCEMENT LEARNING

BY

ALEXANDER L. STREHL

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Michael Littman

and approved by

New Brunswick, New Jersey

October, 2007

ABSTRACT OF THE DISSERTATION

Probably Approximately Correct (PAC) Exploration in Reinforcement Learning

by Alexander L. Strehl

Dissertation Director: Michael Littman

Reinforcement Learning (RL) in finite state and action Markov Decision Processes is studied with an emphasis on the well-studied exploration problem. We provide a general RL framework that applies to all results in this thesis and to other results in RL that generalize the finite MDP assumption. We present two new versions of the *Model-Based Interval Estimation* (MBIE) algorithm and prove that they are both PAC-MDP. These algorithms are provably more efficient any than previously studied RL algorithms. We prove that many model-based algorithms (including R-MAX and MBIE) can be modified so that their worst-case per-step computational complexity is vastly improved without sacrificing their attractive theoretical guarantees. We show that it is possible to obtain PAC-MDP bounds with a model-free algorithm called Delayed Q-learning.

Acknowledgements

Many people helped with my research career and with problems in this thesis. In particular I would like to thank Lihong Li, Martin Zinkovich, John Langford, Eric Wiewiora, and Csaba Szepesvári. I am greatly indebted to my advisor, Michael Littman, who provided me with an immense amount of support and guidance. I am also thankful to my undergraduate advisor, Dinesh Sarvate, who imparted me with a love for research.

Many of the results presented in this thesis resulted from my joint work with other researchers (Michael Littman, Lihong Li, John Langford, and Eric Wiewiora) and have been published in AI conferences and journals. These papers include (Strehl & Littman, 2004; Strehl & Littman, 2005; Strehl et al., 2006c; Strehl et al., 2006a; Strehl et al., 2006b; Strehl & Littman, 2007).

Table of Contents

List of Figures

Introduction

In this thesis, we consider some fundamental problems in the field of Reinforcement Learning (Sutton & Barto, 1998). In particular, our focus is on the problem of exploration: how does an agent determine whether to act to gain new information (explore) or to act consistently with past experience to maximize reward (exploit). We make the assumption that the environment can be described by a finite discounted *Markov Decision Process* (Puterman, 1994) although several extensions will also be considered. Algorithms will be presented and analyzed. The important properties of a learning algorithm are its *computational complexity* (to a lesser extent, its *space complexity*) and its *sample complexity*, which measures it performance. When determining the non-computational performance of an algorithm (i.e. how quickly does it learn) we will use a framework (PAC-MDP) based on the Probably Approximately Correct (PAC) framework (Valiant, 1984). In particular, our focus will be on algorithms that accept a precision parameter (ϵ) and a failure-rate parameter (δ). We will then require our algorithms to make at most a small (polynomial) number of mistrials (actions that are more than ϵ worse than the best action) with high probability (at least $1 - \delta$). The bound on the number of mistrials will be called the sample complexity of the algorithm. This terminology is borrowed from Kakade (2003), who called the same quantity the *sample complexity of exploration*.

Main Results

The main results presented in this thesis are as follows:

1. We provide a general RL framework that applies to all results in this thesis and to other results in RL that generalize the finite MDP assumption.

2. We present two new versions of the *Model-Based Interval Estimation* (MBIE) algorithm and prove that they are both PAC-MDP. These algorithms are provably more efficient any than previously studied RL algorithms.

3. We prove that many model-based algorithms (including R-MAX and MBIE) can be modified so that their worst-case per-step computational complexity is vastly improved without sacrificing their attractive theoretical guarantees.

4. We show that it is possible to obtain PAC-MDP bounds with a model-free algorithm called Delayed Q-learning.

Table

Here's a table summarizing the PAC-MDP sample complexity and per-step computational complexity bounds that we will prove:

Summary Table			
Algorithm	Comp. Complexity	Space Complexity	Sample Complexity
Q-Learning	$O(\ln(A))$	$O(SA)$	Unknown, Possibly EXP
DQL	$O(\ln(A))$	$O(SA)$	$\tilde{O}\left(\frac{SA}{\epsilon^4(1-\gamma)^8}\right)$
DQL-IE	$O(\ln(A))$	$O(SA)$	$\tilde{O}\left(\frac{SA}{\epsilon^4(1-\gamma)^8}\right)$
RTDP-RMAX	$O(S+\ln(A))$	$O(S^2A)$	$\tilde{O}\left(\frac{S^2A}{\epsilon^3(1-\gamma)^6}\right)$
RTDP-IE	$O(S+\ln(A))$	$O(S^2A)$	$\tilde{O}\left(\frac{S^2A}{\epsilon^3(1-\gamma)^6}\right)$
RMAX	$O\left(\frac{SA(S+\ln(A))\ln\frac{1}{\epsilon(1-\gamma)}}{1-\gamma}\right)$	$O(S^2A)$	$\tilde{O}\left(\frac{S^2A}{\epsilon^3(1-\gamma)^6}\right)$
MBIE-EB	$O\left(\frac{SA(S+\ln(A))\ln\frac{1}{\epsilon(1-\gamma)}}{1-\gamma}\right)$	$O(S^2A)$	$\tilde{O}\left(\frac{S^2A}{\epsilon^3(1-\gamma)^6}\right)$

We've used the abbreviations DQL and DQL-IE for the Delayed Q-learning and the Delayed Q-learning with IE algorithms, respectively. The second column shows the per-timestep computational complexity of the algorithms. The last column shows the best known PAC-MDP sample complexity bounds for the algorithms. It is worth emphasizing, especially in reference to sample complexity, is that these are upper bounds. What should not be concluded from the table is that the Delayed Q-learning variants are superior to the other algorithms in terms of sample complexity. First, the upper bounds themselves clearly do not dominate (consider the ϵ and $(1-\gamma)$ terms). They do, however, dominate when we consider only the S and A terms. Second, the upper bounds may not be tight. One important open problem in theoretical RL is whether or not a model-based algorithm, such as R-MAX, is PAC-MDP with a *sparse model*. Specifically, can we reduce the sample complexity bound to $\tilde{O}(SA/(\epsilon^3(1-\gamma)^6)$ or better by using a model-based algorithm whose model-size parameter m is limited to something that depends only logarithmically on the number of states S. This conjecture is presented and discussed in Chapter 8 of Kakade's thesis (Kakade, 2003) and has important implications in terms of the fundamental complexity of exploration.

Another point to emphasize is that the bounds displayed in the above table are worst-case. We have found empirically that the IE approach to exploration performs better than the naïve approach, yet this fact is not reflected in the bounds.

Chapter 1

Formal Definitions, Notation, and Basic Results

This section introduces the Markov Decision Process (MDP) notation (Sutton & Barto, 1998). Let $\mathcal{P}_S$ denote the set of all probability distributions over the set S. A finite MDP M is a five tuple $\langle S, A, T, \mathcal{R}, \gamma \rangle$, where S is a finite set called the state space, A is a finite set called the action space, $T : S \times A \rightarrow \mathcal{P}_S$ is the transition distribution, $\mathcal{R} : S \times A \rightarrow \mathcal{P}_{\mathbb{R}}$ is the reward distribution, and $0 \leq \gamma < 1$ is a discount factor on the summed sequence of rewards. We call the elements of S and A states and actions, respectively. We allow a slight abuse of notation and also use S and A for the number of states and actions, respectively. We let $T(s'|s,a)$ denote the transition probability of state s' of the distribution $T(s,a)$. In addition, $R(s,a)$ denotes the expected value of the distribution $\mathcal{R}(s,a)$.

A *policy* is any strategy for choosing actions. A stationary policy is one that produces an action based on only the current state, ignoring the rest of the agent's history. We assume (unless noted otherwise) that rewards all lie in the interval $[0,1]$. For any policy π, let $V^{\pi}_{M}(s)$ $(Q^{\pi}_{M}(s,a))$ denote the discounted, infinite-horizon value (action-value) function for π in M (which may be omitted from the notation) from state s. If H is a positive integer, let $V^{\pi}_{M}(s,H)$ denote the H-step value of policy π from s. If π is non-stationary, then s is replaced by a *partial path* $c_t = s_1, a_1, r_1, \ldots, s_t$, in the previous definitions. Specifically, let s_t and r_t be the tth encountered state and received reward, respectively, resulting from execution of policy π in some MDP M. Then, $V^{\pi}_{M}(c_t) = E[\sum_{j=0}^{\infty} \gamma^j r_{t+j} | c_t]$ and $V^{\pi}_{M}(c_t, H) = E[\sum_{j=0}^{H-1} \gamma^j r_{t+j} | c_t]$. These expectations are taken over all possible infinite paths the agent might follow in the future. 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 $1/(1-\gamma)$ by the assumption of a maximum reward of 1. Please see Figure 1.1 for an example of an MDP.

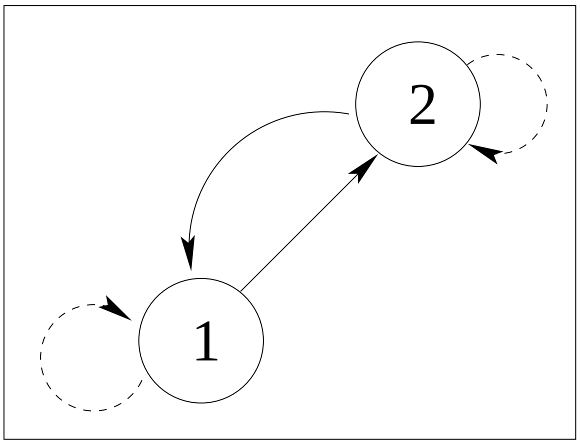


Figure 1.1: An example of a deterministic MDP. The states are represented as nodes and the actions as edges. There are two states and actions. The first is represented as a solid line and the second as a dashed line. The rewards are not shown, but are 0 for both states and actions except that from state 2 under action 1 a reward of 1 is obtained. The optimal policy for all discount factors is to take action 1 from both states.

1.1 The Planning Problem

In the planning problem for MDPs, the algorithm is given as input an MDP M and must produce a policy π that is either optimal or approximately optimal.

1.2 The Learning Problem

Suppose that the learner (also called the *agent*) receives S, A, and γ as input. The learning problem is defined as follows. The agent always occupies a single state s of the MDP M. The agent is told this state and must choose an action a. It then receives an *immediate reward* $r \sim \mathcal{R}(s, a)$ and is transported to a *next state* $s' \sim T(s, a)$. This procedure then repeats forever. The first state occupied by the agent may be chosen arbitrarily. Intuitively, the solution or goal of the problem is to obtain as large as possible reward in as short as possible time. We define a *timestep* to be a single interaction with the environment, as described above. The tth timestep encompasses the process of choosing the tth action. We also define an *experience* of state-action pair

(s, a) to refer to the event of taking action a from state s.

1.3 Learning Efficiently

A reasonable notion of learning efficiency in an MDP is to require an efficient algorithm to achieve near-optimal (expected) performance with high probability. An algorithm that satisfies such a condition can generally be said to be *probably approximately correct* (PAC) for MDPs. The PAC notion was originally developed in the supervised learning community, where a classifier, while learning, does not influence the distribution of training instances it receives (Valiant, 1984). In reinforcement learning, learning and behaving are intertwined, with the decisions made during learning profoundly affecting the available experience.

In applying the PAC notion in the reinforcement-learning setting, researchers have examined definitions that vary in the degree to which the natural mixing of learning and evaluation is restricted for the sake of analytic tractability. We survey these notions next.

1.3.1 PAC reinforcement learning

One difficulty in comparing reinforcement-learning algorithms is that decisions made early in learning can affect significantly the rewards available later. As an extreme example, imagine that the first action choice causes a transition to one of two disjoint state spaces, one with generally large rewards and one with generally small rewards. To avoid unfairly penalizing learners that make the wrong arbitrary first choice, Fiechter (1997) explored a set of PAC-learning definitions that assumed that learning is conducted in trials of constant length from a fixed start state. Under this *reset assumption*, the task of the learner is to find a near-optimal policy from the start state given repeated visits to this state.

Fiechter's notion of PAC reinforcement-learning algorithms is extremely attractive because it is very simple, intuitive, and fits nicely with the original PAC definition. However, the assumption of a reset is not present in the most natural reinforcement learning problem. Theoretically, the reset model is stronger (less general) than the standard reinforcement learning model. For example, in the reset model it is possible to find arbitrarily good policies, with high probability, after a number of experiences that does not depend on the size of the state space. However, this is not possible in general when no reset is available (Kakade, 2003).

1.3.2 Kearns and Singh's PAC Metric

Kearns and Singh (2002) provided an algorithm, E^3, which was proven to obtain near-optimal return quickly in both the average reward and discounted reward settings, without a reset assumption. Kearns and Singh note that care must be taken when defining an optimality criterion for discounted MDPs. One possible goal is to achieve near-optimal return from the initial state. However, this goal cannot be achieved because discounting makes it impossible for the learner to recover from early mistakes, which are inevitable given that the environment is initially unknown. Another possible goal is to obtain return that is nearly optimal when averaged across all visited states, but this criterion turns out to be equivalent to maximizing average return—the discount factor ultimately plays no role. Ultimately, Kearns and Singh opt for finding a near-optimal policy from the final state reached by the algorithm. In fact, we show that averaging discounted return is a meaningful criterion if it is the loss (relative to the optimal policy from each visited state) that is averaged.

1.3.3 Sample Complexity of Exploration

While Kearns and Singh's notion of efficiency applies to a more general reinforcement-learning problem than does Fiechter's, it still includes an unnatural separation between learning and evaluation. Kakade (2003) introduced a PAC performance metric that is more "online" in that it evaluates the behavior of the learning algorithm itself as opposed to a separate policy that it outputs. As in Kearns and Singh's definition, learning takes place over one long path through the MDP. At time t, the partial path $c_t = s_1, a_1, r_1 \ldots, s_t$ is used to determine a next action a_t. The algorithm itself can be viewed as a non-stationary policy. In our notation, this policy has expected value $V^{\mathcal{A}}(c_t)$, where $\mathcal{A}$ is the learning algorithm.

Definition 1 *(Kakade, 2003) Let* $c = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots)$ *be a path generated by executing an algorithm* $\mathcal{A}$ *in an MDP* M*. For any fixed* $\epsilon > 0$*, the* **sample complexity of exploration** *(***sample complexity***, for short) of* $\mathcal{A}$ *with respect to* c *is the number of timesteps* t *such that the policy at time* t*,* $\mathcal{A}_t$*, is not* ϵ*-optimal from the current state,* s_t *at time* t *(formally,* $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$*).*

In other words, the sample complexity is the number of timesteps, over the course of any run, for which the learning algorithm $\mathcal{A}$ is not executing an ϵ-optimal policy from its current state. $\mathcal{A}$ is PAC in this setting if its sample complexity can be bounded by a number polynomial in the relevant quantities with high probability. Kakade showed that the Rmax algorithm (Brafman & Tennenholtz, 2002) satisfies this condition. We will use Kakade's (2003) definition as the standard.

Definition 2 *An algorithm* $\mathcal{A}$ *is 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* $\mathcal{A}$ *are less than some polynomial in the relevant quantities* $(|S|, |A|, 1/\epsilon, 1/\delta, 1/(1-\gamma))$*, with probability at least* $1 - \delta$*. For convenience, we may also say that* $\mathcal{A}$ *is* **PAC-MDP***.*

One thing to note is that we only restrict a PAC-MDP algorithm from behaving poorly (non-ϵ-optimally) on more than a small (polynomially) number of timesteps. We don't place any limitations on when the algorithm acts poorly. This is in contrast to the original PAC notion which is more "off-line" in that it requires the algorithm to make all its mistakes ahead of time before identifying a near-optimal policy.

This difference is necessary. In any given MDP it may take an arbitrarily long time to reach some section of the state space. Once that section is reached we expect any learning algorithm to make some mistakes. Thus, we can hope only to bound the number of mistakes, but can say nothing about when they happen. The first two performance metrics above were able to sidestep this issue somewhat. In Fiechter's framework, a reset action allows a more "offline" PAC-MDP definition. In the performance metric used by Kearns and Singh (2002), a near-optimal policy is required only from a single state.

A second major difference between our notion of PAC-MDP and Valiant's original definition is that we don't require an agent to know when it has found a near-optimal policy, only that it executes one most of the time. In situations where we care only about the behavior of an algorithm, it doesn't make sense to require an agent to estimate its policy. In other situations, where there is a distinct separation between learning (exploring) and acting (exploiting), another performance metric, such as one of first two mentioned above, should be used. Note that requiring the algorithm to "know" when it has adequately learned a task may require the agent to explicitly estimate the value of its current policy. This may complicate the algorithm (for example, E^3 solves two MDP models instead of one).

1.3.4 Average Loss

Although sample complexity demands a tight integration between behavior and evaluation, the evaluation itself is still in terms of the near-optimality of expected values over future policies as opposed to the actual rewards the algorithm achieves while running. We introduce a new performance metric, *average loss*, defined in terms of the actual rewards received by the algorithm while learning. In the remainder of the section, we define average loss formally. It can be shown that efficiency in the sample complexity framework of Section 1.3.3 implies efficiency in the average loss framework (Strehl & Littman, 2007). Thus, throughout the rest of the thesis we will focus on the former even though the latter is of more practical interest.

Definition 3 *Suppose a learning algorithm is run for one trial of H steps in an MDP M. Let s_t be the state encountered on step t and let r_t be the t^{th} reward received. Then, the* **instantaneous loss** *of the agent is $il(t) = V^*(s_t) - \sum_{i=t}^{H} \gamma^{i-t} r_i$, the difference between the optimal value function at state s_t and the actual discounted return of the agent from time t until the end of the trial. The quantity $l = \frac{1}{H} \sum_{t=1}^{H} il(t)$ is called the*

average loss *over the sequence of states encountered.*

In definition 3, the quantity H should be sufficiently large, say $H \gg 1/(1-\gamma)$, because otherwise there is not enough information to evaluate the algorithm's performance. A learning algorithm is *PAC-MDP in the average loss setting* if for any ϵ and δ, we can choose a value H, polynomial in the relevant quantities $(1/\epsilon, 1/\delta, |S|, |A|, 1/(1-\gamma))$, such that the average loss of the agent (following the learning algorithm) on a trial of H steps is guaranteed to be less than ϵ with probability at least $1-\delta$.

It helps to visualize average loss in the following way. Suppose that an agent produces the following trajectory through an MDP.

$$s_1,\ a_1,\ r_1,\ s_2,\ a_2,\ r_2,\ \ldots,\ s_H,\ a_H,\ r_H$$

The trajectory is made up of states, $s_t \in S$; actions, $a_t \in A$; and rewards, $r_t \in [0,1]$, for each timestep $t = 1, \ldots, H$. The instantaneous loss associated for each timestep is shown in the following table.

t	trajectory starting at time t	instantaneous loss: $il(t)$
1	$s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_H, a_H, r_H$	$V^*(s_1) - (r_1 + \gamma r_2 + \ldots \gamma^{H-1} r_H)$
2	$s_2, a_2, r_2, \ldots, s_H, a_H, r_H$	$V^*(s_2) - (r_2 + \gamma r_3 + \ldots \gamma^{H-2} r_H)$
.	.	.
.	.	.
.	.	.
H	s_H, a_H, r_H	$V^*(s_H) - r_H$

The average loss is then the average of the instantaneous losses (in the rightmost column above).

1.4 General Learning Framework

We now develop some theoretical machinery to prove PAC-MDP statements about various algorithms. Our theory will be focused on algorithms that maintain a table of action values, $Q(s,a)$, for each state-action pair (denoted $Q_t(s,a)$ at time t)[1]. We also assume an algorithm always chooses actions greedily with respect to the action values. This constraint is not really a restriction, since we could define an algorithm's action values as 1 for the action it chooses and 0 for all other actions. However, the general framework is understood and developed more easily under the above assumptions. For convenience, we also introduce the notation $V(s)$ to denote $\max_a Q(s,a)$ and $V_t(s)$ to denote $V(s)$ at time t.

Definition 4 *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 $a \in \mathsf{A}$. Let $Q_t(s,a)$ denote the estimate for (s,a) immediately before the tth action of the agent. We say that $\mathcal{A}$ is a* **greedy algorithm** *if the tth action of $\mathcal{A}$, a_t, is $a_t := \operatorname{argmax}_{a\in\mathsf{A}} Q_t(s_t,a)$, where s_t is the tth state reached by the agent.*

For all algorithms, the action values $Q(\cdot,\cdot)$ are implicitly maintained in separate max-priority queues (implemented with max-heaps, say) for each state. Specifically, if $\mathsf{A} = \{a_1,\dots,a_k\}$ is the set of actions, then for each state s, the values $Q(s,a_1),\dots,Q(s,a_k)$ are stored in a single priority queue. Therefore, the operations $\max_{a'\in\mathsf{A}} Q(s,a)$ and $\operatorname{argmax}_{a'\in\mathsf{A}} Q(s,a)$, which appear in almost every algorithm, takes constant time, but the operation $Q(s,a) \leftarrow V$ for any value V takes $O(\ln(A))$ time (Cormen et al., 1990). It is possible that other data structures may result in faster algorithms.

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

Definition 5 *Let $M = \langle \mathcal{S}, \mathsf{A}, T, R, \gamma\rangle$ be an MDP with 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** *$M_K = \langle \mathcal{S} \cup \{z_{s,a} | (s,a) \notin K\}, \mathsf{A}, T_K, R_K, \gamma\rangle$ as follows. For each unknown state-action pair, $(s,a) \notin K$, we add a new state $z_{s,a}$ to M_K, which has self-loops for each action ($T_K(z_{s,a}|z_{s,a},\cdot) = 1$). 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 $(s,a) \notin K$, $R_K(s,a) = Q(s,a)(1-\gamma)$ and $T_K(z_{s,a}|s,a) = 1$. For the new states, the reward is $R_K(z_{s,a},\cdot) = Q(s,a)(1-\gamma)$.*

[1]The results don't rely on the algorithm having an explicit representation of each action value (for example, they could be implicitly held inside of a function approximator).

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 6 *Suppose that for algorithm $\mathcal{A}$ there is a set of state-action pairs K_t (we drop the subscript t if t is clear from context) defined during each timestep t and that depends only on the history of the agent up to timestep t (before the tth action). Let A_K be the event, called the* **escape event***, that some state-action pair (s, a) is experienced by the agent at time t, such that $(s, a) \notin K_t$.*

Our PAC-MDP proofs work by the following scheme (for whatever algorithm we have at hand): (1) Define a set of known state-actions for each timestep t. (2) Show that these satisfy the conditions of Theorem 1. The following is a well-known result of the Chernoff-Hoeffding Bound and will be needed later.

Lemma 1 *Suppose a weighted coin, when flipped, has probability $p > 0$ of landing with heads up. Then, for any positive integer k and real number $\delta \in (0, 1)$, after $O((k/p)\ln(1/\delta))$ tosses, with probability at least $1 - \delta$, we will observe k or more heads.*

Proof: Let a trial be a single act of tossing the coin. Consider performing n trials (n tosses), and let X_i be the random variable that is 1 if the ith toss is heads and 0 otherwise. Let $X = \sum_{i=1}^{n} X_i$ be the total number of heads observeds over all n trials. The multiplicative form of the Hoeffding bound states (for instance, see (Kearns & Vazirani, 1994a)) that

$$\Pr(X < (1 - \epsilon)pn) \le e^{-np\epsilon^2/2}. \quad (1.1)$$

We consider the case of $k \geq 4$, which clearly is sufficient for the asymptotic result stated in the lemma. Equation 1.1 says that we can upper bound the probability that $X \geq pn - \epsilon pn$ doesn't hold. Setting $\epsilon = 1/2$ and $n \geq 2k/p$, we see that it implies that $X \geq k$. Thus, we have only to show that the right hand side of Equation 1.1 is at most δ. This bound holds as long as $n \geq 2\ln(1/\delta)/(p\epsilon^2) = 8\ln(1/\delta)/p$. Therefore, letting $n \geq (2k/p)\ln(1/\delta)$ is sufficient, since $k \geq 4$. In summary, after $n = (2k/p)\max\{1, \ln(1/\delta)\}$ tosses, we are guaranteed to observe at least k heads with proability at least $1 - \delta$. $\square$

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 proof of Theorem 1 follows the structure of the work of Kakade (2003), but generalizes several key steps.

Theorem 1 *(Strehl et al., 2006a) Let $\mathcal{A}(\epsilon, \delta)$ be any greedy learning algorithm such that for every timestep t, there exists a set K_t of state-action pairs that depends only on the agent's history up to timestep t. We assume that $K_t = K_{t+1}$ unless, during timestep t, an update to some state-action value occurs or the escape 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, $\pi_t(s) = \text{argmax}_a Q_t(s, a)$. Suppose that for any inputs ϵ and δ, with probability at least $1 - \delta$, the following conditions hold for all states s, actions a, and timesteps t: (1) $V_t(s) \geq V^*(s) - \epsilon$ (optimism), (2) $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \epsilon$ (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 $1 - 2\delta$.

Proof: Suppose that the learning algorithm $\mathcal{A}(\epsilon, \delta)$ is executed on MDP M. Fix the history of the agent up to the tth timestep and let s_t be the tth state reached. Let $\mathcal{A}_t$ denote the current (non-stationary) policy of the agent. Let $H = \frac{1}{1-\gamma}\ln\frac{1}{\epsilon(1-\gamma)}$. From Lemma 2 of Kearns and Singh (2002), we have that $|V^{\pi}_{M_{K_t}}(s, H) - V^{\pi}_{M_{K_t}}(s)| \leq \epsilon$, for any state s and policy π. Let W denote the event that, after executing policy $\mathcal{A}_t$ from state s_t in M for H timesteps, one of the two following events occur: (a) the algorithm performs a successful update (a change to any of its action values) of some state-action pair (s,a), or (b) some state-action pair $(s,a) \notin K_t$ is experienced (escape event A_K). We have the following:

$$\begin{aligned}
V_M^{\mathcal{A}_t}(s_t,H) &\geq V_{M_{K_t}}^{\pi_t}(s_t,H) - \Pr(W)/(1-\gamma) \\
&\geq V_{M_{K_t}}^{\pi_t}(s_t) - \epsilon - \Pr(W)/(1-\gamma) \\
&\geq V(s_t) - 2\epsilon - \Pr(W)/(1-\gamma) \\
&\geq V^*(s_t) - 3\epsilon - \Pr(W)/(1-\gamma).
\end{aligned}$$

The first step above follows from the fact that following $\mathcal{A}_t$ in MDP M results in behavior identical to that of following π_t in M_{K_t} as long as no action-value updates are performed and no state-action pairs $(s,a) \notin K_t$ are experienced. This bound holds due to the following key observations:

- $\mathcal{A}$ is a greedy algorithm, and therefore matches π_t unless an action-value update occurs.
- M and M_{K_t} are identical on state-action pairs in K_t, and
- by assumption, the set K_t doesn't change unless event W occurs.

The bound then follows from the fact that the maximum difference between two value functions is $1/(1-\gamma)$. The second step follows from the definition of H above. The third and final steps follow from preconditions (2) and (1), respectively, of the proposition.

Now, suppose that $\Pr(W) < \epsilon(1-\gamma)$. Then, we have that the agent's policy on timestep t is 4ϵ-optimal:

$$V_M^{\mathcal{A}_t}(s_t) \geq V_M^{\mathcal{A}_t}(s_t,H) \geq V_M^*(s_t) - 4\epsilon.$$

Otherwise, we have that $\Pr(W) \geq \epsilon(1-\gamma)$, which implies that an agent following $\mathcal{A}_t$ will either perform a successful update in H timesteps, or encounter some $(s,a) \notin K_t$ in H timesteps, with probability at least $\epsilon(1-\gamma)$. Call such an event a "success".

Then, by Lemma 1, after $O(\frac{\zeta(\epsilon,\delta)H}{\epsilon(1-\gamma)}\ln 1/\delta)$ timesteps t where $\Pr(W) \geq \epsilon(1-\gamma)$, $\zeta(\epsilon,\delta)$ successes will occur, with probability at least $1-\delta$. Here, we have identified the event that a success occurs after following the agent's policy for H steps with the event that a coin lands with heads facing up.[2] However, by precondition (3) of the proposition, with probability at least $1-\delta$, $\zeta(\epsilon,\delta)$ is the maximum number of successes that will occur throughout the execution of the algorithm.

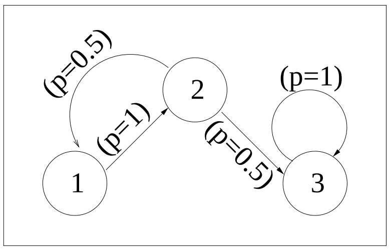
To summarize, we have shown that with probability at least $1-2\delta$, the agent will execute a 4ϵ-optimal policy on all but $O(\frac{\zeta(\epsilon,\delta)H}{\epsilon(1-\gamma)}\ln 1/\delta)$ timesteps. □

1.5 Independence of Samples

Much of our entire analysis is grounded on the idea of using samples, in the form of immediate rewards and next-states, to estimate the reward and transition probability distributions for each state-action pair. The main analytical tools we use are large deviation bounds such as the Hoeffding bound (see, for instance, Kearns and Vazirani (1994b)). The Hoeffding bound allows us to quantify a number of samples sufficient to guarantee, with high probability, an accurate estimate of an unknown quantity (for instance, the transition probability to some next-state). However, its use requires *independent* samples. It may appear at first that the immediate reward and next-state observed after taking a fixed action a from a fixed state s is independent of all past immediate rewards and next-states observed. Indeed, due to the Markov property, the immediate reward and next-state are guaranteed to be independent of the entire history of the agent given the current state. However, there is a subtle way in which the samples may not be independent. We now discuss this issue in detail and show that our use of large deviation bounds still hold.

Suppose that we wish to estimate the transition probability of reaching a fixed state s' after experiencing a fixed state-action pair (s,a). We require an ϵ-accurate estimate

[2]Consider two timesteps t_1 and t_2 with $t_1 < t_2 - H$. Technically, the event of escaping from K within H steps on or after timestep t_2 may not be independent of the same escape event on or after timestep t_1. However, the former event is conditionally independent of the later event given the history of the agent up to timestep t_2. Thus, we are able to apply Lemma 1.



An example MDP.

Figure 1.2: An MDP demonstrating the problem with dependent samples.

with probability at least $1 - \delta$, for some predefined values ϵ and δ. Let D be the distribution that produces a 1 if s' is reached after experiencing (s, a) and 0 otherwise. Using the Hoeffding bound we can compute a number m, polynomial in $1/\epsilon$ and $1/\delta$, so that m independent samples of D can be averaged and used as an estimate $\hat{T}(s'|s, a)$. To obtain these samples, we must wait until the agent reaches state s and takes action a at least m times. Unfortunately, the dynamics of the MDP may exist so that the event of reaching state s at least m times provides information about which m samples were obtained from experiencing (s, a). For example, consider the MDP of Figure 1.2. There are 3 states and a single action. Under action 1, state 1 leads to state 2; state 2 leads, with equal probability, to state 1 and state 3; and state 3 leads to itself. Thus, once the agent is in state 3 it can not reach state 2. Suppose we would like to estimate the probability of reaching state 1 from state 2. After our mth experience of state 2, our estimated probability will be either 1 or $(m-1)/m$, both of which are very far from the true probability of $1/2$. This happens because the samples are not independent.

Fortunately, this issue is resolvable, and we can essentially assume that the samples are independent. The key observation is that in the example of Figure 1.2, the probability of reaching state 2 at least m times is also extremely low. It turns out that the probability that an agent (following any policy) observes any fixed m samples of next-states after experiencing (s, a) is at most the probability of observing those same m samples after m independent draws from the transition distribution T. We formalize this now.

Consider a fixed state-action pair (s, a). Upon execution of a learning algorithm on an MDP, we consider the (possibly finite) sequence $O_{s,a} = [O_{s,a}(i)]$, where $O_{s,a}(i)$ is an ordered pair containing the next-state and immediate reward that resulted from the ith experience of (s, a). Let $\mathcal{Q} = [(s[1], r[1]), \ldots, (s[m], r[m])] \in (|S| \times \mathbb{R})^m$ be any finite sequence of m state and reward pairs. Next, we upper bound the probability that the first m elements of $O_{s,a}$ match $\mathcal{Q}$ exactly.

Claim C1: For a fixed state-action pair (s, a), the probability that the sequence $\mathcal{Q}$ is observed by the learning agent (meaning that m experiences of (s, a) do occur and each next-state and immediate reward observed after experiencing (s, a) matches exactly the sequence in $\mathcal{Q}$) is at most the probability that $\mathcal{Q}$ is obtained by a process of drawing m random next-states and rewards from distributions $T(s, a)$ and $\mathcal{R}(s, a)$, respectively. The claim is a consequence of the Markov property.

Proof: (of Claim C1) Let $s(i)$ and $r(i)$ denote the (random) next-state reached and immediate reward received on the ith experience of (s, a), for $i = 1, \ldots, m$ (where $s(i)$ and $r(i)$ take on special values $\emptyset$ and -1, respectively, if no such experience occurs). Let $Z(i)$ denote the event that $s(j) = s[j]$ and $r(j) = r[j]$ for $j = 1, \ldots, i$. Let $W(i)$ denote the event that (s, a) is experienced at least i times. We want to bound the probability that event $Z := Z(m)$ occurs (that the agent observes the sequence $\mathcal{Q}$). We have that

$$\Pr[Z] = \Pr[s(1) = s[1] \wedge r(1) = r[1]] \cdots \Pr[s(m) = s[m] \wedge r(m) = r[m] | Z(m-1)] \quad (1.2)$$

For the ith factor of the right hand side of Equation 1.2, we have that

$$\begin{aligned}
&\Pr[s(i) = s[i] \wedge r(i) = r[i] | Z(i-1)] \\
&= \Pr[s(i) = s[i] \wedge r(i) = r[i] \wedge W(i) | Z(i-1)] \\
&= \Pr[s(i) = s[i] \wedge r(i) = r[i] | W(i) \wedge Z(i-1)] \Pr[W(i) | Z(i-1)] \\
&= \Pr[s(i) = s[i] \wedge r(i) = r[i] | W(i)] \Pr[W(i) | Z(i-1)].
\end{aligned}$$

The first step follows from the fact that $s(i) = s[i]$ and $r(i) = r[i]$ can only occur if (s, a) is experienced for the ith time (event $W(i)$). The last step is a consequence of the Markov property. In words, the probability that the ith experience of (s, a) (if it occurs) will result in next-state $s[i]$ and immediate reward $r[i]$ is conditionally independent of the event $Z(i-1)$ given that (s,a) is experienced at least i times (event $W(i)$). Using the fact that probabilities are at most 1, we have shown that $\Pr[s(i)=s[i] \wedge r(i)=r[i]|Z(i-1)] \leq \Pr[s(i)=s[i] \wedge r(i)=r[i]|W(i)]$ Hence, we have that

$$\Pr[Z] \leq \prod_{i=1}^{m} \Pr[s(i)=s[i] \wedge r(i)=r[i]|W(i)]$$

The right hand-side, $\prod_{i=1}^{m} \Pr[s(i)=s[i] \wedge r(i)=r[i]|W(i)]$ is the probability that $\mathcal{Q}$ is observed after drawing m random next-states and rewards (as from a generative model for MDP M). $\Box$

To summarize, we may assume the samples are independent if we only use this assumption when upper bounding the probability of certain sequences of next-states or rewards. This is valid because, although the samples may not be independent, any upper bound that holds for independent samples also holds for samples obtained in an online manner by the agent.

1.6 Simulation Properties For Discounted MDPs

In this section we investigate the notion of using one MDP as a model or simulator of another MDP. Specifically, suppose that we have two MDPs, M_1 and M_2, with the same state and action space and discount factor. We ask how similar must the transitions and rewards of M_1 and M_2 be in order to guarantee that difference between the value of a fixed policy π in M_1 and its value in M_2 is no larger than some specified threshold ϵ. Although we aren't able to answer the question completely, we do provide a sufficient condition (Lemma 4) that uses L_1 distance to measure the difference between the two transition distributions. Finally, we end with a result (Lemma 5) that measures the difference between a policy's value in the two MDPs when they have equal transitions and rewards most of the time but are otherwise allowed arbitrarily different transitions and rewards.

The following lemma helps develop Lemma 4, a slight improvement over the "Simulation Lemma" of Kearns and Singh (2002) for the discounted case. In the next three lemmas we allow for the possibility of rewards greater than 1 (but still bounded) because they may be of interest outside of the present work. However, we continue to assume, unless otherwise specified, that all rewards fall in the interval $[0, 1]$.

Lemma 2 *(Strehl & Littman, 2007) Let $M_1 = \langle S, A, T_1, R_1, \gamma\rangle$ and $M_2 = \langle S, A, T_2, R_2, \gamma\rangle$ be two MDPs with non-negative rewards bounded by $R_{\max}$. If $|R_1(s, a) - R_2(s, a)| \leq \alpha$ and $||T_1(s, a, \cdot) - T_2(s, a, \cdot)||_1 \leq \beta$ for all states s and actions a, then the following condition holds for all states s, actions a, and stationary, deterministic policies π:*

$$|Q_1^\pi(s, a) - Q_2^\pi(s, a)| \leq \frac{\alpha + \gamma R_{\max}\beta}{(1-\gamma)^2}.$$

Proof: Let $\Delta := \max_{(s,a)\in S\times A} |Q_1^\pi(s, a) - Q_2^\pi(s, a)|$. Let π be a fixed policy and (s, a) be a fixed state-action pair. We overload notation and let R_i denote $R_i(s, a)$, $T_i(s')$ denote $T_i(s'|s, a)$, and $V_i^\pi(s')$ denote $Q_i^\pi(s', \pi(s'))$ for $i = 1, 2$. We have that

$$\begin{aligned}
&|Q_1^\pi(s, a) - Q_2^\pi(s, a)| \\
&= |R_1 + \gamma \sum_{s'\in S} T_1(s')V_1^\pi(s') - R_2 - \gamma \sum_{s'\in S} T_2(s')V_2^\pi(s')| \\
&\leq |R_1 - R_2| + \gamma|\sum_{s'\in S}[T_1(s')V_1^\pi(s') - T_2(s')V_2^\pi(s')]| \\
&\leq \alpha + \gamma|\sum_{s'\in S}[T_1(s')V_1^\pi(s') - T_1(s')V_2^\pi(s') + T_1(s')V_2^\pi(s') - T_2(s')V_2^\pi(s')]| \\
&\leq \alpha + \gamma|\sum_{s'\in S} T_1(s')[V_1^\pi(s') - V_2^\pi(s')]| + \gamma|\sum_{s'\in S}[T_1(s') - T_2(s')]V_2^\pi(s')| \\
&\leq \alpha + \gamma\Delta + \frac{\gamma R_{\max}\beta}{(1-\gamma)}.
\end{aligned}$$

The first step used Bellman's equation.[3] The second and fourth steps used the triangle inequality. In the third step, we added and subtracted the term $T_1(s')V_2^\pi(s')$. In the fifth step we used the bound on the L1 distance between the two transition distributions and the fact that all value functions are bounded by $R_{\max}/(1-\gamma)$. We have shown that $\Delta \leq \alpha + \gamma\Delta + \frac{\gamma R_{\max}\beta}{(1-\gamma)}$. Solving for Δ yields the desired result. □

[3]For an explanation of Bellman's Equation please see Sutton and Barto (1998)

The result of Lemma 2 is not tight. The following stronger result is tight, as demonstrated in Figure 1.3, but harder to prove.

Lemma 3 *Let $M_1 = \langle \mathcal{S}, \mathcal{A}, T_1, R_1, \gamma \rangle$ and $M_2 = \langle \mathcal{S}, \mathcal{A}, T_2, R_2, \gamma \rangle$ be two MDPs with non-negative rewards bounded by $R_{\max}$. If $|R_1(s,a) - R_2(s,a)| \leq \alpha$ and $||T_1(s,a,\cdot) - T_2(s,a,\cdot)||_1 \leq 2\beta$ for all states s and actions a, then the following condition holds for all states s, actions a and stationary, deterministic policies π:*

$$|Q_1^\pi(s,a) - Q_2^\pi(s,a)| \leq \frac{(1-\gamma)\alpha + \gamma\beta R_{\max}}{(1-\gamma)(1-\gamma+\gamma\beta)}.$$

Proof: First, note that any MDP with cycles can be approximated arbitrarily well by an MDP with no cycles. This will allow us to prove the result for MDPs with no cycles. To see this, let M be any MDP with state space $\mathcal{S}$. Consider a sequence of disjoint state spaces $S_1, S_2, \ldots$ such that $|S_i| = S$, and there is some bijective mapping $f_i : \mathcal{S} \to S_i$ for each i. We think of S_i as a copy of $\mathcal{S}$. Now, let M' be an (infinite) MDP with state space $S' = S_1 \cup S_2 \cup \cdots$ and with the same action space $\mathcal{A}$ as M. For $s \in S_i$ and $a \in \mathcal{A}$, let $R(s,a) = R({f_i}^{-1}(s), a)$, where ${f_i}^{-1}$ is the inverse of f_i. Thus, for each i, f_i is a function, mapping the states $\mathcal{S}$ of M to the states S_i of M'. The image of a state s via f_i is a copy of s, and for any action has the same reward function. To define the transition probabilities, let $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$. Then, set $T(f_i(s), a, f_{i+1}(s')) = T(s, a, s')$ in M', for all i. M' has no cycles, yet $V_M^\pi(s) = V_{M'}^\pi(f_i(s))$ for all s and i. Thus, M' is an MDP with no cycles whose value function is the same as M. However, we are interested in a finite state MDP with the same property. Our construction actually leads to a sequence of MDPs $M(1), M(2), \ldots$, where $M(i)$ has state space $S_1 \cup S_2 \cup \cdots S_i$, and with transitions and rewards the same as in M'. It is clear, due to the fact that $\gamma < 1$, that for any ϵ, there is some positive integer i such that $|V_M^\pi(s) - V_{M(i)}^\pi(f_1(s))| \leq \epsilon$ for all s ($f_1(s)$ is the "first" mapping of $\mathcal{S}$ into $M(i)$). Using this mapping the lemma can be proved by showing that the condition holds in MDPs with no cycles. Note that we can define this mapping for the given MDPs M_1 and M_2. In this case, any restriction of the transition and reward functions between M_1 and M_2 also applies to the MDPs $M_1(i)$ and $M_2(i)$, which have no cycles yet approximate M_1 and M_2 arbitrarily well.

We now prove the claim for any two MDPs M_1 and M_2 with no cycles. We also assume that there is only one action. This is a reasonable assumption, as we could remove all actions except those chosen by the policy π, which is assumed to be stationary and deterministic.[4] Due to this assumption, we omit references to the policy π in the following derivation.

Let $v_{\max} = R_{\max}/(1-\gamma)$, which is no less than the value of the optimal policy in either M_1 or M_2. Let s be some state in M_1 (and also in M_2 which has the same state space). Suppose the other states are $s_2, \ldots, s_n$. Let $p_i = T_1(s_i|s,a)$ and $q_i = T_2(s_i|s,a)$. Thus, p_i is the probability of a transition to state s_i from state s after the action a in the MDP M_1, and q_i is the corresponding transition probability in M_2. Since there are no cycles we have that

$$V_{M_1}(s) = R_{M_1}(s) + \gamma \sum_{i=2}^{n} p_i V_{M_1}(s_i)$$

and

$$V_{M_2}(s) = R_{M_2}(s) + \gamma \sum_{i=2}^{n} q_i V_{M_2}(s_i)$$

Without loss of generality, we assume that $V_{M_2}(s) > V_{M_1}(s)$. Since we are interested in bounding the difference $|V_{M_1}(s) - V_{M_2}(s)|$, we can view the problem as one of optimization. Specifically, we seek a solution to

$$\text{maximize } V_{M_2}(s) - V_{M_1}(s) \tag{1.3}$$

subject to

$$\vec{q}, \vec{p} \in \mathcal{P}_{\mathcal{R}^n}, \tag{1.4}$$

$$0 \leq V_{M_1}(s_i), V_{M_2}(s_i) \leq v_{\max} \; i = 1, \ldots, n, \tag{1.5}$$

[4]It is possible to generalize to stochastic policies.

$$0 \leq R_{M_1}(s), R_{M_2}(s) \leq R_{\max}, \tag{1.6}$$

$$-\Delta \leq V_{M_2}(s_i) - V_{M_1}(s_i) \leq \Delta \; i = 1, \ldots, n, \tag{1.7}$$

$$|R_{M_2}(s) - R_{M_1}(s)| \leq \alpha. \tag{1.8}$$

and

$$||\vec{p} - \vec{q}||_1 \leq 2\beta. \tag{1.9}$$

Here, Δ is any bound on the absolute difference between $V_{M_2}(s_i)$ and $V_{M_2}(s_i)$. First, note that $V_{M_2}(s) - V_{M_1}(s)$ under the constraint of Equation 1.8 is maximized when $R_{M_2}(s) - R_{M_1}(s) = \alpha$. Next, assume that $\vec{q}, \vec{p}$ are fixed probability vectors but that $V_{M_1}(s_i)$ and $V_{M_2}(s_i)$ are real variables for $i = 1, \ldots, n$. Consider a fixed $i \in \{2, \ldots, n\}$. The quantity $V_{M_2}(s) - V_{M_1}(s)$ is non-decreasing when $V_{M_2}(s_i)$ is increased and when $V_{M_1}(s_i)$ is decreased. However, the constraint of Equation 1.7 prevents us from setting $V_{M_2}(s_i)$ to the highest possible value ($v_{\max}$) and $V_{M_1}(s_i)$ to the lowest possible value (0). We see that when $q_i \geq p_i$, increasing both $V_{M_1}(s_i)$ and $V_{M_2}(s_i)$ by the same amount provides a net gain, until $V_{M_2}(s_i)$ is maximized. At that point it's best to decrease $V_{M_1}(s_i)$ as much as possible. By a similar argument, when $q_i < p_i$ it's better to decrease $V_{M_1}(s_i)$ as much as possible and then to increase $V_{M_2}(s_i)$ so that Equation 1.7 is satisfied. This argument shows that one solution of the problem specified by Equation 1.3 is of the form:

$$V_{M_2}(s_i) = v_{\max}, \; V_{M_1}(s_i) = v_{\max} - \Delta, \; \text{when } q_i \geq p_i, \tag{1.10}$$

and

$$V_{M_2}(s_i) = \Delta, \; V_{M_1}(s_i) = 0, \; \text{when } q_i < p_i. \tag{1.11}$$

Now, if we are further allowed to change $\vec{p}$ and $\vec{q}$ under the condition that $||\vec{p} - \vec{q}||_1 \leq 2\beta$, maximization yields

$$V_{M_2}(s) - V_{M_1}(s) = \alpha + \gamma\beta v_{\max} + \gamma(1 - \beta)\Delta \tag{1.12}$$

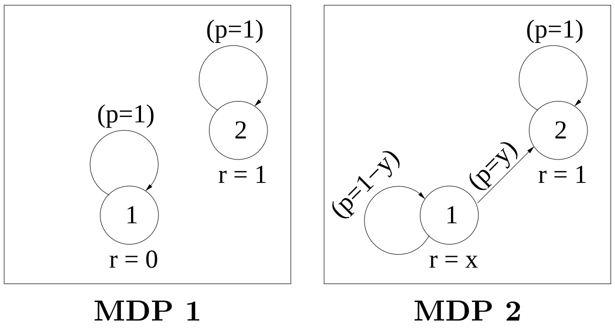


Figure 1.3: An example that illustrates that the bound of Lemma 3 is tight. Each MDP consists of two states and a single action. Each state under each action for the first MDP (on the left) results in a transition back to the originating state (self-loop). From state 1 the reward is always 0 and from state 2 the reward is always 1. In the second MDP, state 1 provides a reward of x and with probability y results in a transition to state 2, which is the same as in the first MDP. Thus, the absolute difference between the value of state 1 in the two MDPs is $\frac{(1-\gamma)x+\gamma y}{(1-\gamma)(1-\gamma+\gamma y)}$. This matches the bound of Lemma 3, where $R_{\max} = 1$, $\alpha = x$, and $\beta = y$.

which holds for any upper bound Δ. Thus, we can find the best such bound (according to Equation 1.12) by replacing the left hand side of Equation 1.12 by Δ. Solving for Δ and using $v_{\max} = R_{\max}/(1-\gamma)$ yields the desired result. $\square$

Algorithms like MBIE act according to an internal model. The following lemma shows that two MDPs with similar transition and reward functions have similar value functions. Thus, an agent need only ensure accuracy in the transitions and rewards of its model to guarantee near-optimal behavior.

Lemma 4 *(Strehl & Littman, 2007) Let $M_1 = \langle S, A, T_1, R_1, \gamma\rangle$ and $M_2 = \langle S, A, T_2, R_2, \gamma\rangle$ be two MDPs with non-negative rewards bounded by $R_{\max}$, which we assume is at least 1. Suppose that $|R_1(s,a) - R_2(s,a)| \leq \alpha$ and $||T_1(s,a,\cdot) - T_2(s,a,\cdot)||_1 \leq \beta$ for all states s and actions a. There exists a constant C, such that for any $0 < \epsilon \leq R_{\max}/(1-\gamma)$ and stationary policy π, if $\alpha = \beta = C\left(\frac{\epsilon(1-\gamma)^2}{R_{\max}}\right)$, then*

$$|Q_1^\pi(s,a) - Q_2^\pi(s,a)| \leq \epsilon. \tag{1.13}$$

Proof: By lemma 2, we have that $|Q_1^\pi(s,a) - Q_2^\pi(s,a)| \leq \frac{\alpha(1+\gamma R_{\max})}{(1-\gamma)^2}$. Thus, it is sufficient to guarantee that $\alpha \leq \frac{\epsilon(1-\gamma)^2}{1+\gamma R_{\max}}$. We choose $C = 1/2$ and by our assumption that $R_{\max} \geq 1$ we have that $\alpha = \frac{\epsilon(1-\gamma)^2}{2R_{\max}} \leq \frac{\epsilon(1-\gamma)^2}{1+\gamma R_{\max}}$. $\square$

The following lemma relates the difference between a policy's value function in two different MDPs, when the transition and reward dynamics for those MDPs are identical on some of the state-action pairs (those in the set K), and arbitrarily different on the other state-action pairs. When the difference between the value of the same policy in these two different MDPs is large, the probability of reaching a state that distinguishes the two MDPs is also large.

Lemma 5 (Generalized Induced Inequality) *(Strehl & Littman, 2007) Let M be an MDP, K a set of state-action pairs, M' an MDP equal to M on K (identical transition and reward functions), π a policy, and H some positive integer. Let A_M be the event that a state-action pair not in K is encountered in a trial generated by starting from state s_1 and following π for H steps in M. Then,*

$$V_M^\pi(s_1, H) \geq V_{M'}^\pi(s_1, H) - (1/(1-\gamma)) \Pr(A_M).$$

Proof: For some fixed *partial path* $p_t = s_1, a_1, r_1 \ldots, s_t, a_t, r_t$, let $P_{t,M}(p_t)$ be the probability p_t resulted from execution of policy π in M starting from state s_1. Let K_t be the set of all paths p_t such that every state-action pair (s_i, a_i) with $1 \leq i \leq t$ appearing in p_t is "known" (in K). Let $r_M(t)$ be the reward received by the agent at time t, and $r_M(p_t, t)$ the reward at time t given that p_t was the partial path generated. Now, we have the following:

$$\begin{aligned}
& E[r_{M'}(t)] - E[r_M(t)] \\
& \quad = \sum_{p_t \in K_t} (P_{t,M'}(p_t) r_{M'}(p_t, t) - P_{t,M}(p_t) r_M(p_t, t)) \\
& \qquad + \sum_{p_t \notin K_t} (P_{t,M'}(p_t) r_{M'}(p_t, t) - P_{t,M}(p_t) r_M(p_t, t)) \\
& \quad = \sum_{p_t \notin K_t} (P_{t,M'}(p_t) r_{M'}(p_t, t) - P_{t,M}(p_t) r_M(p_t, t)) \\
& \quad \leq \sum_{p_t \notin K_t} P_{t,M'}(p_t) r_{M'}(p_t, t) \leq \Pr(A_M).
\end{aligned}$$

The first step in the above derivation involved separating the possible paths in which the agent encounters an unknown state-action from those in which only known state-action pairs are reached. We can then eliminate the first term, because M and M' behave identically on known state-action pairs. The last inequality makes use of the fact that all rewards are at most 1. The result then follows from the fact that $V^{\pi}_{M'}(s_1, H) - V^{\pi}_{M}(s_1, H) = \sum_{t=0}^{H-1} \gamma^t \left(E[r_{M'}(t)] - E[r_M(t)]\right)$. □

The following well-known result allows us to truncate the infinite-horizon value function for a policy to a finite-horizon one.

Lemma 6 *If* $H \geq \frac{1}{1-\gamma} \ln \frac{1}{\epsilon(1-\gamma)}$ *then* $|V^{\pi}(s, H) - V^{\pi}(s)| \leq \epsilon$ *for all policies* π *and states* s*.*

Proof: See Lemma 2 of Kearns and Singh (2002). □

1.7 Conclusion

We have introduced finite-state MDPs and proved some of their mathematical properties. The planning problem is that of acting optimally in a known environment and the learning problem is that of acting near-optimally in an unknown environment. A technical challenge related to the learning problem is the issue of dependent samples. We explained this problem and have shown how to resolve it. In addition, a general framework for proving the efficiency of learning algorithms was provided. In particular, Theorem 1 will be used in the analysis of almost every algorithm in this thesis.

Chapter 2

Model-Based Learning Algorithms

In this chapter we analyze algorithms that are "model based" in the sense that they explicitly compute and maintain an MDP (typically order $S^2 \cdot A$ memory) rather than only a value function (order $S \cdot A$). Model-based algorithms tend to use experience more efficiently but require more computational resources when compared to model-free algorithms.

2.1 Certainty-Equivalence Model-Based Methods

There are several *model-based* algorithms in the literature that maintain an internal MDP as a model for the true MDP that the agent acts in. In this section, we consider using the *maximum liklihood* (also called *Certainty-Equivalence* and *Empirical*) MDP that is computed using the agent's experience. First, we describe the Certainty Equivalence model and then discuss several algorithms that make use of it.

Suppose that the agent has acted for some number of timesteps and consider its experience with respect to some fixed state-action pair (s, a). Let $n(s, a)$ denote the number of times the agent has taken action a from state s. Suppose the agent has observed the following $n(s, a)$ immediate rewards for taking action a from state s: $r[1], r[2], \ldots, r[n(s, a)]$. Then, the empirical mean reward is

$$\hat{R}(s, a) := \frac{1}{n(s, a)} \sum_{i=1}^{n(s,a)} r[i]. \tag{2.1}$$

Let $n(s, a, s')$ denote the number of times the agent has taken action a from state s and immediately transitioned to the state s'. Then, the *empirical transition distribution* is the distribution $\hat{T}(s,a)$ satisfying

$$\hat{T}(s'|s,a) := \frac{n(s,a,s')}{n(s,a)} \text{ for each } s' \in S. \tag{2.2}$$

The Certainty-Equivalence MDP is the MDP with state space $\mathcal{S}$, action space A, transition distribution $\hat{T}(s,a)$ for each (s,a), and deterministic reward function $\hat{R}(s,a)$ for each (s,a). Assuming that the agent will continue to obtain samples for each state-action pair, it is clear that the Certainty-Equivalence model will approach, in the limit, the underlying MDP.

Learning algorithms that make use of the Certainty-Equivalence model generally have the form of Algorithm 1. By choosing a way to initialize the action values (line 2), a scheme for selecting actions (line 11), and a method for updating the action-values (line 17), a concrete Certainty-Equivalence algorithm can be constructed. We now discuss a couple that have been popular.

Algorithm 1 General Certainty-Equivalence Model-based Algorithm

```
0: Inputs: $S$, $A$, $\gamma$
1: for all $(s,a)$ do
2:   Initialize $Q(s,a)$ // action-value estimates
3:   $r(s,a) \leftarrow 0$
4:   $n(s,a) \leftarrow 0$
5:   for all $s' \in S$ do
6:     $n(s,a,s') \leftarrow 0$
7:   end for
8: end for
9: for $t = 1, 2, 3, \cdots$ do
10:  Let $s$ denote the state at time $t$.
11:  Choose some action $a$.
12:  Execute action $a$ from the current state.
13:  Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
     state $s$.
14:  $n(s,a) \leftarrow n(s,a) + 1$
15:  $r(s,a) \leftarrow r(s,a) + r$ // Record immediate reward
16:  $n(s,a,s') \leftarrow n(s,a,s') + 1$ // Record immediate next-state
17:  Update one or more action-values, $Q(s',a')$.
18: end for
```

One of the most basic algorithms we can construct simply uses optimistic initialization, ε-greedy action selection, and value iteration (or some other complete MDP solver) to solve its internal model at each step. Specifically, during each timestep an MDP solver solves the following set of equations to compute its action values:

$$Q(s,a) = \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} Q(s',a') \quad \text{for all } (s,a). \tag{2.3}$$

Solving the system of equations specified above is often a time-consuming task. There are various methods for speeding it up. The *Prioritized Sweeping* algorithm[1] solves the Equations 2.3 approximately by only performing updates that will result in a significant change (Moore & Atkeson, 1993). Computing the state-actions for which a action-value update should be performed requires the knowledge of, for each state, the state-action pairs that might lead to that state (called a predecessor function).

In the Adaptive Real-time Dynamic Programming algorithm of Barto et al. (1995), instead of solving the above equations, only the following single update is performed:

$$Q(s,a) \leftarrow \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} Q(s',a'). \tag{2.4}$$

Here, (s,a) is the most recent state-action pair experienced by the agent.

In Section 4.1, we show that combining optimistic initialization and ε-greedy exploration with the Certainty Equivalence approach fails to produce a PAC-MDP algorithm (Theorem 11).

2.2 E^3

The *Explicit Explore or Exploit* algorithm or E^3 was the first RL algorithm proven to learn near-optimally in polynomial time in general MDPs (Kearns & Singh, 2002). The main intuition behind E^3 is as follows. Let the "known" states be those for which the agent has experienced each action at least m times, for some parameter m. If m is sufficiently large, by solving an MDP model with empirical transitions that provides maximum return for reaching "unknown" states and provides zero reward for all other

[1]The Prioritized Sweeping algorithm also uses the naïve type of exploration and will be discussed in more detail in Section 2.11.

states, a policy is found that is near optimal in the sense of escaping the set of "known" states. The estimated value of this policy is an estimate of the probability of reaching an "unknown" state in T steps (for an appropriately chosen polynomial T). If this probability estimate is very small (less than a parameter *thresh*), then solving another MDP model that uses the empirical transitions and rewards except for the unknown states, which are forced to provide zero return, yields a near-optimal policy.

We see that E^3 will solve two models, one that encourages exploration and one that encourages exploitation. It uses the exploitation policy only when it estimates that the exploration policy does not have a substantial probability of success.

Since E^3 waits to incorporate its experience for state-action pairs until it has experienced them a fixed number of times, it exhibits the naïve type of exploration. Unfortunately, the general PAC-MDP theorem we have developed does not easily adapt to the analysis of E^3 because of E^3's use of two internal models. The general theorem, can, however be applied to the R-MAX algorithm (Brafman & Tennenholtz, 2002), which is similar to E^3 in the sense that it solves an internal model and uses naïve exploration. The main difference between R-MAX and E^3 is that R-MAX solves only a single model and therefore implicitly explores or exploits. The R-MAX and E^3 algorithms were able to achieve roughly the same level of performance in all of our experiments (see Section 5).

2.3 R-MAX

The R-MAX algorithm is similar to the Certainty-Equivalence approaches. In fact, Algorithm 1 is almost general enough to describe R-MAX. R-MAX requires one additional, integer-valued parameter, m. The action selection step is always to choose the action that maximizes the current action value. The update step is to solve the following set of equations:

$$\begin{aligned} Q(s,a) &= \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} Q(s',a'), && \text{if } n(s,a) \geq m, \qquad (2.5) \\ Q(s,a) &= 1/(1-\gamma), && \text{if } n(s,a) < m. \end{aligned}$$

Solving this set of equations is equivalent to computing the optimal action-value function of an MDP, which we call *Model(R-MAX)*. This MDP uses the empirical transition and reward distributions for those state-action pairs that have been experienced by the agent at least m times. The transition distribution for the other state-action pairs is a self loop and the reward for those state-action pairs is always 1, the maximum possible. Another difference between R-MAX and the general Certainty-Equivalence approach is that R-MAX uses *only the first m samples in the empirical model.* That is, the computation of $\hat{R}(s,a)$ and $\hat{T}(s,a)$ in equation 2.5, differs from Section 6.1.3 in that once $n(s,a) = m$, additional samples from $R(s,a)$ and $T(s,a)$ are ignored and not used in the empirical model. To avoid complicated notation, we redefine $n(s,a)$ to be the minimum of the number of times state-action pair (s,a) has been experienced and m. This is consistent with the pseudo-code provided in Algorithm 2.

Any implementation of R-MAX must choose a technique for solving the set of Equations 2.5 and this choice will affect the computational complexity of the algorithm. However, for concreteness[2] we choose *value iteration*, which is a relatively simple and fast MDP solving routine (Puterman, 1994). Actually, for value iteration to solve Equations 2.5 exactly, an infinite number of iterations would be required. One way around this limitation is to note that a very close approximation of Equations 2.5 will yield the same optimal greedy policy. Using this intuition we can argue that the number of iterations needed for value iteration is at most a high-order polynomial in several known parameters of the model, *Model(R-MAX)* (Littman et al., 1995). Another more practical approach is to require a solution to Equations 2.5 that is guaranteed only to produce a near-optimal greedy policy. The following two classic results are useful in quantifying the number of iterations needed.

Proposition 1 *(Corollary 2 from Singh and Yee (1994)) Let $Q'(\cdot,\cdot)$ and $Q^*(\cdot,\cdot)$ be two action-value functions over the same state and action spaces. Suppose that Q^* is the optimal value function of some MDP M. Let π be the greedy policy with respect to Q' and π^* be the greedy policy with respect to Q^*, which is the optimal policy for M. For*

[2]In Section 4.4, we discuss the use of alternative algorithms for solving MDPs.

any $\alpha > 0$ and discount factor $\gamma < 1$, if $\max_{s,a}\{|Q'(s,a) - Q^*(s,a)|\} \leq \alpha(1-\gamma)/2$, *then* $\max_s\{V^{\pi^*}(s) - V^{\pi}(s)\} \leq \alpha$.

Proposition 2 *Let $\beta > 0$ be any real number satisfying $\beta < 1/(1-\gamma)$ and $\gamma < 1$ be any discount factor. Suppose that value iteration is run for $\left\lceil \frac{\ln(1/(\beta(1-\gamma)))}{(1-\gamma)} \right\rceil$ iterations where each initial action-value estimate, $Q(\cdot,\cdot)$, is initialized to some value between* 0 *and* $1/(1-\gamma)$. *Let $Q'(\cdot,\cdot)$ be the resulting action-value estimates. Then, we have that* $\max_{s,a}\{|Q'(s,a) - Q^*(s,a)|\} \leq \beta$.

Proof: Let $Q_i(s,a)$ denote the action-value estimates after the ith iteration of value iteration. The initial values are therefore denoted by $Q_0(\cdot,\cdot)$. Let $\Delta_i := \max_{(s,a)} |Q^*(s,a) - Q_i(s,a)|$. Now, we have that

$$\begin{aligned}
&\Delta_i \\
&\quad = \max_{(s,a)} |(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^*(s')) - (R(s,a) + \gamma \sum_{s'} T(s,a,s')V_{i-1}(s'))| \\
&\quad = \max_{(s,a)} |\gamma \sum_{s'} T(s,a,s')(V^*(s') - V_{i-1}(s'))| \\
&\quad \leq \gamma \Delta_{i-1}.
\end{aligned}$$

Using this bound along with the fact that $\Delta_0 \leq 1/(1-\gamma)$ shows that $\Delta_i \leq \gamma^i/(1-\gamma)$. Setting this value to be at most β and solving for i yields $i \geq \ln(\beta(1-\gamma))/\ln(\gamma)$. We claim that

$$\frac{\ln(\frac{1}{\beta(1-\gamma))})}{(1-\gamma)} \geq \frac{\ln(\beta(1-\gamma))}{\ln(\gamma)}. \tag{2.6}$$

Note that Equation 2.6 is equivalent to the statement $e^{\gamma} - \gamma e \geq 0$, which follows from the the well-known identity $e^x \geq 1 + x$. $\square$

The previous two propositions imply that if we require value iteration to produce an α-optimal policy it is sufficient to run it for $C\frac{\ln(1/(\alpha(1-\gamma)))}{(1-\gamma)}$ iterations, for some constant C. The resulting pseudo-code for R-MAX is given in Algorithm 2. We've added a real-valued parameter, ϵ_1, that specifies the desired closeness to optimality of the policies produced by value iteration. In Section 2.4.2, we show that both m and ϵ_1 can be set as functions of the other input parameters, ϵ, δ, S, A, and γ, in order to make theoretical guarantees about the learning efficiency of R-MAX.

Algorithm 2 R-MAX

```
0: Inputs: $S$, $A$, $\gamma$, $m$, $\epsilon_1$
1: for all $(s, a)$ do
2:   $Q(s, a) \leftarrow 1/(1-\gamma)$ // Action-value estimates
3:   $r(s, a) \leftarrow 0$
4:   $n(s, a) \leftarrow 0$
5:   for all $s' \in S$ do
6:     $n(s, a, s') \leftarrow 0$
7:   end for
8: end for
9: for $t = 1, 2, 3, \cdots$ do
10:   Let $s$ denote the state at time $t$.
11:   Choose action $a := \text{argmax}_{a' \in A} Q(s, a')$.
12:   Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
      state $s$.
13:   if $n(s, a) < m$ then
14:     $n(s, a) \leftarrow n(s, a) + 1$
15:     $r(s, a) \leftarrow r(s, a) + r$ // Record immediate reward
16:     $n(s, a, s') \leftarrow n(s, a, s') + 1$ // Record immediate next-state
17:     if $n(s, a) = m$ then
18:       for $i = 1, 2, 3, \cdots, C\frac{\ln(1/(\epsilon_1(1-\gamma)))}{(1-\gamma)}$ do
19:         for all $(\bar{s}, \bar{a})$ do
20:           if $n(\bar{s}, \bar{a}) \geq m$ then
21:             $Q(\bar{s}, \bar{a}) \leftarrow \hat{R}(\bar{s}, \bar{a}) + \gamma \sum_{s'} \hat{T}(s'|\bar{s}, \bar{a}) \max_{a'} Q(s', a')$.
22:           end if
23:         end for
24:       end for
25:     end if
26:   end if
27: end for
```

There there are many different optimizations available to shorten the number of backups required by value iteration, rather than using the crude upper bound described above. For simplicity, we mention only two important ones, but note that many more appear in the literature. The first is that instead of using a fixed number of iterations, allow the process to stop earlier if possible by examining the maximum change (called the Bellman residual) between two successive approximations of $Q(s, a)$, for any (s, a). It is known that if the maximum change to any action-value estimate in two successive iterations of value iteration is at most $\alpha(1-\gamma)/(2\gamma)$, then the resulting value function yields an α-optimal policy (Williams & Baird, 1993). Using this rule often allows value iteration within the R-MAX algorithm to halt after a number of iterations much less than the upper bound given above. The second optimization is to change the order of the backups. That is, rather than simply loop through each state-action pair during each iteration of value iteration, we update the state-action pairs roughly in the order of how large of a change an update will cause. One way to do so by using the same priorities for each (s, a) as used by the Prioritized Sweeping algorithm (Moore & Atkeson, 1993).

2.4 Analysis of R-MAX

We will analyze R-MAX using the structure from Section 1.4.

2.4.1 Computational Complexity

There is a simple way to change the R-MAX algorithm that has a minimal affect on its behavior and saves greatly on computation. The important observation is that for a fixed state s, the maximum action-value estimate, $\max_a Q(s, a)$ will be $1/(1-\gamma)$ until all actions have been tried m times. Thus, there is no need to run value iteration (lines 17 to 25 in Algorithm 2) until each action has been tried exactly m times. In addition, if there are some actions that have been tried m times and others that have not, the algorithm should choose one of the latter. One method to accomplish this balance is to order each action and try one after another until all are chosen m times. Kearns and Singh (2002) called this behavior "balanced wandering". However, it is not necessary to use balanced wandering; for example, it would be perfectly fine to try the first action m times, the second action m times, and so on. Any deterministic method for breaking ties in line 11 of Algorithm 2 is valid as long as mA experiences of a state-action pair results in all action being chosen m times.

On most timesteps, the R-MAX algorithm performs a constant amount of computation to choose its next action. Only when a state's last action has been tried m times does it solve its internal model. Our version of R-MAX uses value iteration to solve its model. Therefore, the per-timestep computational complexity of R-MAX is

$$\Theta\left(SA(S+\ln(A))\left(\frac{1}{1-\gamma}\right)\ln\frac{1}{\epsilon_1(1-\gamma)}\right). \tag{2.7}$$

This expression is derived using the fact that value iteration performs $O\left(\frac{1}{1-\gamma}\ln\frac{1}{\epsilon_1(1-\gamma)}\right)$ iterations, where each iteration involves SA full Bellman backups (one for each state-action pair). A Bellman backup requires examining all possible $O(S)$ successor states and the update to the priority queue takes time $O(\ln(A))$. Note that R-MAX updates its model at most S times. From this observation we see that the total computation time of R-MAX is $O\left(B+S^2A(S+\ln(A))\left(\frac{1}{1-\gamma}\right)\ln\frac{1}{\epsilon_1(1-\gamma)}\right)$, where B is the number of timesteps for which R-MAX is executed.

2.4.2 Sample Complexity

The main result of this section is to prove the following theorem.

Theorem 2 *(Strehl & Littman, 2006) Suppose that* $0 \le \epsilon < \frac{1}{1-\gamma}$ *and* $0 \le \delta < 1$ *are two real numbers and* $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma\rangle$ *is any MDP. There exists inputs* $m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$ *and* ϵ_1*, satisfying* $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{S+\ln(SA/\delta)}{\epsilon^2(1-\gamma)^4}\right)$ *and* $\frac{1}{\epsilon_1} = O(\frac{1}{\epsilon})$*, such that if R-MAX is executed on* M *with inputs* m *and* ϵ_1*, then the following holds. Let* $\mathcal{A}_t$ *denote R-MAX's policy at time* t *and* s_t *denote the state at time* t*. With probability at least* $1-\delta$*,* $V_M^{\mathcal{A}_t}(s_t) \ge V_M^*(s_t) - \epsilon$ *is true for all but* $O\left(\frac{SA}{\epsilon^3(1-\gamma)^6}\left(S+\ln\frac{SA}{\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$ *timesteps* t*.*

Let $n_t(s,a)$ denote the value of $n(s,a)$ at time t during execution of the algorithm. For R-MAX, we define the "known" state-action pairs K_t, at time t, to be

$$K_t := \{(s,a) \in \mathcal{S} \times \mathsf{A} | n_t(s,a) = m\}, \tag{2.8}$$

which is dependent on the parameter m that is provided as input to the algorithm. In other words, K_t is the set of state-action pairs that have been experienced by the agent at least m times. We call these the "known" state-action pairs, a term taken from Kearns and Singh (2002), because for large enough m, the dynamics, transition and reward, associated with these pairs can be accurately approximated by the agent.

The following event will be used in our proof that R-MAX is PAC-MDP. We will provide a sufficient condition (specifically, L_1-accurate transition and reward functions) to guarantee that the event occurs, with high probability. In words, the condition says that the value of any state s, under any policy, in the empirical known state-action MDP ($\hat{M}_{K_t}$) is ϵ_1-close to its value in the true known state-action MDP (M_{K_t}).

Event A1 *For all stationary policies π, timesteps t and states s during execution of the R-MAX algorithm on some MDP M, $|V^{\pi}_{M_{K_t}}(s) - V^{\pi}_{\hat{M}_{K_t}}(s)| \leq \epsilon_1$.*

Next, we quantify the number of samples needed from both the transition and reward distributions for a state-action pair to compute accurate approximations of both distributions.

Lemma 7 *(Strehl & Littman, 2006) Suppose that $r[1], r[2], \ldots, r[m]$ are m rewards drawn independently from the reward distribution, $\mathcal{R}(s,a)$, for state-action pair (s,a). Let $\hat{R}(s,a)$ be the empirical estimate of $\mathcal{R}(s,a)$, as described in Section 6.1.3. Let δ_R be any positive real number less than 1. Then, with probability at least $1-\delta_R$, we have that $|\hat{R}(s,a) - \mathcal{R}(s,a)| \leq \epsilon^R_{n(s,a)}$, where*

$$\epsilon^R_{n(s,a)} := \sqrt{\frac{\ln(2/\delta_R)}{2m}} \tag{2.9}$$

Proof: This follows directly from Hoeffding's bound (Hoeffding, 1963). □

Lemma 8 *(Strehl & Littman, 2006) Suppose that $\hat{T}(s,a)$ is the empirical transition distribution for state-action pair (s,a) using m samples of next states drawn independently from the true transition distribution $T(s,a)$. Let δ_T be any positive real number less than 1. Then, with probability at least $1-\delta_T$, we have that $||\tilde{T}(s,a) - \hat{T}(s,a)||_1 \leq \epsilon^T_{n(s,a)}$ where*

$$\epsilon^T_{n(s,a)} = \sqrt{\frac{2[\ln(2^S-2) - \ln(\delta_T)]}{m}}. \tag{2.10}$$

Proof: The result follows immediately from an application of Theorem 2.1 of Weissman et al. (2003). □

Lemma 9 *(Strehl & Littman, 2006) There exists a constant C such that if R-MAX with parameters m and ϵ_1 is executed on any MDP $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ and m satisfies*

$$m \geq C \left(\frac{S + \ln{(SA/\delta)}}{{\epsilon_1}^2 (1-\gamma)^4} \right) = \tilde{O} \left(\frac{S}{{\epsilon_1}^2 (1-\gamma)^4} \right),$$

then event $A1$ will occur with probability at least $1 - \delta$.

Proof: Event A1 occurs if R-MAX maintains a close approximation of its known state-action MDP. By Lemma 4, it is sufficient to obtain $C\Big(\epsilon_1(1-\gamma)^2\Big)$-approximate transition and reward functions (where C is a constant), for those state-action pairs in K_t. The transition and reward functions that R-MAX uses are the empirical estimates as described in Section 6.1.3, using only the first m samples (of immediate reward and next-state pairs) for each $(s,a) \in K$. Intuitively, as long as m is large enough, the empirical estimates for these state-action pairs will be accurate, with high probability.[3] Consider a fixed state-action pair (s,a). From Lemma 7, we can guarantee the empirical reward distribution is accurate enough, with probability at least $1-\delta'$, as long as $\sqrt{\frac{\ln(2/\delta')}{2m}} \leq C\Big(\epsilon_1(1-\gamma)^2\Big)$. From Lemma 8, we can guarantee the empirical transition distribution is accurate enough, with probability at least $1-\delta'$, as long as $\sqrt{\frac{2[\ln(2^S-2)-\ln(\delta')]}{m}} \leq C\Big(\epsilon_1(1-\gamma)^2\Big)$. Using these two expressions, we find that it is sufficient to choose m such that

$$m \propto \frac{S + \ln(1/\delta')}{\epsilon_1^2(1-\gamma)^4}. \tag{2.11}$$

[3]There is a minor technicality here. The samples, in the form of immediate rewards and next-states, experienced by an online agent in an MDP are not necessarily independent samples. The reason for this is that the learning environment or the agent could prevent future experiences of state-action pairs based on previously observed outcomes. Nevertheless, all the tail inequality bounds, including the Hoeffding Bound, that hold for independent samples also hold for online samples in MDPs, a fact that is due to the Markov property. There is an extended discussion and formal proof of this fact in Section 1.5.

Thus, as long as m is large enough, we can guarantee that the empirical reward and empirical distribution for a single state-action pair will be sufficiently accurate, with high probability. However, to apply the simulation bounds of Lemma 4, we require accuracy for all state-action pairs. To ensure a total failure probability of δ, we set $\delta' = \delta/(2SA)$ in the above equations and apply the union bound over all state-action pairs. $\square$

Proof: (of Theorem 2). We apply Theorem 1. Let $\epsilon_1 = \epsilon/2$. Assume that Event A1 occurs. Consider some fixed time t. First, we verify Condition 1 of the theorem. We have that $V_t(s) \geq V^*_{\hat{M}_{K_t}}(s) - \epsilon_1 \geq V^*_{M_{K_t}}(s) - 2\epsilon_1 \geq V^*(s) - 2\epsilon_1$. The first inequality follows from the fact that R-MAX computes its action values by computing an ϵ_1-approximate solution of its internal model ($\hat{M}_{K_t}$). The second inequality follows from Event A1 and the third from the fact that M_{K_t} can be obtained from M by removing certain states and replacing them with a maximally rewarding state whose actions are self-loops, an operation that only increases the value of any state. Next, we note that Condition 2 of the theorem follows from Event A1. Finally, observe that the learning complexity, $\zeta(\epsilon, \delta) \leq SAm$, because each time an escape occurs, some $(s, a) \notin K$ is experienced. However, once (s, a) is experienced m times, it becomes part of and never leaves the set K. To guarantee that Event A1 occurs with probability at least $1 - \delta$, we use Lemma 9 to set m. $\square$

2.5 Model-Based Interval Estimation

Interval Estimation (IE) is an advanced technique for handling exploration. It was introduced by (Kaelbling, 1993) for use in the *k-armed bandit problem*, which involves learning in a special class of MDPs. In this section we examine two model-based learning algorithms that use the Interval Estimation (IE) approach to exploration. The first is called Model-based Interval Estimate (MBIE) and the second is called MBIE-EB.

Model-based Interval Estimation maintains the following variables for each state-action pair (s, a) of the MDP M.

- Action-value estimates $\tilde{Q}(s, a)$: These are used by the algorithm to select actions.

They are rough approximations of $Q^*(s, a)$ and are computed by solving the internal MDP model used by MBIE. On timestep t, we denote $\tilde{Q}(s, a)$ by $\tilde{Q}_t(s, a)$. They are initialized optimistically; $\tilde{Q}_0(s, a) = 1/(1 - \gamma)$.

- Reward estimates $\hat{R}(s, a)$: The average reward received for taking action a from state s. On timestep t, we denote $\hat{R}(s, a)$ by $\hat{R}_t(s, a)$.
- Transition estimates $\hat{T}(s, a)$: The maximum liklihood estimate of the true transition distribution $T(s, a)$. On timestep t, we denote $\hat{T}(s, a)$ by $\hat{T}_t(s, a)$.
- Occupancy counts $n(s, a)$: The number of times action a was taken from state s. On timestep t, we denote $n(s, a)$ by $n_t(s, a)$.
- Next-state counts $n(s, a, s')$ for each $s' \in S$: The number of times action a was taken from state s and resulted in next-state s'. On timestep t, we denote $n(s, a, s')$ by $n_t(s, a, s')$.

Besides these, there are several other quantities that MBIE uses.

- Inputs S, A, γ, ϵ, δ. These are provided as inputs to the algorithm before execution time.
- Model size limit m. The maximum number of experiences, per state-action pair (s, a), used to calculate the model parameters $\hat{R}(s, a)$ and $\hat{T}(s, a)$. After the first m experiences of (s, a), the algorithm ignores the data (immediate reward and next state) observed after any additional experiences of (s, a). The precise value of m can be chosen (as in Section 1.3.3) to ensure formal guarantees on the behavior of the algorithm or it can be given as input to the algorithm. In the former case, m will depend on the inputs to the algorithm and especially on ϵ and δ. In the latter case, the value can be chosen using domain knowledge.

When we discuss the behavior of the algorithm at some fixed and specified timestep t, we will often omit the subscript t in the above quantities.

On each timestep, the agent executing MBIE chooses an action greedily with respect to $\tilde{Q}(s, a)$. That is, if s_t is the tth state reached by the agent, then $a_t =$

argmax$\{\tilde{Q}_t(s,a)\}$ is the tth action chosen. If there is a tie, then the algorithm may break the tie arbitrarily.

It is important to note that for each state-action pair (s,a), MBIE uses *only the first m experiences* of (s,a).[4] This means that if on timestep t, action a has been taken from state s more than m times, then the resulting immediate reward and next-state for those experiences are ignored. They are not used to update $\hat{R}(s,a)$ or $\hat{T}(s,a)$. In addition, $n(s,a)$ can be at most m (additional experiences of (s,a) after the mth one do not increase $n(s,a)$). The same is true of $n(s,a,s')$.

We now consider two different but similar algorithms, MBIE and MBIE-EB (short for *Model Based Interval Estimation with Exploration Bonus*). They differ only in the way $\tilde{Q}$ is computed. MBIE builds the upper tail of a confidence interval on the optimal value function of M by simultaneously considering a confidence interval on the entire space of MDPs. The approach taken by MBIE is closely related to that Wiering and Schmidhuber (1998). MBIE-EB uses a simpler form of the confidence intervals it maintains and is closely related to the confidence intervals computed by the Action Elimination algorithm (Even-Dar et al., 2003). It can be viewed as directly computing the upper tail of a confidence interval on the optimal value function of the MDP M. It is an open question which form of MBIE is better in practice. However, we suspect that the simpler version, MBIE-EB, will learn slightly more slowly but use less computation.

2.5.1 MBIE's Model

The action-value estimates $\tilde{Q}(s,a)$ are computed by solving an MDP model. In this section, we describe the model used by MBIE and a method for computing $\tilde{Q}$ from the model. This completes the specification of the MBIE algorithm.

We first provide an intuition behind the model. On timestep t, the algorithm makes

[4]This property of the algorithm is mainly enforced for our analysis. As it is hard to track the value of a changing, non-stationary policy, our analysis requires only a small (polynomial) number of changes to the agent's policy. Since any update to MBIE's model causes a new policy to be computed, we must place a restriction on the number of model updates. This also has the additional benefit of limiting the computational requirements of the algorithm. However, in experiments we have found that setting m to be very large or infinite actually improves the performance of the algorithm when computation is ignored.

use of two confidence intervals for each state-action pair (s, a), one on the mean reward $R(s, a)$ and another on the transition probability distribution $T(s, a)$. These two confidence intervals are then combined into a single confidence interval of MDPs on the underlying MDP M. The model used by MBIE is the MDP within this confidence interval whose optimal policy has the highest value from the current state of the agent (s_t).

Suppose on timestep t, for state-action pair (s, a), the agent has observed the following $n(s, a)$ immediate rewards for taking action a from state s: $r[1], r[2], \ldots, r[n(s, a)]$. Then, the empirical mean reward is

$$\hat{R}(s, a) := \frac{1}{n(s, a)} \sum_{i=1}^{n(s,a)} r[i]. \tag{2.12}$$

The confidence interval for $R(s, a)$ is $CI(R) := (\hat{R}(s, a) - \epsilon^R_{n(s,a)}, \hat{R}(s, a) + \epsilon^R_{n(s,a)})$, where

$$\epsilon^R_{n(s,a)} := \sqrt{\frac{\ln(2/\delta_R)}{2n(s, a)}} \tag{2.13}$$

We refer to $CI(R)$ as the *reward confidence interval* because of the following property. Assuming that the rewards $r[1], r[2], \ldots, r[n(s, a)]$ are independently drawn samples from $\mathcal{R}(s, a)$, we know from the Hoeffding bound that with probability at least $1 - \delta_R$, $R(s, a) \in CI(R)$ will hold.

For a fixed state-action pair (s, a) and timestep t, recall that $T(s, a)$ is the true transition probability distribution for (s, a). It can be viewed as an $|S|$-dimension vector where each component is between 0 and 1. On a fixed timestep t, the *empirical transition vector* is the vector $\hat{T}(s, a)$ with components

$$\hat{T}(s'|s, a) := \frac{n(s, a, s')}{n(s, a)}. \tag{2.14}$$

The right hand side of Equation 2.14 is the proportion of the number of times, among the first m experiences of (s, a) up to timestep t, that the resulting next-state was s'.

The confidence interval for $T(s,a)$ is

$$CI(T) := \{\tilde{T}(s,a) \in \mathcal{P}_S \mid ||\tilde{T}(s,a) - \hat{T}(s,a)||_1 \le \epsilon^T_{n(s,a)}\} \tag{2.15}$$

where $\mathcal{P}_S$ is the set of all $|S|$-dimensional probability vectors (each component is between 0 and 1 and the sum of all components is 1) and

$$\epsilon^T_{n(s,a)} = \sqrt{\frac{2[\ln(2^{|S|}-2) - \ln(\delta_T)]}{m}}. \tag{2.16}$$

It is important to keep in mind that $CI(T)$ is a set (and, in fact, a confidence interval) of probability distributions over the finite state space S.

Let $s[1], s[2], \ldots, s[n(s,a)]$ be the $n(s,a)$ next-states observed by the agent after taking action a from state s and used to compute $\hat{T}(s,a)$. We refer to $CI(T)$ as the *transition confidence interval* because of the following property. Assuming these $n(s,a)$ next-states are independently drawn samples from $T(s,a)$, we have that with probability at least $1-\delta_T$, $T(s,a) \in CI(T)$ will hold (Weissman et al., 2003).

On every timestep t, the main computation of the MBIE is to solve the following set of $|S||A|$ equations for $\tilde{Q}(\cdot,\cdot)$.

$$\tilde{Q}(s,a) = \max_{\tilde{R}(s,a)\in CI(R)} \tilde{R}(s,a) + \max_{\tilde{T}(s,a)\in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s,a) \max_{a'} \tilde{Q}(s',a'). \tag{2.17}$$

Note that this expression effectively combines the uncertainty in the rewards and transitions to provide the MDP model used by MBIE. If a state-action pair (s,a) has never been experienced by the agent, then it's not clear what $CI(R)$ and $CI(T)$ should be. In this case we simply define $\tilde{Q}(s,a) = 1/(1-\gamma)$ for those (s,a) for which $n(s,a) = 0$, a process referred to as *optimistic initialization.*

It's not immediately clear how to quickly solve Equation 2.17 for each (s,a) simultaneously. First, note that the reward term (the first part of the right hand side) is maximized easily. Using this we can simplify Equation 2.17 to $\tilde{Q}(s,a) = (\hat{R}(s,a) + \epsilon^R_{n(s,a)}) + \max_{\tilde{T}(s,a)\in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s,a) \max_{a'} \tilde{Q}(s',a')$. The set of equations now looks a lot like the Bellman equations for solving a single MDP. However, the additional maximization over transition probability distributions complicates the situation and rules out a straight-forward reduction to MDP solving.

We now argue that Equation 2.17 can be solved using value iteration. This problem is strongly related to the problem considered by Givan et al. (2000) that they call finding an "optimistically optimal policy". The main difference is that in their problem, confidence intervals (on the transition distributions) are specified by a bound on the L_∞ rather than L_1 norms. The attainment of our sample-complexity bounds require the use of L_1 norms. The problem of finding a policy that is no worse than the optimal policy for any MDP whose transition function falls in the given confidence interval is interesting but different than our problem and has been well studied (Givan et al., 2000; Nilim & Ghaoui, 2004).

Suppose we start with an arbitrary value function, $Q : S \times A \to \mathbb{R}$. Then, we can obtain a new value function, Q', by using Equation 2.17 as an assignment statement:

$$Q'(s,a) := \max_{\tilde{R}(s,a) \in CI(R)} \tilde{R}(s,a) + \max_{\tilde{T}(s,a) \in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s,a) \max_{a'} Q(s',a'). \tag{2.18}$$

This procedure can be repeated indefinitely to produce a sequence of value functions, which we claim converges to the unique optimal solution to $\tilde{Q}$ of Equation 2.17. This follows from the fact that Equation 2.17 leads to a contraction mapping.

Proposition 3 *(Strehl & Littman, 2007) Let $Q(s,a)$ and $W(s,a)$ be value functions and $Q'(s,a)$ and $W'(s,a)$ be the result of applying Equation 2.18 to Q and W, respectively. The update results in a contraction mapping in max-norm, that is,* $\max_{(s,a)} |Q'(s,a) - W'(s,a)| \le \gamma \max_{(s,a)} |Q(s,a) - W(s,a)|$.

Proof: The argument is nearly identical to the standard convergence proof of value iteration (Puterman, 1994), noting that the maximization over the compact set of probability distributions does not interfere with the contraction property. Let $(s^*, a^*) = \text{argmax}_{(s,a)} |Q'(s,a) - W'(s,a)|$, and let $CI(T)$ denote the transition confidence interval (see Equation 2.15) for the state-action pair (s^*, a^*). Let $V(s) := \max_a Q(s,a)$ and

$X(s) := \max_a W(s,a)$ for all states s. Let $T_1 := \operatorname{argmax}_{\tilde{T}(s^*,a^*) \in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s^*,a^*)V(s')$ and $T_2 := \operatorname{argmax}_{\tilde{T}(s^*,a^*) \in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s^*,a^*)X(s')$. Without loss of generality, assume that $Q'(s^*,a^*) \geq W'(s^*,a^*)$. Then,

$$\begin{aligned}
&\max_{s,a} |Q'(s,a) - W'(s,a)| = Q'(s^*,a^*) - W'(s^*,a^*) \\
=\;& \max_{\tilde{T}(s^*,a^*) \in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s^*,a^*)V(s') - \max_{\bar{T}(s^*,a^*) \in CI(T)} \gamma \sum_{s'} \bar{T}(s'|s^*,a^*)X(s') \\
=\;& \gamma \sum_{s'} T_1(s'|s^*,a^*)V(s') - \gamma \sum_{s'} T_2(s'|s^*,a^*)X(s') \\
\leq\;& \gamma \sum_{s'} T_1(s'|s^*,a^*)V(s') - \gamma \sum_{s'} T_1(s'|s^*,a^*)X(s') \\
\leq\;& \gamma \max_{s} |V(s) - X(s)| \leq \gamma \max_{(s,a)} |Q(s,a) - W(s,a)|.
\end{aligned}$$

The first inequality results from the definition of T_2. Specifically, T_2 is the probability distribution in $CI(T)$ that maximizes the expected value of the next state under $X(\cdot)$, so it is greater than the expected value with respect to the distribution T_1. The rest of the steps follow from basic algebraic manipulation. $\square$

To implement value iteration based on Equation 2.17, we need to solve the following computational problem. Maximize

$$M_V = \sum_{s'} \tilde{T}(s'|s,a)V(s'),$$

over all probability distributions, $\tilde{T}(\cdot|s,a)$, subject to the constraint $||\tilde{T}(\cdot|s,a) - \hat{T}(\cdot|s,a)||_1 \leq \varepsilon$, where $\varepsilon = \epsilon^T_{n(s,a)}$ as in Equation 2.16.

The following procedure can be used to find the maximum value. Let $\tilde{T}(\cdot|s,a) = \hat{T}(\cdot|s,a)$ to start. Let $s^* = \operatorname{argmax}_s V(s)$ and increment $\tilde{T}(s^*|s,a)$ by $\varepsilon/2$. At this point, $\tilde{T}(\cdot|s,a)$ is not a probability distribution, since it sums to $1 + \varepsilon/2$; however, its L_1 distance to $\hat{T}(\cdot|s,a)$ is $\varepsilon/2$. We need to remove exactly $\varepsilon/2$ weight from $\tilde{T}(\cdot|s,a)$. The weight is removed iteratively, taking the maximum amount possible from the state $s^- = \operatorname{argmin}_{s'|\tilde{T}(s'|s,a)>0} V(s')$, since this decreases M_V the least.

Proposition 4 *(Strehl & Littman, 2007) The procedure just described maximizes* M_V.

Proof: First, the weight on s^* is at most $\hat{T}(s^*|s,a) + \varepsilon/2$. If it's larger, $\tilde{T}(\cdot|s,a)$ violates the L_1 constraint. Let $\tilde{T}(s'|s,a) - \hat{T}(s'|s,a)$ be called the *residual* of state s'. If $||\tilde{T}(\cdot|s,a) - \hat{T}(\cdot|s,a)||_1 = \varepsilon$, then the sum of the positive residuals is $\varepsilon/2$ and the sum of the negative residuals is $-\varepsilon/2$. For two states s_1 and s_2, if $V(s_1) > V(s_2)$, then M_V is increased by moving positive residual from s_2 to s_1 or negative residual from s_1 to s_2. Therefore, putting all positive residual on s^* and moving all the negative residual toward states with smallest $V(s)$ values maximizes M_V among all distributions $\tilde{T}(\cdot|s,a)$ with the given L_1 constraint. $\square$

2.5.2 MBIE-EB's Model

MBIE-EB simply solves the following set of equations to compute its action-value estimates, $\tilde{Q}(\cdot,\cdot)$:

$$\tilde{Q}(s,a) = \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} \tilde{Q}(s',a') + \frac{\beta}{\sqrt{n(s,a)}}, \tag{2.19}$$

where $\hat{R}$ and $\hat{T}$ form the empirical model based on the first m experiences for each state-action pair as described above and β is a constant provided as input to the algorithm. Equation 2.19 can be solved efficiently using any technique for solving an MDP.

Pseudo-code is provided in Algorithm 3.

2.6 Analysis of MBIE

In this section we provide a formal analysis of both versions of the MBIE algorithm. Throughout this section we measure complexity assuming individual numbers require unit storage and can be manipulated arithmetically in unit time. Removing this assumption increases space and computational complexities by logarithmic factors.

2.6.1 Computation Complexity of MBIE

The computational complexity of MBIE depends on the implementation. Our complexity bounds reflect one possible implementation of the algorithm. In this section we first

Algorithm 3 MBIE-EB

```
0: Inputs: $S$, $A$, $\gamma$, $m$, $\beta$, $\epsilon_1$
1: for all $(s, a)$ do
2:   $Q(s, a) \leftarrow 1/(1-\gamma)$ // Action-value estimates
3:   $r(s, a) \leftarrow 0$
4:   $n(s, a) \leftarrow 0$
5:   for all $s' \in S$ do
6:     $n(s, a, s') \leftarrow 0$
7:   end for
8: end for
9: for $t = 1, 2, 3, \cdots$ do
10:  Let $s$ denote the state at time $t$.
11:  Choose action $a := \operatorname{argmax}_{a' \in A} Q(s, a')$.
12:  Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
     state $s$.
13:  if $n(s, a) < m$ then
14:    $n(s, a) \leftarrow n(s, a) + 1$
15:    $r(s, a) \leftarrow r(s, a) + r$ // Record immediate reward
16:    $n(s, a, s') \leftarrow n(s, a, s') + 1$ // Record immediate next-state
17:    for $i = 1, 2, 3, \cdots, O\left(\frac{\ln(1/\epsilon_1(1-\gamma))}{(1-\gamma)}\right)$ do
18:      for all $(\bar{s}, \bar{a})$ do
19:        $Q(\bar{s}, \bar{a}) \leftarrow \hat{R}(\bar{s}, \bar{a}) + \gamma \sum_{s'} \hat{T}(s'|\bar{s}, \bar{a}) \max_{a'} Q(s', a') + \frac{\beta}{\sqrt{(n(s,a)}}$.
20:      end for
21:    end for
22:  end if
23: end for
```

discuss the worst case per-timestep computational complexity of the algorithm. Then we discuss the worst case total computational complexity. The per-timestep complexity is interesting from a learning viewpoint, where the longest time between two actions may be more important than the total time taken over all actions.

Consider some timestep t during execution of MBIE. Suppose that s is the tth state reached, a is the tth action chosen, r is the resulting immediate reward, and s' is the next-state.

At this point, the first computation of the algorithm is to update its internal model. That is, it updates the variables $\hat{R}(s, a)$, $\hat{T}(s, a)$, $n(s, a)$, and $n(s, a, s')$, which can be done in constant time. The next computation is to recalculate $\tilde{Q}(s, a)$ for each (s, a) using value iteration as described in Section 2.5.1. Each iteration of value iteration involves a loop through each state-action pair of the MDP. Within this loop, the algorithm performs the optimization described in Section 2.5.1. This optimization sorts the observed reachable next-states by their estimated value, which takes $\Theta(|S| \log_2(|S|)) = \Theta(|S| \ln(|S|))$ computation time. It also requires knowledge of the state s^* whose current value estimate is maximum. To obtain this quickly, our implementation of MBIE includes a global priority queue[5] of size $|S||A|$ that holds all the action-value estimates for each state. Finally, the algorithm must choose an action. To facilitate this choice, we also maintain a local priority queue for each state s that stores the values, $\tilde{Q}(s, a)$, for each action a.

To summarize, if $\mathcal{N}$ is the maximum number of iterations of value iteration used to solve Equation 2.17 during execution of MBIE on any timestep, then the worst case per-timestep computational complexity of MBIE is $O(|S||A|\mathcal{N}(|S| \ln(|S|) + |S| + \ln(|S||A|) + \ln(|A|))) = O(|S||A|\mathcal{N}(|S| \ln(|S|) + \ln(|S||A|)))$. The $|S||A|\mathcal{N}$ factor is the number of updates performed during value iteration. The $|S| \ln(|S|)$ term is for sorting the next-states by value. The $|S| + \ln(|S||A|)$ term follows from the cost of computing the new action-value estimate for a single state-action pair and updating the global priority queue. The $\ln(|A|)$ term follows from the fact that the new action-value estimate needs to be stored in a local priority queue for the current state.

For simplicity in our analysis and description of the algorithm, we have assumed that MBIE solves its model (specified by Equations 2.17) exactly. However, it is easy to modify the analysis to show that we only need value iteration to produce a $C\epsilon$-optimal policy (rather than an optimal one), for some constant C. Thus, we can easily bound $\mathcal{N}$ by a polynomial, in particular, letting $\mathcal{N} = O\left(\ln\left(\frac{1}{\epsilon(1-\gamma)}\right)/(1-\gamma)\right)$ is sufficient, by the results of Section 2.3. In practice, we have found that it works well to stop once successive iterations don't result in a change to any action-value estimate $\tilde{Q}(s, a)$ that is greater than some small threshold. The space complexity of MBIE is $O(m|S||A|)$, which can be achieved by maintaining the counts $n(s, a, s')$ only for those s' that have been reached by the agent during the first m experiences of (s, a).

[5]In Strehl and Littman (2005) an approximation that removes the necessity of maintaining this priority queue was introduced. The approximation is simply to use the value $1/(1-\gamma)$ as a bound on the value of any state.

We have shown that the per-step computation time of our implementation of MBIE is $O(|S||A|\mathcal{N}(|S|\ln(|S|) + \ln(|S||A|)))$, where $\mathcal{N}$ is an upper bound on the number of iterations used by the version of value iteration described in Section 4.4. Note that on each timestep, MBIE performs more than a constant amount of computation only if it updates its internal model. If no model update is required, than MBIE simply needs to choose an action by solving $\text{argmax}_{a\in A} \tilde{Q}(s,a)$, where s is the current state. This can be done in constant time by using a priority queue as discussed above. Since MBIE can update its model at most $|S||A|m$ times, the total computation complexity is

$$O(B + |S|^2|A|^2 m\mathcal{N}(|S|\ln(|S|) + \ln(|S||A|))) \tag{2.20}$$

where B is the number of timesteps for which MBIE is executed.

2.6.2 Computational Complexity of MBIE-EB

As we have done for MBIE, we first consider the worst case per-timestep computational complexity of the algorithm. In fact, the only difference, in terms of computation time, between the two algorithms is that MBIE-EB does not need to perform the optimization step of Section 2.5.1. Instead it can perform traditional value iteration on its model. This incurs a per-step computational cost of $O(|S||A|\mathcal{N}(|S| + \ln(|A|)))$, where $\mathcal{N}$ is the maximum number of iterations of value iteration used to solve Equation 2.19 during execution of MBIE-EB on any timestep. After each action-value update, a cost of $\ln(|A|)$ is incurred for storing the new action-value estimate in a local priority queue for the current state. Like MBIE, the space complexity of MBIE is $O(m|S||A|)$.

We have shown that the per-step computation time of our implementation of MBIE-EB is $O(|S||A|\mathcal{N}(|S| + \ln(|A|)))$. This leads to a total computation complexity of

$$O(B + |S|^2|A|^2 m\mathcal{N}(|S| + \ln(|A|))) \tag{2.21}$$

where B is the number of timesteps for which MBIE-EB is executed.

2.6.3 Sample Complexity of MBIE

In this section, we study the sample complexity of exploration of MBIE. Our main result is summarized by the following theorem.

Theorem 3 *(Strehl & Littman, 2007) Suppose that ϵ and δ are two real numbers between 0 and 1 and $M = \langle S, A, T, \mathcal{R}, \gamma \rangle$ is any MDP. There exists inputs $\delta_R = \delta_T = \delta/(2|S||A|m)$ and $m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$, satisfying $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{|S|}{\epsilon^2(1-\gamma)^4} + \frac{1}{\epsilon^2(1-\gamma)^4} \ln \frac{|S||A|}{\epsilon(1-\gamma)\delta}\right)$, such that if MBIE is executed on M, then the following holds. Let $\mathcal{A}_t$ denote MBIE's policy at time t and s_t denote the state at time t. With probability at least $1 - \delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ is true for all but $O\left(\frac{|S||A|}{\epsilon^3(1-\gamma)^6}\left(|S| + \ln \frac{|S||A|}{\epsilon(1-\gamma)\delta}\right) \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right)$ timesteps t.*

At the beginning of a run, every state-action (s, a) pair is said to be *unknown.* At any step of the algorithm, the set of *known* state-action pairs K is defined to be those (s, a) experienced at least m times (Kearns & Singh, 2002). For large enough m, any $(s, a) \in K$ will be accurately modeled.

An overview of the sample-complexity analysis is as follows. At each timestep, MBIE follows the optimal policy of its model $\tilde{M}$. Lemmas 4, 5, and 10 show that the value of MBIE's policy in its model is very close to its true value as long as the probability of reaching an unknown state-action pair is low. By Lemma 11, the estimated value of its policy is at least as large, with high probability, as the true optimal value function. Thus, MBIE chooses its actions based on a policy that is either nearly optimal or one with a high probability of encountering an unknown (s, a). However, the number of times a given (s, a) can be experienced before it becomes known is shown to be no more than polynomial in the relevant quantities. Therefore, the agent will act nearly optimally on all but a bounded number of timesteps—it has polynomial sample complexity.

As the MBIE agent gathers experience, it is continuously updating and solving its model of the world according to Equation 2.17. Let CI be any confidence interval computed by MBIE. We say that CI is *admissible* if it contains the mean of the distribution that produced the samples for which CI was computed from. For our following analysis, we require that all confidence intervals—reward and transition—be admissible for all state-action pairs over every time-step, with high probability. Note that MBIE will compute at most $|S||A|m$ confidence intervals (m confidence intervals per state-action pair). Thus, to ensure admissible confidence intervals with probability at least $1-\delta$, it is sufficient to set $\delta_T = \delta_R = \delta/(2|S||A|m)$ and apply the union bound.

The next lemma quantifies the amount of experience, for each state-action pair, required by MBIE to accurately model the dynamics of the environment.

Lemma 10 *(Strehl & Littman, 2007) Suppose that $\delta_T = \delta_R = \delta/(2|S||A|m)$ and that all confidence intervals computed by MBIE are admissible. Then, for any $\tau > 0$, there exists an $m = O\left(\frac{|S|}{\tau^2} + \frac{1}{\tau^2}\ln\frac{|S||A|}{\tau\delta}\right)$ such that $|\tilde{R}(s,a) - R(s,a)| \leq \tau$ and $||\tilde{T}(s,a,\cdot) - T(s,a,\cdot)||_1 \leq \tau$ holds for all state-action pairs (s,a) that have been experienced at least m times.*

Proof: Consider a fixed state-action pair (s,a) and some fixed timestep t during execution of MBIE. From Equation 2.13, the size of the reward confidence interval associated with (s,a) is $2\sqrt{\frac{\ln(2/\delta_R)}{2n(s,a)}}$. From Equation 2.16, the size of the corresponding transition confidence interval is $2\sqrt{\frac{2[\ln(2^{|S|}-2)-\ln(\delta_T)]}{n(s,a)}}$. Once (s,a) has been experienced m times by the agent, these intervals become fixed, $n(s,a) = m$, and the two expressions above become $2\sqrt{\frac{\ln(2/\delta_R)}{2m}}$ and $2\sqrt{\frac{2[\ln(2^{|S|}-2)-\ln(\delta_T)]}{m}}$. Using the assumption that both confidence intervals are admissible, we have that

$$m \geq \max\left\{\frac{8[\ln(2^{|S|}-2)-\ln(\delta_T)]}{\tau^2}, \frac{2\ln(2/\delta_R)}{\tau^2}\right\} \tag{2.22}$$

is sufficient to guarantee that $|\tilde{R}(s,a) - R(s,a)| \leq \tau$ and $||\tilde{T}(s,a,\cdot) - T(s,a,\cdot)||_1 \leq \tau$ when $n(s,a) = m$. By substitution of $\delta/(2|S||A|m)$ for δ_R and δ_T, the right hand side of Equation 2.22, it can be rewritten as $\max\left\{\frac{8[\ln(2^{|S|}-2)+\ln(2|S||A|m/\delta))]}{\tau^2}, \frac{2\ln(4|S||A|m/\delta)}{\tau^2}\right\}$. It is a well known fact that for any constant $C > 0$, $n \geq 2C\ln(C)$ implies $n \geq C\ln(n)$. Using this, it is clear that m can be chosen large enough so that Equation 2.22 holds, but small enough so that $m = O\left(\frac{|S|}{\tau^2} + \frac{1}{\tau^2}\ln\frac{|S||A|}{\epsilon\delta}\right)$. $\square$

The MBIE algorithm exhibits "optimism in the face of uncertainty". This notion is captured formally by the following lemma. Specifically, we show that the expected return of acting in the MBIE agent's model is at least as large as the expected return of acting in the underlying environment, with high probability.

Lemma 11 *(Strehl & Littman, 2007) Suppose that all confidence intervals computed by MBIE are admissible. Then, for any state s and action a, the condition $\tilde{Q}(s,a) \geq Q^*(s,a)$ is satisfied during any iteration of MBIE.*

Proof: At each step of the learning problem, MBIE solves the MDP $\tilde{M}$. We prove the claim by induction on the number of steps of value iteration.[6] For the base case, assume that the Q values are initialized to $1/(1-\gamma) \geq V^*(s)$, for all s. Now, for the induction, suppose that the claim holds for the current value function $\tilde{Q}(s,a)$.

MBIE computes two confidence intervals. $CI(R)$ is an interval of real numbers of the form $(\hat{R}(s,a) - \epsilon^R_{n(s,a)}, \hat{R}(s,a) + \epsilon^R_{n(s,a)})$. $CI(T)$ is the set of probability distributions $T'(\cdot|s,a)$ of the form $||\hat{T}(\cdot|s,a) - T'(\cdot|s,a)||_1 \leq \epsilon^T_{n(s,a)}$. By assumption, we have that $R(s,a) \in CI(R)$ and $T(s,a) \in CI(T)$.

The term $\tilde{Q}(s',a')$ on the right-hand side of Equation 2.17 is the result of the previous iteration and is used to compute the new Q-value $\tilde{Q}(s,a)$ on the left-hand side of the equation. By our confidence-interval assumption, we have $\tilde{R}(s,a) \geq R(s,a)$ and

$$\begin{aligned} & \max_{\tilde{T}(\cdot|s,a)\in CI(T)} \gamma \sum_{s'} \tilde{T}(s'|s,a) \max_{a'} \tilde{Q}(s',a') \\ \geq \; & \gamma \sum_{s'} T(s'|s,a) \max_{a'} \tilde{Q}(s',a') \\ \geq \; & \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q^*(s',a'). \end{aligned}$$

The first step follows from the assumption that $T(\cdot|s,a) \in CI(T)$ and the second from the induction assumption. $\square$

We are now ready to prove Theorem 3.

[6] We assume here that value iteration is halted after a finite number of iterations.

Proof: (of Theorem 3). We shall assume that MBIE's internal model is solved completely during every timestep (an optimal policy is obtained). The generalization to the case where the model is solved approximately is straight forward and is similar to the corresponding result for R-MAX (see the proof of Theorem 2). We also assume that all confidence intervals computed by MBIE are admissible, an assumption that holds with probability at least $1 - \delta/2$, by the union bound and our choice of δ_R and δ_T.

We apply Theorem 1. Consider some fixed timestep t. First, note that Condition 1 of Theorem 1 follows immediately from lemma 11.

Second, we verify Condition 2 of Theorem 1. Define K to be the set of "known" state-action pairs, that is, the set of all state-action pairs (s, a) that have been experienced at least m times ($n(s, a) \geq m$); we will provide the value of m shortly. Recall that the MBIE agent $\mathcal{A}_t$ chooses its next action by following an optimal policy $\tilde{\pi}$ of MBIE's internal model $\tilde{M}$ at time t. Let M_K be the MDP that is equal to M on K (meaning equal reward and transition distributions for $(s, a) \in K$) and equal to $\tilde{M}$ on $S \times A - K$. Using the terminology of Theorem 1, M_K is the "known state-action MDP" with respect the MBIE's state-action values $\tilde{Q}(\cdot, \cdot)$ and the set K. Using Lemma 4, Lemma 10 (with $\tau = C\epsilon_1(1 - \gamma)^2$, where C is the constant specified in Lemma 4), and our assumption of admissible confidence intervals, it follows that we can choose m with $m = O\left(\frac{|S|}{\epsilon^2(1-\gamma)^4} + \frac{1}{\epsilon^2(1-\gamma)^4} \ln \frac{|S||A|}{\epsilon(1-\gamma)\delta}\right)$ so that

$$|V^{\tilde{\pi}}_{M_K}(s) - V^{\tilde{\pi}}_{\tilde{M}}(s)| \leq \epsilon \tag{2.23}$$

holds for all s. This equation implies that Condition 2 of Theorem 1 holds (using the notation of Theorem 1, $\pi_t = \tilde{\pi}$ and $V_t = V^{\tilde{\pi}}_{\tilde{M}}$).

Next, we verify Condition 3 of Theorem 1. An update to some state-action value can occur only when some "unknown" state-action pair (s, a) is visited. A fixed state-action pair (s, a) can be visited at most m times until it becomes "known". Once it's known, it can never become unknown. Thus, we get a crude bound of S^2A^2m for $\zeta(\cdot, \cdot)$. However, we can do better. Note that without changing the proof of Theorem 1 we can redefine $\zeta(\cdot,\cdot)$ to be an upper bound on the number of timesteps that either an escape or an update occurs. Thus, we can bound $\zeta(\cdot,\cdot)$ by SAm, which yields the desired result upon substitution into Theorem 1. $\square$

2.6.4 Sample Complexity of MBIE-EB

We start off by showing that MBIE-EB also exhibits "optimism in the face of uncertainty".

Lemma 12 *(Strehl & Littman, 2007) Suppose MBIE-EB is executed on any MDP M with $\beta = (1/(1-\gamma))\sqrt{\ln(2|S||A|m/\delta)/2}$. Then, with probability at least $1-\delta/2$, $\tilde{Q}(s,a) \geq Q^*(s,a)$ will always hold.*

Proof: First, for some state-action pair (s,a), consider the first $k \leq m$ experiences of (s,a) by the agent (timesteps for which action a was taken from state s). Let $X_1, X_2, \ldots, X_k$ be the k random variables defined by:

$$X_i := r_i + \gamma V^*(s_i) \tag{2.24}$$

where r_i is the ith reward received and s_i is the ith state visited after taking action a from state s. Note that $E[X_i] = Q^*(s,a)$ and that $0 \leq X_i \leq 1/(1-\gamma)$ for all $i = 1,\ldots,k$. Assuming that the X_i are independently and identically distributed (see Appendix A), we can apply the Hoeffding bound to arrive at

$$\Pr\left[E[X_1] - (1/k)\sum_{i=1}^{k} X_i \geq \beta/\sqrt{k}\right] \leq e^{-2\beta^2(1-\gamma)^2}.$$

The value of β specified by the lemma guarantees that the right-hand side above is $\delta/(2|S||A|m)$. Note that $(1/k)\sum_{i=1}^{k} X_i = \hat{R}(s,a) + \gamma\sum_{s'}\hat{T}(s'|s,a)V^*(s')$. Once (s,a) has been experienced m times, $\hat{R}$ and $\hat{T}$ cannot change again. Therefore, by the union bound we have that

$$\hat{R}(s,a) + \gamma\sum_{s'}\hat{T}(s'|s,a)V^*(s') - Q^*(s,a) \geq -\beta/\sqrt{k} \tag{2.25}$$

holds for all timesteps t and all state-action pairs (s,a) with probability at least $1-\delta/2$, where $k = n_t(s,a)$.

Fix a timestep t. Recall that MBIE-EB uses value iteration to solve the following set of equations:

$$\tilde{Q}(s,a) = \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} \tilde{Q}(s',a') + \frac{\beta}{\sqrt{n(s,a)}}, \tag{2.26}$$

The proof is by induction on the number of steps of value iteration.[7] Let $\tilde{Q}^{(i)}(s,a)$ denote the ith iterate of value iteration for (s,a), and let $\tilde{V}^{(i)}(s) = \max_a \tilde{Q}^{(i)}(s,a)$. For the base case, by optimistic initialization we have that $\tilde{Q}^{(0)} = 1/(1-\gamma) \geq Q^*(s,a)$ for all state-action pairs (s,a). Now, for the induction, suppose that the claim holds for the current value function $\tilde{Q}^{(i)}(s,a)$. We have that

$$\begin{aligned}
&\tilde{Q}^{(i+1)}(s,a) \\
&= \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} \tilde{Q}^{(i)}(s',a') + \frac{\beta}{\sqrt{n(s,a)}} \\
&= \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \tilde{V}^{(i)}(s') + \frac{\beta}{\sqrt{n(s,a)}} \\
&\geq \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) V^{*}(s') + \frac{\beta}{\sqrt{n(s,a)}} \\
&\geq Q^*(s,a).
\end{aligned}$$

The first two steps follow from definitions. In the last two steps we used the inductive hypothesis and Equation 2.25, respectively. □

We now show that MBIE-EB is also PAC-MDP.

Theorem 4 *(Strehl & Littman, 2007) Suppose that ϵ and δ are two real numbers between 0 and 1 and $M = \langle S, A, T, \mathcal{R}, \gamma\rangle$ is any MDP. There exists an input $m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$, satisfying $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{|S|}{\epsilon^2(1-\gamma)^4} + \frac{1}{\epsilon^2(1-\gamma)^4} \ln \frac{|S||A|}{\epsilon(1-\gamma)\delta}\right)$, and $\beta = (1/(1-\gamma))\sqrt{\ln(2|S||A|m/\delta)/2}$ such that if MBIE-EB is executed on MDP M, then the following holds. Let $\mathcal{A}_t$ denote MBIE-EB's policy at time t and s_t denote the state at time t. With probability at least*

[7]We assume here that value iteration is halted after a finite number of iterations.

$1-\delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ is true for all but $O\left(\frac{|S||A|}{\epsilon^3(1-\gamma)^6}\left(|S| + \ln\frac{|S||A|}{\epsilon(1-\gamma)\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$ timesteps t.

Proof: We shall assume that MBIE-EB's internal model is solved completely during every timestep (an optimal policy is obtained). The generalization to the case where the model is solved approximately is straight forward and is similar to the corresponding result for R-MAX (see the proof of Theorem 2). We also assume that the reward confidence intervals computed by MBIE-EB are admissible.

We apply Theorem 1. Consider some fixed timestep t. First, note that Condition 1 of Theorem 1 follows immediately from lemma 12.

First, we argue that after each (s, a) has been experienced a polynomial number of times, m, the empirical model learned from those experiences, $\hat{R}(s, a)$ and $\hat{T}(s, a)$, will be sufficiently close to the true distributions, $\mathcal{R}(s, a)$ and $T(s, a)$, so that the value of any policy in an MDP model based on $\hat{R}$ and $\hat{T}$ is no more than ϵ away from its value in the MDP based on R and T (but otherwise the same), with high probability. It follows from Lemma 4 that it is sufficient to require that $||\hat{T}(s, a) - T(s, a)||_1 \leq \tau$ and $|\hat{R}(s, a) - R(s, a)| \leq \tau$ for $\tau = C\epsilon(1-\gamma)^2$, where C is the constant specified in Lemma 4. Using the form of reward and confidence intervals used by MBIE-EB with $\delta_R = \delta_T = \delta/(2|S||A|m)$, it follows that

$$m \geq C_1\left(\frac{|S|}{\epsilon^2(1-\gamma)^4} + \frac{1}{\epsilon^2(1-\gamma)^4}\ln\frac{|S||A|}{\epsilon(1-\gamma)\delta}\right) \tag{2.27}$$

for some positive constant C_1 is sufficient for these two conditions to hold, with probability at least $1 - \delta/2$. A formal proof of this is almost identical to the proof of Lemma 10, keeping in mind that while MBIE uses the upper tail of the transition and reward confidence intervals (see Section 2.5.1), the empirical model uses the center of these confidence intervals.[8]

Second, we verify Condition 2 of Theorem 1. Let s_t be the current state of the agent. Define K to be the set of all state-action pairs (s, a) that have been experienced

[8]This difference amounts to a factor of 2 in the analysis.

at least m times ($n(s,a) = m$); we will provide the value of m shortly. We call K the set of *known state-action pairs*. Recall that the MBIE-EB agent $\mathcal{A}_t$ chooses its next action by following an optimal policy $\tilde{\pi}$ of its internal model $\tilde{M}$ at time t. Let M' be the MDP that is equal to M on K (meaning equal reward and transition distributions for $(s,a) \in K$) and equal to $\tilde{M}$ on $S \times A - K$. Let $\hat{M}'$ be the MDP that is equal to $\hat{M}$ on K and equal to $\tilde{M}$ on $S \times A - K$. From our choice of m above, we have that

$$|V^{\tilde{\pi}}_{M'}(s) - V^{\tilde{\pi}}_{\hat{M}'}(s)| \leq \epsilon \tag{2.28}$$

holds for all s, with probability at least $1 - \delta/2$. Also, note that $\tilde{M}$ is identical to the MDP $\hat{M}'$ except that some state-action pairs (precisely, those in K) have an additional reward bonus of $\beta/\sqrt{m}$. Thus, we have that $V^{\tilde{\pi}}_{\tilde{M}}(s) \leq V^{\tilde{\pi}}_{\hat{M}'}(s) + \beta/(\sqrt{m}(1-\gamma))$. For our analysis, we require that

$$\beta/(\sqrt{m}(1-\gamma)) \leq \epsilon. \tag{2.29}$$

We define

$$\beta = (1/(1-\gamma))\sqrt{\ln(2|S||A|m/\delta)/2}. \tag{2.30}$$

It's not hard to show that we can make m large enough so that Equations 2.27 and 2.29 hold, yet small enough so that $m = O\left(\frac{|S|}{\epsilon^2(1-\gamma)^4} + \frac{1}{\epsilon^2(1-\gamma)^4}\ln\frac{|S||A|}{\epsilon(1-\gamma)\delta}\right)$. To see this fact, note that when substitution the value of β from Equation 2.30 into Equation 2.29 and simplifying, we arrive at

$$m \geq \frac{\ln(|S||A|m/\delta)/2}{(1-\gamma)^4\epsilon^2}.$$

The rest follows from further basic algebraic manipulation. Given that Equation 2.29 holds, we have that

$$V^{\tilde{\pi}}_{\tilde{M}}(s) \leq V^{\tilde{\pi}}_{\hat{M}'}(s) + \epsilon. \tag{2.31}$$

Combining Equation 2.28 with Equation 2.31 and using the notation of Theorem 1 we have proved that $V_t(s) - V_{M_{K_t}}(s) \leq 2\epsilon$ for all s. Thus, dividing ϵ by 2 in the previous derivation allows us to satisfy Condition 2 of Theorem 1.

Next, we verify Condition 3 of Theorem 1. An update to some state-action value can occur only when some "unknown" state-action pair (s, a) is visited. A fixed state-action pair (s, a) can be visited at most m times until it becomes "known". Once it's known, it can never become unknown. Thus, we get a crude bound of S^2A^2m for $\zeta(\cdot, \cdot)$. However, we can do better. Note that without changing the proof of Theorem 1 we can redefine $\zeta(\cdot, \cdot)$ to be an upper bound on the number of timesteps that either an escape or an update occurs. Thus, we can bound $\zeta(\cdot, \cdot)$ by SAm, which yields the desired result upon substitution into Theorem 1. □

One of the main drawbacks of the R-MAX and MBIE algorithms is their high per-step computational complexity (due to solution of an internal MDP model). An agent that must act in real time would most likely require an algorithm that chooses actions faster. We have already noted that a complete solution of the internal model is not necessary for an efficient algorithm. Interestingly, even a close approximation of the solution is unnecessary. Next, we discuss the RTDP-RMAX and RTDP-IE algorithms, whose analysis verifies that fast (per-step) algorithms can still take advantage of maintaining an internal model (Strehl et al., 2006a). Both algorithms "incrementally" solve their internal models using ideas related to the Real-time Dynamic Programming (RTDP) algorithm of Barto et al. (1995).

2.7 RTDP-RMAX

The RTDP-RMAX algorithm uses the same internal MDP model as the R-MAX algorithm. Rather than solving the model at each step, which can be computationally costly, RTDP-RMAX only partially solves its model. Specifically, instead of running value iteration to compute a near-optimal policy, RTDP-RMAX performs at most a single Bellman backup on the most recent state-action pair experienced by the agent. In fact, it performs an update only if the new state-action estimate is significantly smaller than the previous estimate. The amount that the estimate must decrease to warrant an update is quantified by an additional parameter, ϵ_1.

In addition to the standard inputs, the RTDP-RMAX algorithm requires two additional parameters, a positive integer parameter m and a positive real number ϵ_1. Later, in the analysis of Section 2.8, we provide a formal procedure for choosing m and ϵ_1, but for now we consider them as free parameters. The value of m, in particular, controls the exploration behavior[9] of the agent (larger values of m encourage greater exploration while smaller values encourage greedier behavior).

Suppose that a is the tth action of the agent and is taken from state s with $n_t(s,a) \geq m$ being satisfied. Recall that $V_t(s')$, for any state s', is shorthand for $\max_{a'} Q_t(s', a')$. The following update then occurs:

$$Q_{t+1}(s,a) = \hat{R}(s,a) + \gamma \sum_{s' \in S} \hat{T}(s'|s,a) V_t(s'), \tag{2.32}$$

where $\hat{R}$ and $\hat{T}$ are the empirical reward and transition functions (the maximum likelihood estimates) computed using using only the first m experiences (next states and immediate rewards) for (s,a). We allow the update, Equation 2.32, 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:

$$Q_t(s,a) - \left(\hat{R}_t(s,a) + \gamma \sum_{s' \in S} \hat{T}(s'|s,a) V_t(s') \right) \geq \epsilon_1. \tag{2.33}$$

Otherwise, no change is made and $Q_{t+1}(s,a) = Q_t(s,a)$. For all other $(s', a') \neq (s,a)$, no update occurs. That is, $Q_{t+1}(s',a') = Q_t(s',a')$. Similarly if $n_t(s,a) < m$, then no update occurs. Pseudo-code for the algorithm is provided in Algorithm 4.

To summarize, the RTDP-RMAX algorithm chooses, at each step, to either update a single state-action pair or not. If the last state occupied by the agent, under the last action chosen by the agent, has been experienced at least m times, then the action-value estimate for that state-action pair is updated as long as it results in a significant decrease to the action-value estimate. The update is a standard full Bellman backup, as in value iteration, where the empirical transition probabilities and empirical reward

[9]The parameter ϵ_1 can also affect the exploration behavior of the agent.

functions are used (in place of the true transition and reward functions, which are unknown by the agent). This approach differs from R-MAX in that one step of value iteration (VI) is taken from one state instead of running VI to completion over all states.

Algorithm 4 RTDP-RMAX

```
 0: Inputs: $S$, $A$, $\gamma$, $m$, $\epsilon_1$
 1: for all $(s,a)$ do
 2:   $Q(s,a) \leftarrow 1/(1-\gamma)$
 3:   $r(s,a) \leftarrow 0$
 4:   $n(s,a) \leftarrow 0$
 5:   for all $s' \in S$ do
 6:     $n(s,a,s') \leftarrow 0$
 7:   end for
 8: end for
 9: for $t = 1,2,3,\cdots$ do
10:   Let $s$ denote the state at time $t$.
11:   Choose action $a := \mathrm{argmax}_{a' \in A} Q(s,a')$.
12:   Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
      state $s$.
13:   if $n(s,a) < m$ then
14:     $n(s,a) \leftarrow n(s,a) + 1$
15:     $r(s,a) \leftarrow r(s,a) + r$ // Record immediate reward
16:     $n(s,a,s') \leftarrow n(s,a,s') + 1$ // Record immediate next-state
17:     if $n(s,a) = m$ then
18:       $q \leftarrow \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} Q(s',a')$
19:       if $Q(s,a) - q \geq \epsilon_1$ then
20:         $Q(s,a) \leftarrow q$.
21:       end if
22:     end if
23:   end if
24: end for
```

2.8 Analysis of RTDP-RMAX

2.8.1 Computational Complexity

During each timestep, RTDP-RMAX performs no more than a single Bellman backup. Therefore, its worst case computational complexity per timestep is

$$\theta(S + \ln(A)), \tag{2.34}$$

where the log term is due to updating the priority queue that holds the action-value estimates for each state.

As implemented in Algorithm 4, the algorithm will perform an unbounded number of Bellman backups leading to a total computational complexity of

$$O\left((S+\ln(A))B\right), \tag{2.35}$$

where B is the number of timesteps for which RTDP-RMAX is executed. However, on some timesteps the computation of a Bellman backup is unnecessary. The reason is that the computation is redundant (meaning that it will not result in a change to the current action-value estimate) unless there has been a change to the model or some other action-value estimate. The algorithm can easily check for such conditions in constant time by using logic very similar to the $LEARN$ flags of Delayed Q-learning (see Section 3.2). It can be shown that this modification reduces the total computational complexity to

$$O\left(B+\frac{SA\ln(A)}{\epsilon_1(1-\gamma)}+\frac{S^3A^2}{(1-\gamma)}\right). \tag{2.36}$$

A rough outline of the derivation of Equation 2.36 follows. RTDP-RMAX performs at most $SA/(\epsilon_1(1-\gamma))$ successful updates to its action-value estimates because the estimates begin at $1/(1-\gamma)$, cannot fall below zero, and decrease by at least ϵ_1 on every successful update. The action-value estimate for a fixed state-action pair does not change until it has been experienced at least m times. In addition, the potential update value (q in line 18 of Algorithm 4) will not change over time unless some action-value estimate changes. Therefore, the expression need only be computed at most $1+SA/(\epsilon_1(1-\gamma))$ times. To summarize, there will be at most $SA/(\epsilon_1(1-\gamma))$ updates (line 20 of Algorithm 4), each of which takes $O(\ln(A))$ computation time, and at most $1+SA(\frac{SA}{1-\gamma})$ Bellman backups, each of which takes $O(S)$ computation time. Taking the weighted sum of these terms yields the asymptotic bound of Equation 2.36. The first term of B comes from the constant amount of computation the algorithm uses on every timestep, even when no backup is performed.

2.8.2 Sample Complexity

It is possible to prove that RTDP-RMAX is an efficient PAC-MDP algorithm with a sample complexity bound that is asymptotically identical to the one we were able to prove for RMAX. The main insight is that Theorem 1 can be applied by defining K to be the set of all state-action pairs (s, a) such that

1. The pair has been experienced at least m times: $n(s, a) \geq m$, and
2. The Bellman residual is small:

$$Q_t(s,a) - \left(\hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a)V_t(s')\right) \leq \epsilon_1. \tag{2.37}$$

This novel definition extends the standard definition (as used in the analysis of E^3 and RMAX), which associates K with the state-action pairs that have been tried m times, to allow incremental updates to propagate value information more gradually. Specifically, a state-action pair is unknown if its Bellman error is high or the rewards and transitions are not adequately estimated. We omit the complete analysis due to its similarity to the analysis in Section 2.10.

Theorem 5 *(Strehl et al., 2006a) Suppose that ϵ and δ are two real numbers between 0 and 1 and $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ is any MDP. There exists inputs $m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$, and ϵ_1, satisfying $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{S + \ln(SA/\delta)}{\epsilon^2(1-\gamma)^4}\right)$ and $\frac{1}{\epsilon_1} = O(\frac{1}{\epsilon(1-\gamma)})$, such that if RTDP-RMAX is executed on M with inputs m and ϵ_1, then the following holds. Let $\mathcal{A}_t$ denote RTDP-RMAX's policy at time t and s_t denote the state at time t. With probability at least $1 - \delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ is true for all but $O\left(\frac{SA}{\epsilon^3(1-\gamma)^6}\left(S + \ln\frac{SA}{\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$ timesteps t.*

2.9 RTDP-IE

As RTDP-RMAX uses the same model as R-MAX and solves it incrementally with one Bellman backup during each step, RTDP-IE incrementally solves the same model as MBIE-EB.

The RTDP-IE (short for "real-time dynamic programming with interval estimation") algorithm also requires two additional real-valued parameters, β (larger values encourage greater exploration while smaller values encourage greedier behavior) and ϵ_1, that can be chosen to provide a formal learning-time guarantee, as we show in Section 2.10.

Suppose that a is the tth action by the agent and is taken from state s. Consider the following update:

$$Q_{t+1}(s,a) = \hat{R}_t(s,a) + \gamma \sum_{s' \in S} \hat{T}_t(s'|s,a) V_t(s') + \frac{\beta}{\sqrt{n(s,a)}}, \tag{2.38}$$

where $\hat{R}_t$ and $\hat{T}_t$ are the empirical reward and transition functions at time t computed using only the first m experiences (next states and immediate rewards) for (s,a). The update specified by Equation 2.38 is allowed to occur only if it would result in a decrease of at least ϵ_1. Note that we do not allow $n(s,a)$ to increase beyond m even if (s,a) is experienced more than m times.

Like RTDP-RMAX, the update is a standard full Bellman backup, where the empirical transition probabilities and reward functions are used, plus an "exploration bonus" proportional to β that decreases at a rate inversely proportional to the square root of the number of times the state-action pair has been experienced. Thus, a higher bonus is provided to state-action pairs that have not been tried as often. Complete pseudo-code is provided in Algorithm 5.

2.10 Analysis of RTDP-IE

2.10.1 Computational Complexity

During each timestep, RTDP-IE performs no more computation, asymptotically, than required for a single Bellman backup. Therefore, its worst case computational complexity per timestep is

$$\theta(S + \ln(A)), \tag{2.39}$$

Algorithm 5 RTDP-IE

```
0: Inputs: $S$, $A$, $\gamma$, $m$, $\epsilon_1$, $\beta$
1: for all $(s, a)$ do
2:   $Q(s, a) \leftarrow 1/(1-\gamma)$ // Action-value estimates
3:   $r(s, a) \leftarrow 0$
4:   $n(s, a) \leftarrow 0$
5:   for all $s' \in S$ do
6:     $n(s, a, s') \leftarrow 0$
7:   end for
8: end for
9: for $t = 1, 2, 3, \cdots$ do
10:   Let $s$ denote the state at time $t$.
11:   Choose action $a := \text{argmax}_{a' \in A} Q(s, a')$.
12:   Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
      state $s$.
13:   if $n(s, a) < m$ then
14:     $n(s, a) \leftarrow n(s, a) + 1$
15:     $r(s, a) \leftarrow r(s, a) + r$ // Record immediate reward
16:     $n(s, a, s') \leftarrow n(s, a, s') + 1$ // Record immediate next-state
17:   end if
18:   $q \leftarrow \hat{R}(s, a) + \gamma \sum_{s'} \hat{T}(s'|s, a) \max_{a'} Q(s', a') + \beta/\sqrt{n(s, a)}$
19:   if $Q(s, a) - q \geq \epsilon_1$ then
20:     $Q(s, a) \leftarrow q$.
21:   end if
22: end for
```

where the log term is due to updating the priority queue that holds the action-value estimates for each state.

As implemented in Algorithm 5, the algorithm may perform the equivalent of an unbounded number of Bellman backups, leading to a total computational complexity of

$$O\left((S + \ln(A))B\right), \tag{2.40}$$

where B is the number of timesteps for which RTDP-IE is executed. However, on some timesteps the computation of a Bellman backup is unnecessary; it is redundant unless there has been a change to the model or some action-value estimate. The algorithm can easily check for such conditions in constant time by using logic very similar to the *LEARN* flags of Delayed Q-learning (see Section 3.2). It can be shown that this modification reduces the total computational complexity to

$$O\left(B + S^2Am + \frac{SA\ln(A)}{\epsilon_1(1-\gamma)} + \frac{S^3A^2}{(1-\gamma)}\right). \tag{2.41}$$

The derivation of this bound is similar to the one for RTDP-RMAX from Section 2.8. The difference is that RTDP-IE will potentially perform m additional Bellman backup computations (in line 18 in Algorithm 5) for each state-action pair corresponding to the first m visits to each state-action pair, and adding an additional term of S^2Am to the total complexity.

2.10.2 Sample Complexity

The analysis of RTDP-IE actually follows that of RTDP-RMAX very closely. As in Section 2.8, we modify the algorithm slightly.

In the version of RTDP-IE that we analyze, an update is performed as specified by Equation 2.38 only if it would result in a decrease of at least ϵ_1. In addition, the empirical transitions and rewards, $\hat{T}_t$ and $\hat{R}_t$, respectively, are computed using only the first m experiences (m is an additional parameter supplied to the algorithm) for (s, a). We let $\hat{T}(\cdot|s, a)$ and $\hat{R}(s, a)$ denote the model for (s, a) learned after m experiences of (s, a). Furthermore, once (s, a) has been experienced m times, the bonus of $\beta/\sqrt{n(s, a, t)}$ in Equation 2.38 is replaced by ϵ_1.

For RTDP-IE, we use the following set of "known" state-actions: During timestep t of execution of RTDP-IE, we define K_t to be the set of all state-action pairs (s, a), with $n(s, a, t) \geq m$ such that:

$$Q_t(s, a) - \left(\hat{R}(s, a) + \gamma \sum_{s'} \hat{T}(s'|s, a)V_t(s')\right) \leq 2\epsilon_1 \tag{2.42}$$

Proposition 5 *(Strehl et al., 2006a) For any* $0 < \delta < 1$*, if* $m \geq \beta^2/{\epsilon_1}^2$ *and* $\beta \geq (1/(1-\gamma))\sqrt{\ln(SAm/\delta)/2}$*, then during execution of RTDP-IE, with probability at least* $1 - \delta/2$*,* $Q_t(s, a) \geq Q^*(s, a)$ *holds for all state-action pairs* (s, a) *and timesteps* t*.*

Proof: First, for some state-action pair (s, a), consider the first $k \leq m$ experiences of (s, a) by the agent (timesteps for which action a was taken from state s). Let $X_1, X_2, \ldots, X_k$ be the k independently and identically distributed random variables defined by:

$$X_i := r_i + \gamma V^*(s_i) \tag{2.43}$$

where r_i is the ith reward received and s_i is the ith state visited after taking action a from state s. Note that $E[X_i] = Q^*(s, a)$ and that $0 \leq X_i \leq 1/(1-\gamma)$ for all $i = 1, \ldots, k$. By the additive form of the Hoeffding bound we have that

$$\Pr\left[E[X_1] - (1/k)\sum_{i=1}^{k} X_i \geq \beta/\sqrt{k}\right] \leq e^{-2\beta^2(1-\gamma)^2}.$$

The value of β specified by the lemma guarantees that the right-hand side above is most $\delta/(SAm)$. Note that $(1/k)\sum_{i=1}^{k} X_i = \hat{R}_t(s, a) + \gamma \sum_{s'} \hat{T}_t(s, a, s')V^*(s')$. Once (s, a) has been experienced m times, $\hat{R}_t$ and $\hat{T}_t$ cannot change again. Therefore, by the union bound we have that $\hat{R}_t(s, a) + \gamma \sum_{s'} \hat{T}_t(s, a, s')V^*(s') - Q^*(s, a) \geq -\beta/\sqrt{k}$ holds for all timesteps t and all state-action pairs (s, a) with probability at least $1 - \delta$, where $k = \min\{n(s, a, t), m\}$.

The proof is by induction on 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 $Q_t(s, a) \geq Q^*(s, a)$ and $V_t(s) \geq V^*(s)$ for all (s, a). Suppose s is the $(t+1)$st state reached and a the action taken at time t. Let n denote $n(s, a, t+1)$. If no update occurs then we are done because the action-values will still be optimistic. Otherwise, if $n < m$, we have that $Q_{t+1}(s, a) = \hat{R}_{t+1}(s, a) + \gamma \sum_{s' \in S} \hat{T}_{t+1}(s, a, s')V_t(s') + \frac{\beta}{\sqrt{n}} \geq \hat{R}_{t+1}(s, a) + \gamma \sum_{s' \in S} \hat{T}_{t+1}(s, a, s')V^*(s') + \frac{\beta}{\sqrt{n}} \geq Q^*(s, a)$, by the above argument. If $n \geq m$, then $Q_{t+1}(s, a) = \hat{R}_{t+1}(s, a) + \gamma \sum_{s' \in S} \hat{T}_{t+1}(s, a, s')V_t(s') + \epsilon_1 \geq \hat{R}_{t+1}(s, a) + \gamma \sum_{s' \in S} \hat{T}_{t+1}(s, a, s')V_t(s') + \frac{\beta}{m} \geq Q^*(s, a)$, by the above argument. We have used that fact that if $m \geq \beta^2/{\epsilon_1}^2$, then $\epsilon_1 \geq \frac{\beta}{\sqrt{m}}$. Finally, we note that the two preconditions of the lemma, $m \geq \beta^2/{\epsilon_1}^2$ and $\beta \geq (1/(1-\gamma))\sqrt{\ln(SAm/\delta)/2}$, are satisfied for $\beta = (1/(1-\gamma))\sqrt{\ln(SAm/\delta)/2}$ and

$m = O(\frac{1}{\epsilon^2(1-\gamma)^4} \ln \frac{SA}{\epsilon^2(1-\gamma)^4\delta})$. To see this note that $m \geq \beta^2/{\epsilon_1}^2$ is equivalent to

$$m \geq \frac{\ln(SAm/\delta)}{2(1-\gamma)^2{\epsilon_1}^2}.$$

Using the fact that

- for any positive numbers α and β, if $m \geq 2\alpha \ln(\alpha\beta)$, then $m \geq \alpha \ln(\beta m)$

yields the desired result. □

The following condition will be needed for our proof that RTDP-IE is PAC-MDP. We will provide a sufficient condition (specifically, L_1-accurate transition and reward functions) to guarantee that it holds. In words, the first part of the condition says that the value of the greedy policy (with respect to the agent's action values) in the empirical known state-action MDP ($\hat{M}_{K_t}$) is ϵ_1-close to its value in the true known state-action MDP (M_{K_t}). The second part says that the optimal value function of the last and final model learned by RTDP-RMAX is not too far from the what it would be if the correct transitions and rewards were used for those state-actions tried at least m times.

Assumption A1 *For all timesteps t and states s, we have that $|V^{\pi_t}_{M_{K_t}}(s) - V^{\pi_t}_{\hat{M}_{K_t}}(s)| \leq \epsilon_1$ where π_t is the greedy policy (with respect to the agent's action-value estimates) at time t.* Also, $|V^*_{M_{\tilde{K}}}(s) - V^*_{\hat{M}_{\tilde{K}}}(s)| \leq \epsilon_1$ where $\tilde{K} = \{(s,a) \mid \exists u \in \mathbb{Z}^+ \ s.t. \ n(s,a,u) \geq m\}$.

Theorem 6 *(Strehl et al., 2006a) Suppose that $0 \leq \epsilon < \frac{1}{1-\gamma}$ and $0 \leq \delta < 1$ are two real numbers and $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ is any MDP. There exists inputs $m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$ and ϵ_1 satisfying $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{S+\ln(SA/\epsilon\delta(1-\gamma))}{\epsilon^2(1-\gamma)^4}\right)$ and $\frac{1}{\epsilon_1} = O(\frac{1}{\epsilon(1-\gamma)})$, such that if RTDP-IE is executed on M, then the following holds. Let $\mathcal{A}_t$ denote RTDP-IE's policy at time t and s_t denote the state at time t. With probability at least $1-\delta$, $V^{\mathcal{A}_t}_M(s_t) \geq V^*_M(s_t) - \epsilon$ is true for all but $O\left(\frac{SA}{\epsilon^3(1-\gamma)^6}\left(S + \ln \frac{SA}{\epsilon\delta(1-\gamma)}\right) \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)}\right)$ timesteps t.*

Proof: We apply Proposition 1. We let $\beta := 1/(1-\gamma)\sqrt{\ln(SAm/\delta)/2}$ and $m = \tilde{O}\left(\frac{S}{\epsilon^2(1-\gamma)^4}\right)$ so that $m \geq \beta^2/{\epsilon_1}^2$ and so that Assumption A1 holds, with probability at least $1-\delta/2$. During timestep t of execution of RTDP-IE, we define K_t to be the set of all state-action pairs (s, a), with $n(s, a, t) \geq m$ such that:

$$Q_t(s,a) - \left(\hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) V_t(s') \right) \leq 2\epsilon_1 \tag{2.44}$$

Every update of a state-action pair decreases its action-value by at least ϵ_1. Since its action-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. Also, once a state-action pair has been experienced m times, an experience of that state-action pair can cause event A_K to happen if and only if an update also occurs (we have defined K_t to make this true). Thus, there will be at most $SAm + SA/(\epsilon_1(1-\gamma))$ timesteps t such that either A_K occurs or an update is performed to any (s, a). By Proposition 5, we have that the optimism precondition is satisfied.

Finally, we claim that, by Assumption A1, $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq 2\epsilon_1/(1-\gamma)$ always holds. To verify this claim, note that $V^{\pi_t}_{\hat{M}_{K_t}}$ is the solution to the following set of equations:

$$V^{\pi_t}_{\hat{M}_{K_t}}(s) = \hat{R}(s, \pi_t(s)) + \gamma \sum_{s' \in S} \hat{T}(s'|s, \pi_t(s)) V^{\pi_t}_{\hat{M}_{K_t}}(s'), \quad \text{if } (s, \pi_t(s)) \in K_t,$$

$$V^{\pi_t}_{\hat{M}_{K_t}}(s) = Q_t(s, \pi_t(s)), \quad \text{if } (s, \pi_t(s)) \notin K_t.$$

The vector V_t is the solution to a similar set of equations except with some additional positive reward terms, each bounded by $2\epsilon_1$ (see Equation 2.44). It follows that $V_t(s) - V^{\pi_t}_{\hat{M}_{K_t}}(s) \leq 2\epsilon_1/(1-\gamma)$. Combining this fact with Assumption A1 yields $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq 3\epsilon_1/(1-\gamma)$. Thus, by letting $\epsilon_1 = \epsilon(1-\gamma)/3$, we satisfy $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \epsilon$, as desired (to fulfill Condition (2) of Proposition **??**). Ignoring log factors, this analysis leads to a total sample complexity bound of

$$\tilde{O}\left(\left(SAm_1 + \frac{SA}{\epsilon(1-\gamma)^2} \right) \frac{1}{\epsilon(1-\gamma)^2} \right). \tag{2.45}$$

□

Before moving on, we first summarize the algorithms that have been discussed so far. The R-MAX and E^3 algorithms successfully utilize the naïve exploration approach to yield PAC-MDP algorithms. They work by maintaining and solving an internal MDP model. The solution of this model is computationally costly. The RTDP-RMAX algorithm uses the same model as the R-MAX algorithm, but avoids the costly step of solving the model. Instead, during each step, it updates an approximate solution of the model by focusing on the state most recently occupied and the action taken by the agent. The algorithm is PAC-MDP and has better per-step computational complexity when compared to R-MAX.[10] The MBIE-EB algorithm makes use of the IE approach to exploration rather than the naïve approach, which allows the algorithm to more effectively incorporate the agent's experience into its internal model. Like R-MAX, the MBIE-EB algorithm involves solution of an internal model. To avoid this step, the RTDP-IE algorithm solves the model incrementally, in the same fashion as the RTDP-RMAX algorithm. Both RTDP-RMAX and RTDP-IE have a worst-case per-step computational complexity of $\theta(S+\ln(A))$, which can be very large when the MDP has many states. In Chapter 3 we review an algorithm due to Strehl et al. (2006c) that is PAC-MDP with a per-step computational complexity of only $\theta(\ln(A))$.

2.11 Prioritized Sweeping

Traditional model-based algorithms, like R-MAX, require $O(S^2A)$, computation, per timestep, in the worst case. To improve on this bound, RTDP-RMAX and RTDP-IE perform at most a single full Bellman-style backup (Equations 2.32 and 2.38) per timestep. This operation has a cost of $O(S + \ln(A))$. A further reduction of computational complexity to $O(\ln(A))$ per timestep is provided by the Delayed Q-learning algorithm (see Section 3.2). Suppose we want an algorithm to run on a real-time system, such as a robot. There is a certain amount of time between actions that can be used for computation. It may be that there is enough time for several full Bellman backups, but not enough to solve an entire internal model. Then, we can imagine an

[10]Interestingly, the total amortized computational complexity bounds for RTDP-RMAX are slightly worse than those for R-MAX. This represents an overhead cost for delaying the solution of the model after it has been updated.

algorithm that uses a model, like R-MAX, but rather than solving the model entirely at each step, it performs as many Bellman backups as possible within the externally imposed time constraint. This is, in fact, the approach taken by the Prioritized Sweeping algorithm (Moore & Atkeson, 1993).

Prioritized Sweeping performs the Bellman backups by first ordering them by *priority*, a number that roughly indicates how much we expect the update caused by the backup to affect the model. The analyses of RTDP-RMAX and RTDP-IE require a single Bellman backup per timestep, but don't fail if other backups are executed. Specifically, we can augment the RTDP-RMAX and RTDP-IE algorithms by performing any additional updates of state-action pairs, (s, a), as long as RTDP-RMAX or RTDP-IE, respectively, would perform those updates normally if (s, a) were experienced by the agent. The original version of Prioritized Sweeping used the naïve strategy for exploration (in exactly the same way as the R-MAX algorithm). Thus, using the analysis of Section 2.8, we have that Prioritized Sweeping is a PAC-MDP algorithm.[11] Similarly, using the analysis of Section 2.10, we have that a version of Prioritized Sweeping that uses the IE approach to exploration is also PAC-MDP. These results are summarized in Section 2.11.1. We believe that this observation is an important new result, because Prioritized Sweeping is one of the few RL algorithms that can adaptively adjust its computation time based on external real-time constraints. As the algorithm is given more computation, it solves more of its model, and learns faster. However, even if the algorithm is given minimal computation, we still have formal guarantees on its performance.

2.11.1 Analysis of Prioritized Sweeping

We considered two versions of Prioritized Sweeping, one that uses naïve-like exploration and one that uses IE-like exploration. The fact that both algorithms are PAC-MDP follows directly from the fact that both RTDP-RMAX and RTDP-IE are PAC-MDP.

[11]This result applies to a version of Prioritized Sweeping that is a slight modification of the algorithm presented by (Moore & Atkeson, 1993). Specifically, like the RTDP-RMAX algorithm, the parameters ϵ_1 and m are used to restrict the total number of model and action-value updates.

Prioritized Sweeping may perform more Bellman backups during one timestep then either of these algorithms but the additional backups can only help in terms of sample complexity (although they do not improve the worst-case bounds).

2.12 Conclusion

In section we have examined several model-based RL algorithms. These algorithms make maximal use of their experience by computing an internal model of their environment. This model is used to reason about future actions and evaluate potential behaviors. A common theme among the provably efficient algorithms is that the agent must also compute its uncertainty of its own model. This allows a learning agent to seek out knowledge about uncertain regions in its model. By doing so in a principled manner, we were able to prove that during every step of the algorithm either a near-optimal policy is executed or the agent behaves in a way that tends to improve its model.

Chapter 3

Model-free Learning Algorithms

3.1 Q-learning

Q-learning is a very popular RL algorithm due to Watkins and Dayan (1992). In reality, Q-learning describes a large family of algorithms. The general outline of this family is given by the pseudo-code of Algorithm 6.

By choosing a way to initialize the action values (line 2), a scheme for selecting actions (line 6), and a method for computing the learning rates α_t (line 8), a concrete Q-learning algorithm can be constructed. For example, we could use *optimistic initialization*, *ε-greedy exploration*, and a constant learning rate α. These choices yield Algorithm 7.

Algorithm 6 Q-learning

```
0: Inputs: $S$, $A$, $\gamma$
1: for all $(s, a)$ do
2:     Initialize $Q(s, a)$ // action-value estimates
3: end for
4: for $t = 1, 2, 3, \cdots$ do
5:     Let $s$ denote the state at time $t$.
6:     Choose some action $a$.
7:     Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
       state $s$.
8:     $Q(s, a) \leftarrow (1 - \alpha_t)Q(s, a) + \alpha_t(r_t + \gamma V(s'))$
9: end for
```

3.1.1 Q-learning's Computational Complexity

During each timestep Q-learning must compute a random number, an argmax over actions, and update a single state-action value $Q(s, a)$. By storing the state-action values in a max-heap priority queue and assuming constant-time access to random

Algorithm 7 Q-learning with optimistic initalization, ε-greedy exploration, and constant learning rate

```
0: Inputs: $S$, $A$, $\gamma$, $\varepsilon$, $\alpha$
1: for all $(s, a)$ do
2:   $Q(s, a) \leftarrow 1/(1-\gamma)$ // action-value estimates
3: end for
4: for $t = 1, 2, 3, \cdots$ do
5:   Let $s$ denote the state at time $t$.
6:   Choose, uniformly at random, a number $Rand$ between 0 and 1.
7:   if $Rand < \varepsilon$ then
8:     Let $a$ be an action chosen uniformly at random from the set of all actions.
9:   else
10:    Let $a = \text{argmax}_{a' \in A} Q(s, a')$
11:  end if
12:  Execute action $a$ from the current state.
13:  Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
     state $s$.
14:  $Q(s, a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r_t + \gamma V(s'))$
15: end for
```

numbers, we have that the per-step computational complexity is $\ln(A)$.

3.1.2 Q-learning's Sample Complexity

It is difficult to analyze the sample complexity of every possible Q-learning algorithm. In Section 4.1, we show that several common varieties (optimistic initialization, constant or linear learning rates, and ε-greedy exploration) are provably not PAC-MDP. No member of this Q-learning family of algorithms is known to be PAC-MDP and whether any exists is an important open problem in the field. More discussion can be found in Section 4.1.

3.2 Delayed Q-learning

The *Delayed Q-learning* algorithm was introduced by Strehl et al. (2006c) as the first algorithm that is known to be PAC-MDP and whose per-timestep computational demands are minimal (roughly equivalent to those of Q-learning). Due to its low memory requirements it can also be viewed as a *model-free* algorithm and the first to be provably PAC-MDP. Its analysis is also noteworthy because the polynomial upper bound on its sample complexity is a significant improvement, asymptotically, over the best previously known upper bound for any algorithm, when only the dependence on S and A is considered. The algorithm is called "delayed" because it waits until a state-action pair has been experienced m times before updating that state-action pair's associated action value, where m is a parameter provided as input. When it does update an action value, the update can be viewed as an average of the target values for the m most recently missed update opportunities. Like all of the algorithms for which we provide a PAC-MDP proof, Delayed Q-learning performs a finite number of action-value updates. Due to the strict restrictions on the computational demands used by Delayed Q-learning, slightly more sophisticated internal logic is needed to guarantee this property. Pseudo-code[1] for Delayed Q-learning is provided in Algorithm 8.

In addition to the standard inputs, the algorithm also relies on two free parameters,

- $\epsilon_1 \in (0, 1)$: Used to provide a constant "exploration" bonus that is added to each action-value estimate when it is updated.
- A positive integer m: Represents the number of experiences of a state-action pair before an update is allowed.

In the analysis of Section 3.3, we provide precise values for m and ϵ_1 in terms of the other inputs (S, A, ϵ, δ, and γ) that guarantee the resulting algorithm is PAC-MDP. In addition to its action-value estimates, $Q(s, a)$, the algorithm also maintains the following internal variables,

- $l(s, a)$ for each (s, a): The number of samples (or target values) gathered for (s, a).
- $U(s, a)$ for each (s, a): Stores the running average of target values used to update $Q(s, a)$ once enough samples have been gathered.
- $b(s, a)$ for each (s, a): The timestep for which the first experience of (s, a) was obtained for the most recent or ongoing attempted update.

[1]Compared to the implementation provided by Strehl et al. (2006c), we've modified the algorithm to keep track of $b(s, a)$, the "beginning" timestep for the current attempted update for (s, a). The original pseudo-code kept track of $t(s, a)$, the time of the last attempted update for (s, a). The original implementation is less efficient and adds a factor of 2 to the computational bounds. The analysis of Strehl et al. (2006c) applies to the pseudo-code presented here.

Algorithm 8 Delayed Q-learning

```
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:   $U(s,a) \leftarrow 0$   // used for attempted updates
4:   $l(s,a) \leftarrow 0$   // counters
5:   $b(s,a) \leftarrow 0$   // beginning timestep of attempted update
6:   $LEARN(s,a) \leftarrow true$   // the LEARN flags
7: end for
8: $t^* \leftarrow 0$   // time of most recent action value change
9: for $t = 1, 2, 3, \cdots$ do
10:   Let $s$ denote the state at time $t$.
11:   Choose action $a := \text{argmax}_{a' \in A} Q(s,a')$.
12:   Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
      state $s$.
13:   if $b(s,a) \leq t^*$ then
14:     $LEARN(s,a) \leftarrow true$
15:   end if
16:   if $LEARN(s,a) = true$ then
17:     if $l(s,a) = 0$ then
18:       $b(s,a) = t$
19:     end if
20:     $l(s,a) \leftarrow l(s,a) + 1$
21:     $U(s,a) \leftarrow U(s,a) + r + \gamma \max_{a'} Q(s',a')$
22:     if $l(s,a) = m$ then
23:       if $Q(s,a) - U(s,a)/m \geq 2\epsilon_1$ then
24:         $Q(s,a) \leftarrow U(s,a)/m + \epsilon_1$
25:         $t^* \leftarrow t$
26:       else if $b(s,a) > t^*$ then
27:         $LEARN(s,a) \leftarrow false$
28:       end if
29:       $U(s,a) \leftarrow 0$
30:       $l(s,a) \leftarrow 0$
31:     end if
32:   end if
33: end for
```

- $LEARN(s,a) \in \{true, false\}$ for each (s,a): A Boolean flag that indicates whether, or not, samples are being gathered for (s,a).

3.2.1 The Update Rule

Suppose that at time $t \geq 1$, action a is performed from state s, resulting in an *attempted update*, according to the rules to be defined in Section 3.2.2. Let $s_{k_1}, s_{k_2}, \ldots, s_{k_m}$ be the m most recent next-states observed from executing (s,a), at times $k_1 < k_2 < \cdots < k_m$, respectively ($k_m = t$). For the remainder of the paper, we also let r_i denote the ith reward received during execution of Delayed Q-learning.

Thus, at time k_i, action a was taken from state s, resulting in a transition to state s_{k_i} and an immediate reward r_{k_i}. After the tth action, the following update occurs:

$$Q_{t+1}(s,a) = \frac{1}{m}\sum_{i=1}^{m}(r_{k_i} + \gamma V_{k_i}(s_{k_i})) + \epsilon_1 \tag{3.1}$$

as long as performing the update would result in a new action-value estimate that is at least ϵ_1 smaller than the previous estimate. In other words, the following equation must be satisfied for an update to occur:

$$Q_t(s,a) - \left(\frac{1}{m}\sum_{i=1}^{m}(r_{k_i} + \gamma V_{k_i}(s_{k_i}))\right) \geq 2\epsilon_1. \tag{3.2}$$

If any of the above conditions do not hold, then no update is performed. In this case, $Q_{t+1}(s,a) = Q_t(s,a)$.

3.2.2 Maintenance of the *LEARN* Flags

We provide an intuition behind the behavior of the *LEARN* flags. Please see Algorithm 8 for a formal description of the update rules. The main computation of the algorithm is that every time a state-action pair (s,a) is experienced m times, an update of $Q(s,a)$ is attempted as in Section 3.2.1. For our analysis to hold, however, we cannot allow an infinite number of attempted updates. Therefore, attempted updates are only allowed for (s, a) when $LEARN(s, a)$ is $true$. Besides being set to $true$ initially, $LEARN(s, a)$ is also set to $true$ when any state-action pair is updated (because our estimate $Q(s, a)$ may need to reflect this change). $LEARN(s, a)$ can only change from $true$ to *false* when no updates are made during a length of time for which (s, a) is experienced m times and the next attempted update of (s, a) fails. In this case, no more attempted updates of (s, a) are allowed until another action-value estimate is updated.

3.2.3 Delayed Q-learning's Model

Delayed Q-learning was introduced as a *model-free* algorithm. This terminology was justified by noting that the space complexity of Delayed Q-learning ($O(SA)$) is much less than what is needed to explicitly represent an MDP's transition probabilities ($O(S^2A)$). However, there is a sense in which Delayed Q-learning can be thought of as using a model. This interpretation follows from the fact that Delayed Q-learning's update (Equation 3.1) is identical to ϵ_1 plus the result of a full Bellman backup using the empirical (maximum likelihood) model derived from the m most recent experiences of the state-action pair being updated. Since m is much less than what is needed to accurately model the true transition probability (in the $L1$ distance metric), we say that Delayed Q-learning uses a *sparse model* (Kearns & Singh, 1999). In fact, Delayed Q-learning uses this sparse model precisely once, throws it away, and then proceeds to gather experience for another sparse model. When $m = 1$, this process may occur on every step and the algorithm behaves very similarly to Q-learning.

3.3 Analysis of Delayed Q-learning

3.3.1 Computational Complexity

On most timesteps, Delayed Q-learning performs only a constant amount of computation. Its worst case computational complexity per timestep is

$$\theta(\ln(A)), \tag{3.3}$$

where the log term is due to updating the priority queue that holds the action-value estimates for each state. Since Delayed Q-learning performs at most $SA\left(1+\frac{SA}{(1-\gamma)\epsilon_1}\right)$ attempted updates (see Lemma 14), the total computation time of Delayed Q-learning is

$$O\left(B+\frac{S^2A^2\ln(A)}{\epsilon_1(1-\gamma)}\right), \tag{3.4}$$

where B is the number of timesteps for which Delayed Q-learning is executed.

3.3.2 Sample Complexity

In this section, we show that Delayed Q-learning is PAC-MDP.

Theorem 7 *(Strehl et al., 2006c) Suppose that* $0 \le \epsilon < \frac{1}{1-\gamma}$ *and* $0 \le \delta < 1$ *are two real numbers and* $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma\rangle$ *is any MDP. There exists inputs* $m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$ *and* ϵ_1*, satisfying* $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{\ln(3SA(1+SA/(\epsilon_1(1-\gamma)))/\delta)}{2{\epsilon_1}^2(1-\gamma)^2}\right)$ *and* $\frac{1}{\epsilon_1} = O(\frac{1}{\epsilon(1-\gamma)})$*, such that if Delayed Q-learning is executed on* M*, then the following holds. Let* $\mathcal{A}_t$ *denote Delayed Q-learning's policy at time* t *and* s_t *denote the state at time* t*. With probability at least* $1-\delta$*,* $V_M^{\mathcal{A}_t}(s_t) \ge V_M^*(s_t) - \epsilon$ *is true for all but* $O\left(\frac{SA}{\epsilon^4(1-\gamma)^8}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\ln\frac{SA}{\delta\epsilon(1-\gamma)}\right)$ *timesteps* t*.*

Definition 7 *An* **update** *(or* **successful update***) of state-action pair* (s,a) *is a timestep* t *for which a change to the action-value estimate* $Q(s,a)$ *occurs. An* **attempted update** *of state-action pair* (s,a) *is a timestep* t *for which* (s,a) *is experienced,* $LEARN(s,a) = true$ *and* $l(s,a) = m$*.*

To prove the main theorem we need some additional results. The following Lemmas are taken from Strehl et al. (2006c).

Lemma 13 *(Strehl et al., 2006c) The total number of updates during any execution of Delayed Q-learning is at most*

$$\frac{SA}{\epsilon_1(1-\gamma)}. \tag{3.5}$$

Proof: Consider a fixed state-action pair (s,a). Its associated action-value estimate $Q(s,a)$ is initialized to $1/(1-\gamma)$ before any updates occur. Each time $Q(s,a)$ is updated it decreases by at least ϵ_1. Since all rewards encountered are non-negative, the quantities involved in any update (see Equation 3.1) are non-negative. Thus, $Q(s,a)$ cannot fall below 0. From this, it follows that $Q(s,a)$ cannot be updated more than $1/(\epsilon(1-\gamma))$ times. Since there are SA state-action pairs, we have that there are at most $SA/(\epsilon(1-\gamma))$ total updates. □

Lemma 14 *(Strehl et al., 2006c) The total number of attempted updates during any execution of Delayed Q-learning is at most*

$$SA\left(1+\frac{SA}{(1-\gamma)\epsilon_1}\right). \tag{3.6}$$

Proof: Consider a fixed state-action pair (s,a). Once (s,a) is experienced for the mth time, an attempted update will occur. Suppose that an attempted update of (s,a) occurs during timestep t. After this, for another attempted update to occur during some later timestep t', it must be the case that a successful update of some state-action pair (not necessarily (s,a)) has occurred on or after timestep t and before timestep t'. From Lemma 13, there can be at most $SA/(\epsilon(1-\gamma))$ total successful updates. Therefore, there are at most $1+SA/(\epsilon(1-\gamma))$ attempted updates of (s,a). Since there are SA state-action pairs, there can be at most $SA(1+SA/(\epsilon(1-\gamma)))$ total attempted updates. □

Definition 8 *During timestep t during execution of Delayed Q-learning, we define K_t to be the set*

$$K_t := \left\{(s,a)\in\mathcal{S}\times\mathsf{A} \mid Q_t(s,a)-\left(R(s,a)+\gamma\sum_{s'}T(s'|s,a)V_t(s')\right)\le 3\epsilon_1\right\}. \tag{3.7}$$

Definition 9 *Suppose we execute Delayed Q-learning in an MDP M. Define* **Event A1** *to be the event that there does not exist a timestep t such that an attempted yet unsuccessful update of some state-action pair (s,a) occurs during timestep t and the following holds:*

$$(s,a)\notin K_{k_1}, \tag{3.8}$$

where $k_1 < k_2 < \cdots < k_m = t$ are m timesteps during which (s,a) is experienced consecutively by the agent.

Lemma 15 *(Strehl et al., 2006c) Suppose we execute Delayed Q-learning with parameter m satisfying*

$$m \geq \frac{\ln\left(3SA(1 + SA/(\epsilon_1(1-\gamma)))/\delta\right)}{2{\epsilon_1}^2(1-\gamma)^2} \tag{3.9}$$

in an MDP M. The probability that event A1 occurs is greater than or equal to $1-\delta/3$.

Proof: Fix any timestep k_1 (and the complete history of the agent up to k_1) satisfying: $(s,a) \notin K_{k_1}$ is to be experienced by the agent on timestep k_1 and if (s,a) is experienced $m-1$ more times after timestep k_1, then an attempted update will result. Let $\mathcal{Q} = [(s[1], r[1]), \ldots, (s[m], r[m])] \in (S \times \mathbb{R})^m$ be any sequence of m next-state and immediate reward tuples. Due to the Markov assumption, whenever the agent is in state s and chooses action a, the resulting next-state and immediate reward are chosen independently of the history of the agent. Thus, the probability that (s,a) is experienced $m-1$ more times and that the resulting next-state and immediate reward sequence equals $\mathcal{Q}$ is at most the probability that $\mathcal{Q}$ is obtained by m independent draws from the transition and reward distributions (for (s,a)). Therefore, it suffices to prove this lemma by showing that the probability that a random sequence $\mathcal{Q}$ could cause an unsuccessful update of (s,a) is at most $\delta/3$. We prove this statement next.

Suppose that m rewards, $r[1], \ldots, r[m]$, and m next states, $s[1], \ldots, s[m]$, are drawn independently from the reward and transition distributions, respectively, for (s,a). By a straightforward application of the Hoeffding bound (with random variables $X_i := r[i] + \gamma V_{k_1}(s[i])$), it can be shown that our choice of m guarantees that

$$\frac{1}{m}\sum_{i=1}^{m}(r[i] + \gamma V_{k_1}(s[i])) - E[X_1] < \epsilon_1$$

holds with probability at least $1 - \delta/(3SA(1+SA\kappa))$. If it does hold and an attempted update is performed for (s,a) using these m samples, then the resulting update will succeed. To see the claim's validity, suppose that (s,a) is experienced at times $k_1 < k_2 < \cdots < k_m = t$ and at time k_i the agent is transitioned to state $s[i]$ and receives reward $r[i]$ (causing an attempted update at time t). Then, we have that

$$\begin{aligned} & Q_t(s,a) - \left(\frac{1}{m} \sum_{i=1}^{m} (r[i] + \gamma V_{k_i}(s[i])) \right) \\ > \;\; & Q_t(s,a) - E[X_1] - \epsilon_1 \;\; > \;\; 2\epsilon_1. \end{aligned}$$

We have used the fact that $V_{k_i}(s') \leq V_{k_1}(s')$ for all s' and $i = 1, \ldots, m$. Therefore, with high probability, Equation 3.2 will be true and the attempted update of $Q(s,a)$ at time k_m will succeed.

Finally, we extend our argument, using the union bound, to all possible timesteps k_1 satisfying the condition above. The number of such timesteps is bounded by the same bound we showed for the number of attempted updates ($SA(1 + SA\kappa)$). $\square$

The next lemma states that, with high probability, Delayed Q-learning will maintain optimistic action values.

Lemma 16 *(Strehl et al., 2006c) During execution of Delayed Q-learning, if m satisfies Equation 3.9, then $Q_t(s,a) \geq Q^*(s,a)$ holds for all timesteps t and state-action pairs (s,a), with probability at least $1 - \delta/3$.*

Proof: It can be shown, by a similar argument as in the proof of Lemma 15, that $(1/m)\sum_{i=1}^{m}(r_{k_i} + \gamma V^*(s_{k_i})) > Q^*(s,a) - \epsilon_1$ holds, for all attempted updates, with probability at least $1 - \delta/3$. Assuming this equation does hold, 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 $Q_t(s,a) \geq Q^*(s,a)$, and $V_t(s) \geq V^*(s)$ for all (s,a). Suppose s is the tth state reached and a is the action taken at time t. If it doesn't result in an attempted update or it results in an unsuccessful update, then no Action-value estimates change, and we are done. Otherwise, by Equation 3.1, we have that $Q_{t+1}(s,a) = (1/m)\sum_{i=1}^{m}(r_{k_i} + \gamma V_{k_i}(s_{k_i})) + \epsilon_1 \geq (1/m)\sum_{i=1}^{m}(r_{k_i} + \gamma V^*(s_{k_i})) + \epsilon_1 \geq Q^*(s,a)$, by the induction hypothesis and an application of the equation from above. $\square$

Lemma 17 *(Strehl et al., 2006c) If Event A1 occurs, then the following statement* *holds: If an unsuccessful update occurs at time t and $LEARN_{t+1}(s,a) = false$, then $(s,a) \in K_{t+1}$.*

Proof: Suppose an attempted update of (s,a) occurs at time t. Let $s_{k_1}, s_{k_2}, \ldots, s_{k_m}$ be the m most recent next-states resulting from executing action a from state s at times $k_1 < k_2 < \cdots < k_m = t$, respectively. By A1, if $(s,a) \notin K_{k_1}$, then the update will be successful. Now, suppose that $(s,a) \in K_{k_1}$ but that $(s,a) \notin K_{k_i}$ for some $i \in \{2, \ldots, m\}$. However, some Action-value estimate was successfully updated between time k_1 and time k_m (otherwise K_{k_1} would equal K_{k_1}). Thus, by the rules of Section 3.2.2, $LEARN(s,a)$ will be set to *true* after this unsuccessful update ($LEARN_{t+1}(s,a)$ will be true). □

The following lemma bounds the number of timesteps t in which a state-action pair $(s,a) \notin K_t$ is experienced.

Lemma 18 *(Strehl et al., 2006c) If Event A1 occurs, then the number of timesteps t such that a state-action pair $(s_t, a_t) \notin K_t$ is at most $2mSA/(\epsilon_1(1-\gamma))$.*

Proof: Suppose $(s,a) \notin K_t$ is experienced at time t and $\texttt{LEARN}_t(s,a) = \texttt{false}$ (implying the last attempted update was unsuccessful). By Lemma 17, we have that $(s,a) \in K_{t'+1}$ where t' was the time of the last attempted update of (s,a). Thus, some successful update has occurred since time $t'+1$. By the rules of Section 3.2.2, we have that $\texttt{LEARN}(s,a)$ will be set to `true` and by A1, the next attempted update will succeed.

Now, suppose that $(s,a) \notin K_t$ is experienced at time t and $\texttt{LEARN}_t(s,a) = \texttt{true}$. Within at most m more experiences of (s,a), an attempted update of (s,a) will occur. Suppose this attempted update takes place at time q and that the m most recent experiences of (s,a) happened at times $k_1 < k_2 < \cdots < k_m = q$. By A1, if $(s,a) \notin K_{k_1}$, the update will be successful. Otherwise, if $(s,a) \in K_{k_1}$, then some successful update must have occurred between times k_1 and t (since $K_{k_1} \neq K_t$). Hence, even if the update is unsuccessful, $LEARN(s,a)$ will remain `true`, $(s,a) \notin K_{q+1}$ will hold, and the next attempted update of (s,a) will be successful.

In either case, if $(s, a) \notin K_t$, then within at most $2m$ more experiences of (s, a), a successful update of $Q(s, a)$ will occur. Thus, reaching a state-action pair not in K_t at time t will happen at most $2mSA\kappa$ times. $\square$

Using these Lemmas we can prove the main result.

Proof: (of Theorem 7) We apply Theorem 1. Let $m = \frac{\ln(3SA(1+SA/(\epsilon_1(1-\gamma)))/\delta)}{2{\epsilon_1}^2(1-\gamma)^2}$ and $\epsilon_1 = \epsilon(1-\gamma)/3$. First, note that K_t is defined with respect to the agent's action-value estimates $Q(\cdot, \cdot)$ and other quantities that don't change during learning. Thus, we have that $K_t = K_{t+1}$ unless an update to some action-value estimate takes place. We now assume that Event A1 occurs, an assumption that holds with probability at least $1-\delta/3$, by Lemma 15. By Lemma 16, we have that Condition 1 of Theorem 1 holds, namely that $V_t(s) \geq V^*(s) - \epsilon$ for all timesteps t. Next, we claim that Condition 2, $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \frac{3\epsilon_1}{1-\gamma} = \epsilon$ also holds. For convenience let M' denote M_{K_t}. Recall that for all (s, a), either $Q_t(s, a) = Q^{\pi_t}_{M'}(s, a)$ when $(s, a) \notin K_t$, or $Q_t(s, a) - (R(s, a) + \gamma \sum_{s'} T(s'|s, a)V_t(s')) \leq 3\epsilon_1$ when $(s, a) \in K_t$ (see Equation 3.7). Note that $V^{\pi_t}_{M'}$ is the solution to the following set of equations:

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

The vector V_t is the solution to a similar set of equations except with some additional positive reward terms, each bounded by $3\epsilon_1$, implying that $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \frac{3\epsilon_1}{1-\gamma}$, as desired. Finally, for condition (3) of Theorem 1, we note that by Lemmas 13 and 18, $\zeta(\epsilon, \delta) = O\left(\frac{2mSA}{\epsilon_1(1-\gamma)}\right) = O\left(\frac{SA}{\epsilon^3(1-\gamma)^6} \ln \frac{SA}{\epsilon\delta(1-\gamma)}\right)$, where $\zeta(\epsilon, \delta)$ is the number of updates and escape events that occur during execution of Delayed Q-learning with inputs ϵ and δ (equivalently with inputs ϵ_1 and m, which are derived from ϵ and δ). $\square$

In summary, the development and analysis of Delayed Q-learning demonstrates that the naïve approach to exploration is powerful enough to yield a PAC-MDP model-free algorithm with minimal computational and space complexity. We say that Delayed Q-learning uses the naïve type of exploration because it waits for at least m samples before performing an update. Based on the results of our experiments (see Section 5), it is clear that Delayed Q-learning is not extremely practical. Next, we present a new algorithm that preserves the attractive theoretical properties of Delayed Q-learning but often achieves better performance.

3.4 Delayed Q-learning with IE

The Delayed Q-learning can be modified to use the IE approach rather than the naïve approach, improving its performance in some domains. This modification yields a new algorithm, *Delayed Q-learning with IE.* Pseudo-code is provided in Algorithm 9. The main difference between Delayed Q-learning and Delayed Q-learning with IE is that while Delayed Q-learning waits for m experiences to attempt a single update, Delayed Q-learning with IE is allowed to attempt an update during each of these m experiences. The exploration bonus that is added to the update decreases proportional to the square root of m. Besides the inputs provided to the Delayed Q-learning algorithm, Delayed Q-learning with IE has one addition parameter,

- A positive real number β: This quantity is used to determine the "exploration bonus" added to each action-value estimate when it is updated.

3.4.1 The Update Rule

Consider a single state-action pair during execution of Delayed Q-learning with IE. At the beginning of execution $LEARN(s, a)$ is set to *true*. It will remain this way until (s, a) is experienced m times and no successful update of any state-action pair occurs. Furthermore, each time (s, a) is experienced before the mth time, its associated counter, $l(s, a)$ is increased by one. Once (s, a) is successfully updated or $l(s, a)$ reaches m, the counter is reset to 0. We call each sequence of experiences of (s, a) at times $t_1 < t_2 < \ldots < t_j$ a *set of attempted updates* for (s, a), if (1) $LEARN_{t_i}(s, a) = true$ for $i = 1, \ldots, j$, (2) there are no other experiences of (s, a) between times t_1 and t_j, (3) $l_{t_i}(s, a) = i - 1$, for $i = 1, \ldots, j$, and (4) $l_{t_j+1}(s, a) = 0$. If none of the attempted

Algorithm 9 Delayed Q-learning with IE

```
0: **Inputs:** $\gamma$, $S$, $A$, $m$, $\epsilon_1$, $\beta$
1: **for all** $(s, a)$ **do**
2:    $Q(s, a) \leftarrow 1/(1 - \gamma)$    // Action-value estimates
3:    $U(s, a) \leftarrow 0$    // used for attempted updates
4:    $l(s, a) \leftarrow 0$    // counters
5:    $b(s, a) \leftarrow 0$    // beginning timestep of attempted update
6:    $LEARN(s, a) \leftarrow true$    // the LEARN flags
7: **end for**
8: $t^* \leftarrow 0$    // time of most recent Action-value change
9: **for** $t = 1, 2, 3, \cdots$ **do**
10:    Let $s$ denote the state at time $t$.
11:    Choose action $a := \text{argmax}_{a' \in A} Q(s, a')$.
12:    Let $r$ be the immediate reward and $s'$ the next state after executing action $a$ from
       state $s$.
13:    **if** $b(s, a) \leq t^*$ **then**
14:       $LEARN(s, a) \leftarrow true$
15:    **end if**
16:    **if** $LEARN(s, a) = true$ **then**
17:       **if** $l(s, a) = 0$ **then**
18:          $b(s, a) \leftarrow t$
19:          $U(s, a) \leftarrow 0$
20:       **end if**
21:       $l(s, a) \leftarrow l(s, a) + 1$
22:       $U(s, a) \leftarrow U(s, a) + r + \gamma \max_{a'} Q(s', a')$
23:       **if** $Q(s, a) - \left(U(s, a)/l(s, a) + \beta/\sqrt{l(s, a)}\right) \geq \epsilon_1$ **then**
24:          $Q(s, a) \leftarrow U(s, a)/l(s, a) + \beta/\sqrt{l(s, a)}$
25:          $t^* \leftarrow t$
26:          $l(s, a) \leftarrow 0$
27:       **else if** $l(s, a) = m$ **then**
28:          $l(s, a) \leftarrow 0$
29:          **if** $b(s, a) > t^*$ **then**
30:             $LEARN(s, a) \leftarrow false$
31:          **end if**
32:       **end if**
33:    **end if**
34: **end for**
```

updates at times $t_1, \ldots, t_j$ for (s, a) succeed, then we say that the set of attempted updates was unsuccessful. Next, we discuss the form of the attempted update.

Suppose that at time $t \geq 1$, action a is performed from state s, resulting in an *attempted update* (meaning that $LEARN(s, a)$ is *true*). This attempted update is part of a set of attempted updates, which so far includes the l experiences of (s, a) at times $k_1 < k_2 < \ldots < k_l = t$, where $l_{k_i}(s, a) = i - 1$. The attempted updates at times $k_1, \ldots, k_{l-1}$ were necessarily unsuccessful (otherwise the attempted update at time t would exist as part of a different set of attempted updates). Let $s_{k_1}, s_{k_2}, \ldots, s_{k_l}$ be the l most recent next-states observed from executing (s, a), at times $k_1 < k_2 < \cdots < k_l$, respectively ($k_l = t$).

Thus, at time k_i, action a was taken from state s, resulting in a transition to state s_{k_i} and an immediate reward r_{k_i}. Note that the counter $l_t(s, a)$ maintained by the algorithm has value $l - 1$ (it is incremented to l in line 21 of Algorithm 9). After the tth action, the following update occurs:

$$Q_{t+1}(s, a) = \frac{1}{l} \sum_{i=1}^{l} (r_{k_i} + \gamma V_{k_i}(s_{k_i})) + \beta/\sqrt{l}, \tag{3.10}$$

as long as performing the update would result in a new action-value estimate that is at least ϵ_1 smaller than the previous estimate. In other words, the following equation must be satisfied for an update to occur:

$$Q_t(s, a) - \left(\frac{1}{l} \sum_{i=1}^{l} (r_{k_i} + \gamma V_{k_i}(s_{k_i})) + \beta/\sqrt{l} \right) \geq \epsilon_1. \tag{3.11}$$

If any of the above conditions do not hold, then no update is performed. In this case, $Q_{t+1}(s, a) = Q_t(s, a)$.

3.4.2 Maintenance of the *LEARN* Flags

We provide an intuition behind the behavior of the *LEARN* flags. Please see Algorithm 9 for a formal description of the update rules. Attempted updates are only allowed for (s, a) when $LEARN(s, a)$ is *true*. Besides being set to *true* initially, $LEARN(s, a)$

is also set to *true* when any state-action pair is updated (because our estimate $Q(s,a)$ may need to reflect this change). $LEARN(s,a)$ can only change from *true* to *false* when no updates are made during a length of time for which (s,a) is experienced m times and none of the attempted updates during this period of experiences of (s,a) succeed. In this case, no more attempted updates of (s,a) are allowed until another action-value estimate is updated.

3.5 Analysis of Delayed Q-learning with IE

3.5.1 Computational Complexity

On most timesteps, Delayed Q-learning with IE performs only a constant amount of computation. Its worst case computational complexity per timestep is

$$\theta(\ln(A)), \tag{3.12}$$

where the log term is due to updating the priority queue that holds the action-value estimates for each state. Since Delayed Q-learning with IE performs at most $SAm\left(1+\frac{SA}{\epsilon_1(1-\gamma)}\right)$ attempted updates (see Lemma 20), the total computation time of Delayed Q-learning with IE is

$$O\left(B+\frac{S^2A^2m\ln(A)}{\epsilon_1(1-\gamma)}\right), \tag{3.13}$$

where B is the number of timesteps for which Delayed Q-learning with IE is executed.

3.5.2 Sample Complexity

The main result of this section is to prove the following theorem, which states that Delayed Q-learning with IE is PAC-MDP. For convenience we introduce the following notation:

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

Theorem 8 *(Strehl & Littman, 2006) Suppose that $0 \leq \epsilon < \frac{1}{1-\gamma}$ and $0 \leq \delta < 1$ are two real numbers and $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma\rangle$ is any MDP. There exists inputs β,*

$m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$, and ϵ_1, satisfying $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{\ln\left(3\xi/(2\epsilon_1^2(1-\gamma)^2\delta)\right)}{2{\epsilon_1}^2(1-\gamma)^2}\right)$ and $\frac{1}{\epsilon_1} = O(\frac{1}{\epsilon(1-\gamma)})$ such that if Delayed Q-learning with IE is executed on M, then the following holds. Let $\mathcal{A}_t$ denote Delayed Q-learning with IE's policy at time t and s_t denote the state at time t. With probability at least $1-\delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ is true for all but $O\left(\frac{SA}{\epsilon^4(1-\gamma)^8} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{SA}{\delta\epsilon(1-\gamma)}\right)$ timesteps t.

To prove this theorem we need some lemmas. The following is a helpful definition.

Definition 10 *An* **update** *(or* **successful attempted update***) of state-action pair (s,a) is a timestep t for which a change to the action-value estimate $Q(s,a)$ occurs. An* **attempted update** *of state-action pair (s,a) is a timestep t for which (s,a) is experienced and $LEARN(s,a) = true$. A* **set of** l **attempted updates** *is a non-extendable sequence of consecutive attempted updates at times $k_1 < k_2 < \ldots < k_l$ such that the counters $l_{k_i}(s,a) = i-1$ for $i = 1, 2, \ldots, l$. This definition is equivalent to the one given in Section 3.4.1. An* **unsuccessful set of attempted updates** *is a set of m attempted updates, none of which are successful.*

The next two Lemmas show that the number of successful and unsuccessful attempted updates during any execution of Delayed Q-learning with IE is finite. This fact is crucial to our argument that Delayed Q-learning with IE is PAC-MDP.

Lemma 19 *(Strehl & Littman, 2006) The total number of updates during any execution of Delayed Q-learning with IE is at most*

$$\frac{SA}{\epsilon_1(1-\gamma)}. \tag{3.14}$$

Proof: Consider a fixed state-action pair (s,a). Its associated action-value estimate $Q(s,a)$ is initialized to $1/(1-\gamma)$ before any updates occur. Each time $Q(s,a)$ is updated it decreases by at least ϵ_1. Since all rewards encountered are non-negative, the quantities involved in any update are non-negative. Thus, $Q(s,a)$ cannot fall below 0. It follows that $Q(s,a)$ cannot be updated more than $1/(\epsilon(1-\gamma))$ times. Since there are SA state-action pairs, there will be at most $SA/(\epsilon(1-\gamma))$ total updates. $\square$

Lemma 20 *(Strehl & Littman, 2006) The total number of attempted updates during any execution of Delayed Q-learning with IE is at most $m\xi$.*

Proof: Consider a single state-action pair (s, a). An attempted update for (s, a) occurs on those timesteps that (s, a) is experienced and $LEARN(s, a)$ is set to *true*. Every m experiences of (s, a) will cause $LEARN(s, a)$ to be set to *false* until a successful update of some state-action pair occurs. From Lemma 19, we know that there are at most $SA/(\epsilon_1(1-\gamma))$ successful updates. Hence, there will be at most $m + mSA/(\epsilon_1(1-\gamma))$ attempted updates of (s, a). Multiplying this expression by the number of state-action pairs, SA, yields the desired result. □

We can extend the previous bounds to a bound on the number of sets of attempted updates.

Corollary 1 *(Strehl & Littman, 2006) The total number of sets of m attempted updates is at most ξ.*

Now, we define the set K that will be used to apply Theorem 1 to the analysis of Delayed Q-learning with IE.

Definition 11 *On timestep t during execution of Delayed Q-learning with IE we define K_t to be the set*

$$K_t := \left\{ (s, a) \in \mathcal{S} \times \mathsf{A} \;\middle|\; Q_t(s, a) - \left(R(s, a) + \gamma \sum_{s'} T(s'|s, a) V_t(s') \right) \leq 3\epsilon_1 \right\}. \quad (3.15)$$

The set K (K_t at time t during execution of the algorithm) consists of the state-action pairs with low Bellman residual. The state-action pairs not in K are the ones whose action-value estimates are overly optimistic in the sense that they would decrease significantly if subjected to a Bellman backup (as in value iteration). The attempted updates of a state-action pair approximate a Bellman backup in two ways: random samples are used instead of the expected value and a bonus of $\beta/\sqrt{l(s, a)}$ is added to the action-value estimate. However, if β is chosen to be small enough, we would intuitively expect a set of attempted updates for a state-action pair not in K to result in a successful update. This idea is formalized below by showing that the event A1 (no successful update will take place for a state-action pair $(s, a) \notin K$) will occur.

Definition 12 *Suppose we execute Delayed Q-learning with IE in an MDP M. Define* **Event A1** *to be the event that there does not exist a timestep t such that an attempted yet unsuccessful update of some state-action pair (s, a) occurs during timestep t and the following conditions hold:*

$$l_t(s, a) = m \quad \text{and} \quad (s, a) \notin K_{k_1}, \tag{3.16}$$

where $k_1 < k_2 < \cdots < k_m = t$ are m timesteps during which (s, a) is experienced consecutively by the agent.

Lemma 21 *(Strehl & Littman, 2006) Suppose we execute Delayed Q-learning with IE with parameters m and β satisfying*

$$m \geq \frac{\ln(3\xi/\delta)}{2{\epsilon_1}^2(1-\gamma)^2} \tag{3.17}$$

and

$$\beta \leq \epsilon_1\sqrt{m} \tag{3.18}$$

in an MDP M. The probability that Event A1 occurs is greater than or equal to $1 - \delta/3$.

Proof: Fix any timestep k_1 (and the complete history of the agent up to k_1) satisfying all of the following three conditions: (1) $(s, a) \notin K_{k_1}$, (2) $l(s, a) = 0$, and (3) $LEARN(s, a) = true$ or $b(s, a) < t^*$. Intuitively, conditions (2) and (3) mean that this timestep is the first of a set of attempted updates for (s, a). We want to show that it is very unlikely for (s, a) to be experienced at $m - 1$ future timesteps, $k_1 < k_2 < \cdots < k_m$ and for no successful update of (s, a) to occur.

Let $Q = [(s[1], r[1]), \ldots, (s[m], r[m])] \in (S \times \mathbb{R})^m$ be any sequence of m next-state and immediate reward tuples. Due to the Markov assumption, whenever the agent is in state s and chooses action a, the resulting next-state and immediate reward are chosen independently of the history of the agent. Thus, the probability that (s, a) is experienced $m-1$ more times and that the resulting next-state and immediate reward sequence equals $\mathcal{Q}$ is at most the probability that $\mathcal{Q}$ is obtained by m independent draws from the transition and reward distributions (for (s,a)). Therefore, it suffices to prove this lemma by showing that the probability that a random sequence $\mathcal{Q}$ could cause unsuccessful updates of (s,a) at any times $k_1,\ldots,k_m$ is at most $\delta/3$. We prove this statement next.

Suppose that m rewards, $r[1],\ldots,r[m]$, and m next states, $s[1],\ldots,s[m]$, are drawn independently from the reward and transition distributions, respectively, for (s,a). By a straightforward application of the Hoeffding bound (with random variables $X_i := r[i]+\gamma V_{k_1}(s[i])$), it can be shown that our choice of m guarantees that

$$\frac{1}{m}\sum_{i=1}^{m}(r[i]+\gamma V_{k_1}(s[i])) - E[X_1] < \epsilon_1 \tag{3.19}$$

holds with probability at least $1-\delta/(3SA(1+SA/(\epsilon_1(1-\gamma)))) = 1-\delta/(3\xi)$. If it does hold and an attempted update is performed for (s,a) using these m samples, then the resulting update will succeed. To see the claim's validity, suppose that (s,a) is experienced at times $k_1<k_2<\cdots<k_m=t$ and at time k_i the agent is transitioned to state $s[i]$ and receives reward $r[i]$ (causing an attempted update at time t). Then, we have that

$$\begin{aligned}
&Q_{k_1}(s,a) - \left(\frac{1}{m}\sum_{i=1}^{m}(r[i]+\gamma V_{k_i}(s[i])) + \beta/\sqrt{m}\right)\\
&\geq\ Q_{k_1}(s,a) - \left(\frac{1}{m}\sum_{i=1}^{m}(r[i]+\gamma V_{k_1}(s[i])) + \beta/\sqrt{m}\right)\\
&>\ Q_{k_1}(s,a) - E[X_1] - \epsilon_1 - \beta/\sqrt{m}\\
&>\ 2\epsilon_1 - \beta/\sqrt{m}\ >\ \epsilon_1.
\end{aligned}$$

The first step follows from the fact that $V_{k_i}(s') \leq V_{k_1}(s')$ for all s' and $i=1,\ldots,m$. The second step results from an application of Equation 3.19. The third step uses the fact that $(s,a)\notin K_{k_1}$. The last step makes use of the bound $\beta \leq \epsilon_1\sqrt{m}$. Thus, with high probability, we have that

$$Q_{k_1}(s,a) - \left(\frac{1}{m} \sum_{i=1}^{m} (r[i] + \gamma V_{k_i}(s[i])) + \beta/\sqrt{m} \right) \geq \epsilon_1. \tag{3.20}$$

From the condition of Equation 3.11, this argument implies that, with high probability, at least one of the attempted updates on timesteps $k_1, \ldots, k_m$ will succeed. To verify this claim, note that if all of the first $m-1$ attempted updates are unsuccessful, then the mth attempted update is successful precisely when Equation 3.20 holds. We also note that it is possible that (s,a) will not be experienced $m-1$ more times after timestep k_1. This outcome is also fine because then event A1 will not be violated.

Finally, we extend our argument, using the union bound, to all possible timesteps k_1 satisfying the condition above. Each such timestep is the first of a set of attempted updates. Thus, by Corollary 1, the number of such timesteps is bounded by ξ. □

The next lemma states that, with high probability, Delayed Q-learning with IE will maintain optimistic action values.

Lemma 22 *(Strehl & Littman, 2006) During execution of Delayed Q-learning with IE with parameters satisfying*

$$\beta \geq \left(\frac{1}{1-\gamma} \right) \sqrt{\frac{\ln(\frac{3m\xi}{\delta})}{2}}, \tag{3.21}$$

$Q_t(s,a) \geq Q^*(s,a)$ *holds for all timesteps* t *and state-action pairs* (s,a) *with probability at least* $1 - \delta/3$.

Proof:

For a fixed state-action pair (s,a), suppose we draw l random rewards $r[i] \sim \mathcal{R}(s,a)$ and next states $s[i] \sim T(s,a)$. Define the random variables $X_i := r[i] + \gamma V^*(s[i])$, so that $E[X_i] = Q^*(s,a)$, for $i = 1, \ldots, l$. By Hoeffding's bound, we have that

$$\Pr\left[\frac{1}{l} \sum_{i=1}^{l} X_i < E[X_1] - \alpha \right] \leq e^{\frac{-2\alpha^2 l}{(1-\gamma)^2}}, \tag{3.22}$$

for any non-negative α. Consider any fixed attempted update of the state-action pair (s,a) at any time t. Recall from Equation 3.10 that if the update is successful, then the new action value estimate will be of the form

$$Q_{t+1}(s,a) = \frac{1}{l}\sum_{i=1}^{l}\left(r_{k_i} + \gamma V_{k_i}(s_{k_i})\right) + \beta/\sqrt{l}, \tag{3.23}$$

where $r_{k_i} \sim \mathcal{R}(s,a)$ and $s_{k_i} \sim T(s,a)$ were observed by the agent at some times k_i after experiencing (s,a), for $i = 1,\ldots,l$. Thus, by our choice of β and from Equation 3.22 (substituting $\beta/\sqrt{l}$ for α), we have that

$$\Pr\left[\frac{1}{l}\sum_{i=1}^{l}\left(r_{k_i} + \gamma V^*(s_{k_i})\right) < Q^*(s,a) - \beta/\sqrt{l}\right] \leq \frac{\delta}{3m\xi}. \tag{3.24}$$

Since there are at most $m\xi$ attempted updates, by Lemma 20, we conclude that

$$\frac{1}{l}\sum_{i=1}^{l}\left(r_{k_i} + \gamma V^*(s_{k_i})\right) \geq Q^*(s,a) - \beta/\sqrt{l} \tag{3.25}$$

holds, with probability at least $1-\delta/3$, for all attempted updates executed by Delayed Q-learning with IE. From now on, assume that Equation 3.25 is not violated.

Now, 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). Suppose the claim holds for all timesteps less than or equal to t. Thus, we have that $Q_t(s,a) \geq Q^*(s,a)$, and $V_t(s) \geq V^*(s)$ for all (s,a). Suppose s is the tth state reached and a is the action taken at time t. If it doesn't result in an attempted update or the resulting attempted udpate is unsuccessful, then no action-value estimates change, and we are done. Otherwise, by Equation 3.10, we have that $Q_{t+1}(s,a) = (1/l)\sum_{i=1}^{l}(r_{k_i} + \gamma V_{k_i}(s_{k_i})) + \beta/\sqrt{l} \geq (1/l)\sum_{i=1}^{l}(r_{k_i} + \gamma V^*(s_{k_i})) + \beta/\sqrt{l} \geq Q^*(s,a)$, by the induction hypothesis and an application of Equation 3.25. $\square$

Lemma 23 *(Strehl & Littman, 2006) If Event A1 occurs, then the following statement holds: If a set of m unsuccessful attempted updates ends at time t and $LEARN_{t+1}(s,a) = false$, then $(s,a) \in K_{t+1}$.*

Proof: Suppose an attempted but unsuccessful update of (s,a) occurs at time t and $l_t(s,a) = m-1$, so that this step is the final attempted update of a set of m attempted updates. We want to show that if $(s,a) \notin K_{t+1}$ then $LEARN_{t+1}(s,a) = false$. Let $s_{k_1}, s_{k_2}, \ldots, s_{k_m}$ be the m most recent next-states resulting from executing action a from state s at times $k_1 < k_2 < \cdots < k_m = t$, respectively. By Event A1, if $(s,a) \notin K_{k_1}$, then the update will necessarily be successful, which is a contradiction. Thus, $(s,a) \in K_{k_1}$. Suppose that $(s,a) \notin K_{k_i}$ for some $i \in \{2, \ldots, m\}$. This condition implies that $(s,a) \notin K_m$ due to the fact that the action-value estimates are non-increasing and by the fact that no successful updates of (s,a) have occurred between times k_1 and k_m. Then, some action-value estimate was successfully updated between time k_1 and time k_m (otherwise K_{k_1} would equal K_{k_m}). Thus, by the rules of Section 3.4.2, $LEARN(s,a)$ will be set to *true* after this unsuccessful update (equivalently, $LEARN_{t+1}(s,a) = true$). □

The following lemma bounds the number of timesteps t in which a state-action pair $(s,a) \notin K_t$ is experienced.

Lemma 24 *(Strehl & Littman, 2006) If Event A1 occurs, then the number of timesteps t such that a state-action pair $(s_t, a_t) \notin K_t$ is at most $2mSA/(\epsilon_1(1-\gamma))$.*

Proof: Suppose $(s,a) \notin K_t$ is experienced at time t and $LEARN_t(s,a) = false$ (implying the last set of attempted updates was unsuccessful). By Lemma 23, we have that $(s,a) \in K_{t'+1}$ where t' was the time of the last attempted update of (s,a). Thus, some successful update has occurred since time $t'+1$. By the rules of Section 3.4.2, we have that $LEARN(s,a)$ will be set to *true* and by Event A1, the next attempted update will succeed.

Now, suppose that $(s,a) \notin K_t$ is experienced at time t and $LEARN_t(s,a) = true$. Within at most m more experiences of (s,a), a set of attempted updates for (s,a) will take place. Suppose this set of attempted updates ends at time q and that the $l \leq m$ most recent experiences of (s,a) happened at times $k_1 < k_2 < \cdots < k_l = q$. By A1, if $(s,a) \notin K_{k_1}$, the set of attempted updates will be successful (one of the attempted updates will succeed). Otherwise, if $(s,a) \in K_{k_1}$, then some successful update must have occurred between times k_1 and t (since $K_{k_1} \neq K_t$). Hence, even if the update is unsuccessful, $LEARN(s,a)$ will remain *true*, $(s,a) \notin K_{q+1}$ will hold, and the next set of attempted updates of (s,a) will be successful.

In either case, if $(s,a) \notin K_t$, then within at most $2m$ more experiences of (s,a), a successful update of $Q(s,a)$ will occur. Thus, reaching a state-action pair not in K_t at time t will happen at most $2mSA/(\epsilon_1(1-\gamma)$ times. $\square$

Next, we present the proof of the main result.

Proof: (of Theorem 8) We apply Theorem 1. Let $\beta = \epsilon_1\sqrt{m}$, $m = \frac{\ln\left(3\xi/(2\epsilon_1^2(1-\gamma)^2\delta)\right)}{2{\epsilon_1}^2(1-\gamma)^2}$, and $\epsilon_1 = \epsilon(1-\gamma)/9$. First, note that K_t is defined with respect to the agent's action-value estimates $Q(\cdot,\cdot)$ and other quantities that don't change during learning. Thus, we have that $K_t = K_{t+1}$ unless an update to some action-value estimate takes place. We now assume that Event A1 occurs, an assumption that holds with probability at least $1-\delta/3$, by Lemma 21. By Lemma 22, we have that Condition 1 of Theorem 1 holds, namely that $V_t(s) \geq V^*(s) - \epsilon$ for all timesteps t. Next, we claim that Condition 2, $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \frac{3\epsilon_1}{1-\gamma} = \epsilon$ also holds. Recall that for all (s,a), either $Q_t(s,a) = Q^{\pi_t}_{M'}(s,a)$ (when $(s,a) \notin K_t$), or $Q_t(s,a) - (R(s,a) + \gamma\sum_{s'} T(s'|s,a)V_t(s')) \leq 3\epsilon_1$ (when $(s,a) \in K_t$). Note that $V^{\pi_t}_{M'}$ is the solution to the following set of equations:

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

The vector V_t is the solution to a similar set of equations except with some additional positive reward terms, each bounded by $3\epsilon_1$, implying that $V_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \frac{3\epsilon_1}{1-\gamma}$, as desired. Finally, for Condition 3 of Theorem 1, we note that by Lemma 24, $\zeta(\epsilon,\delta) \leq \frac{2mSA}{\epsilon_1(1-\gamma)} = O\left(\frac{SA}{\epsilon^3(1-\gamma)^6}\ln\frac{SA}{\epsilon\delta(1-\gamma)}\right)$. $\square$

In summary, we have shown that the Delayed Q-learning with IE algorithm preserves the attractive theoretical complexity bounds as the Delayed Q-learning algorithm. However, since it allows for quicker updates in some cases, we expect it to learn much faster. In our experiments in Section 5, Delayed Q-learning with IE outperformed Delayed Q-learning every time. In two out of three domains it outperformed Q-learning. In one domain it outperformed the model-based approaches R-MAX and E^3.

3.6 Conclusion

In this chapter we analyzed two model-free RL algorithms that are provably PAC-MDP. The result that Delayed Q-learning is PAC-MDP is the first one to show that model-free PAC-MDP algorithms are possible. Another interesting aspect about the analysis of Delayed Q-learning is that the sample complexity bounds depend only linearly on S. The Delayed Q-learning with IE algorithm and its analysis are new. Our experiments show that Delayed Q-learning with IE significantly outperforms Delayed Q-learning in terms of maximizing reward.

Chapter 4
Further Discussion

In this chapter we address various important issues related to be both model-based and model-free algorithms.

4.1 Lower Bounds

Although we have mostly considered only PAC-MDP algorithms, many algorithms are not PAC-MDP. In this section, we demonstrate that several common RL algorithms are not PAC-MDP. All proofs are deferred to the Appendix. First, we show that ε-greedy exploration combined with two different Q-learning style algorithms is not PAC-MDP. Then, we show that a model-based Certainty Equivalence approach is also not PAC-MDP, even when combined with optimistic initialization and ε-greedy exploration.

Consider the Q-learning algorithm with a constant learning rate, α, and ε-greedy exploration (see Algorithm 7). The next Lemma shows that ε cannot be too large. Intuitively, even if Q-learning discovers the optimal value function, as ε is increased, the value of a ε-greedy exploration policy degrades.

Lemma 25 *(Strehl & Littman, 2006) Let $\mathcal{A}$ be the Q-learning algorithm with ε-greedy exploration and constant learning rate $\alpha > 0$. If it is PAC-MDP, then $\varepsilon \le \epsilon(1-\gamma)$.*

Proof: Consider the following MDP M (see Figure 4.1). It has 3 states and 2 actions, a_1, and a_2. Action a_1 from state 1 results in a transition to state 2 and a reward of 0. Action a_2 from state 1 results in a transition to state 3 and a reward of 1. Both actions from states 2 and 3 are self-loops that return the agent to the state it started from. Both actions from state 2 result in a reward of 0 while both actions from state 3 result in a reward of 1. We have that the optimal policy is to take action a_2 from state

1 and that $V^*(1) = 1/(1-\gamma) = V^*(3)$ but that $V^*(2) = 0$. Now, suppose that once the agent reaches either state 2 or 3, learning halts and the agent is put back in state 1. Also, assume that the action values of the agent satisfy $Q(2,\cdot) = Q^*(2,\cdot) = 0$ and $Q(3,\cdot) = Q^*(3,\cdot) = 1/(1-\gamma)$. Such a situation could be simulated to desired accuracy by a more complex MDP[1] and allowing enough time for $Q(2,\cdot)$ and $Q(3,\cdot)$ to converge, but such a construction is omitted for simplicity.

Whenever in state 1, the agent executes some non-stationary policy. However, because of the nature of M, the first action choice of the agent is the only one that has any effect on the value of the agent's policy. Thus, the value of the agent's policy is essentially the value of $\pi(\varepsilon)$ or $\pi(1-\varepsilon)$, where $\pi(x)$ denotes the policy that chooses action a_1 from state 1 with probability x and action a_2 from state 1 with probability $1-x$. We now characterize the possible values of x for which $\pi(x)$ is ϵ-optimal:

$$\begin{aligned} & V^*(1) - V^{\pi(x)}(1) \leq \epsilon \\ \Leftrightarrow\quad & \frac{1}{1-\gamma} - (1-x)\frac{1}{1-\gamma} \leq \epsilon \\ \Leftrightarrow\quad & 1-x \geq 1-\epsilon(1-\gamma) \\ \Leftrightarrow\quad & x \leq \epsilon(1-\gamma). \end{aligned}$$

Consider running Q-learning on the MDP M with ϵ-greedy exploration. We have assumed that $V(s_2) < V(s_3)$. Since it will visit s_1 infinitely often it will eventually learn to favor action a_2. Thus, from that point on it will execute policy $\pi(\varepsilon)$. Since it is PAC-MDP, it must be ϵ-optimal and therefore $\varepsilon \leq \epsilon(1-\gamma)$, as desired. $\square$

We note that the proof of Lemma 25 holds for any learning algorithm that uses ε-greedy exploration and whose action value estimates $Q(s,a)$ converge to $Q^*(s,a)$ in deterministic domains assuming that each state-action pair is tried infinitely often.

Theorem 9 *(Strehl & Littman, 2006) The Q-learning algorithm with a constant learning rate, $\alpha > 0$, and ε-greedy exploration is not a PAC-MDP algorithm.*

[1]We could, for instance, allow a tiny probability of transition from states 2 and 3 back to state 1.

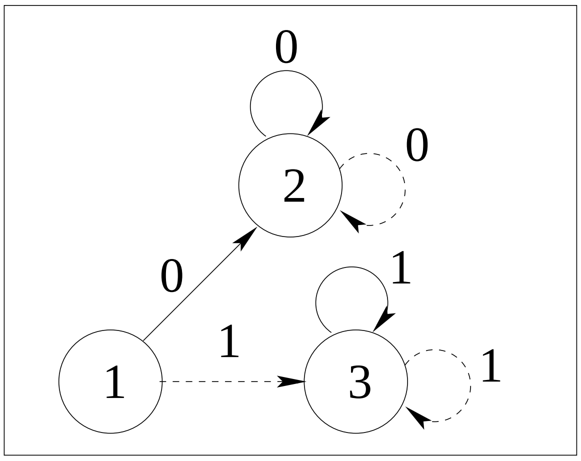


Figure 4.1: The MDP used in the proof of Lemma 25, showing that Q-learning must have a small random exploration probability. The states are represented as nodes and the actions as edges ($a1$ is a solid edge and $a2$ is a dashed edge). The reward for taking an action is written above the corresponding edge.

Proof:(of Theorem 9) Consider the following MDP M (see Figure 4.2). It has 4 states and 2 actions, r and l. The reward for both actions from states 1 and 3 is 0, from state 2 is 1, and from state 4 is $(1-\gamma)$. From state 1, action r is stochastic and results in a transition to state 2 with probability 1/2 and to state 3 with probability 1/2. From state 1, action l is deterministic and results in a transition to state 4. Both actions from states 2, 3, and 4 are self-loops, leaving the agent in the same state that it started in. Thus, we see that the value of state 2 is $1/(1-\gamma)$, while the values of states 3 and 4 are 0 and 1, respectively. We also have that $Q^*(1,l) = \gamma$ and $Q^*(1,r) = \gamma/(2(1-\gamma))$. Assuming $\gamma > 1/2$, the optimal policy is to take action r from state 1. We will assume that once the agent reaches either state 2, 3, or 4, learning halts and the agent is put back in state 1. We also assume that the action values of the agent satisfy $Q(2,\cdot) = Q^*(2,\cdot) = 1/(1-\gamma)$, $Q(3,\cdot) = Q^*(3,\cdot) = 0$, and $Q(4,\cdot) = Q^*(4,\cdot) = 1$. Such a situation could be simulated to desired accuracy by a more complex MDP (and allowing enough time for $Q(2,\cdot)$, $Q(3,\cdot)$, and $Q(4,\cdot)$ to converge) but such an argument is omitted here for simplicity.

Let $\gamma > 4/5$, $0 < \epsilon < 1/2$, and $\delta > 0$. Let α and ε be the learning rate and exploration parameters, respectively, used by Q-learning. They can be chosen as any function of S,A,γ,ϵ, and δ. First, consider the case of $\varepsilon > 0$. The action values are initialized arbitrarily. We know that both actions from state 1 will be tried infinitely often, due to the fact that $\varepsilon > 0$ and that state 1 is visited infinitely often. From Lemma 25, we can assume that $\varepsilon \leq \epsilon(1-\gamma)/\gamma$, because otherwise the algorithm would definitely not be PAC-MDP and we'd be done. On any timestep on which the agent is in state 1, the agent has some non-stationary policy. However, because of the nature of M, the first action choice of the agent is the only one that has any effect on the value of the agent's policy. Thus, the value of the agent's policy is essentially the value of $\pi(\varepsilon)$ or $\pi(1-\varepsilon)$, where $\pi(x)$ denotes the policy that chooses action r from state 1 with probability x and action l from state 1 with probability $1-x$. We claim that $\pi(\varepsilon)$ is not an ϵ-optimal policy. First, observe that

$$\begin{aligned} & V^*(1) - V^{\pi(\varepsilon)}(1) \\ = \ & \frac{\gamma}{2(1-\gamma)} - \left(\varepsilon\frac{\gamma}{2(1-\gamma)} + (1-\varepsilon)(\gamma)\right) \\ = \ & (1-\varepsilon)\left(\frac{\gamma}{2(1-\gamma)} - \gamma\right). \end{aligned}$$

Therefore, the policy $\pi(\varepsilon)$ is ϵ-optimal only if

$$1 - \varepsilon \leq \frac{\epsilon}{\gamma(1/(2(1-\gamma)) - 1)}. \tag{4.1}$$

From our assumption that $\varepsilon < \epsilon(1-\gamma)/\gamma$, we have that

$$1 - \varepsilon > 1 - \epsilon(1-\gamma)/\gamma. \tag{4.2}$$

From our choices of $\gamma > 4/5$ and $\epsilon < 1/2$ it follows that Equations 4.1 and 4.2 cannot both hold simultaneously. Therefore, we conclude that $\pi(\varepsilon)$ is not ϵ-optimal. Thus, on every timestep t that the agent is in state 1 and $Q(s,l) > Q(s,r)$, we have that its policy is not ϵ-optimal. We have only to show that with probability at least δ the number of such timesteps is greater than some exponential function of the input parameters. In fact, we claim that with probability 1 the number of such timesteps is infinite.

Due to the fact that the action l will be taken infinitely often from state 1, the quantity $Q(1,l)$ will converge to $Q^*(1,l) = \gamma$. Let t be a timestep such that on every timestep after t, $Q(1,l) > \epsilon'$ for $\epsilon' = \gamma/2$. Recall that we've assumed that $Q(2,l) = Q(2,r) = V^*(2) = 1/(1-\gamma)$ and $Q(3,l) = Q(3,r) = V^*(3) = 0$. Now, we have that $0 \le Q(1,r) \le 1/(1-\gamma)$. Consider the act of taking action r from state 1. If a transition to state 2 occurs, then $Q(1,r)$ will be incremented or decremented towards $\gamma/(1-\gamma)$. If a transition to state 3 occurs, $Q(1,r)$ will be decremented towards 0. Let T be a number of timesteps such that if $Q(1,r) = 1/(1-\gamma)$ and the agent experiences T consecutive transitions from state 1 to state 3 after taking action r, then $Q(1,r) < \epsilon'$. Such a T exists because the learning rate is constant. Since state 1 is visited infinitely often, we have that such a sequence of T consecutive transitions from state 1 to state 3 will occur infinitely often after timestep t. However, after each one of these, on the next visit to state 1, we have that $Q(1,r) < \epsilon' < Q(1,l)$, which implies that the agent's policy is $\pi(\varepsilon)$. Since we've shown that the value of $\pi(\varepsilon)$ is not ϵ-optimal, we are done. In summary, if $\varepsilon > 0$, then the sample complexity of Q-learning with a constant learning rate is infinite.

Suppose that $\varepsilon = 0$. Note that for any $0 < x < 1$, we can modify the transition probabilities to yield a new MDP M' that has the same parameters (S, A, γ) as M, such that if we run Q-learning with $\varepsilon = 0$ in M', it is identical to running Q-learning with $\varepsilon = x$ in M. Since we've shown that the latter has infinite sample complexity, this statement implies that the former also has infinite sample complexity. $\square$

When Q-learning is used in practice, the learning rate α is often not kept as a constant. It is usually decreased over time. One common way to decrease it is to allow the learning rate on the tth timestep to be $\alpha_t = (1/t)$. This setting is called a *linear* learning rate. It can be shown that when running Q-learning with ϵ-greedy exploration and a linear learning rate, the agent's action-value estimates will converge to the optimal action values with probability 1 (Watkins & Dayan, 1992). We show next that when this procedure of decreasing α is used along with ϵ-greedy, it results in an algorithm that is not PAC-MDP.

Theorem 10 *(Strehl & Littman, 2006) The Q-learning algorithm with a linear learning rate, optimistic initialization, and ε-greedy exploration is not a PAC-MDP algorithm.*

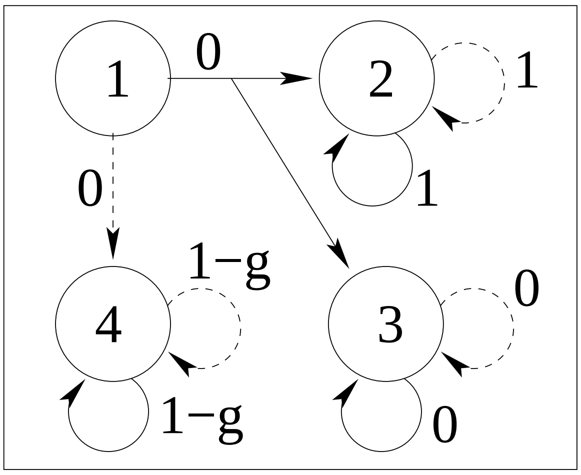


Figure 4.2: The MDP used in the proof of Theorem 9, which proves that Q-learning with random exploration is not PAC-MDP.. The states are represented as nodes and the actions as edges (r is a solid edge and l is a dashed edge). The edge for each action is labeled with its corresponding reward ("g" stands for γ in the figure). The transition probabilities are not shown but are all deterministic except for the split edge r coming from state 1, which leads to state 2 with probability 1/2 and to state 3 with probability 1/2.

Proof Sketch: Consider an MDP M with a single state 1 and action l (see Figure 4.3). The reward for taking the action is 0. In the paper by (Even-Dar & Mansour, 2003), it is shown that when Q-learning with a linear learning rate is used to learn the optimal value function for M, an exponential (in $1/(1-\gamma)$) number of experiences are needed to reduce the value function estimate from its initial value of $1/(1-\gamma)$ to any ϵ satisfying $1 > \epsilon > 0$. By adding an additional action, r, from the state that has small but non-zero reward $1-\gamma$, we obtain an MDP for which Q-learning with a linear learning rate and ε-greedy exploration is not PAC-MDP. To verify this claim, first note that the argument of Lemma 25 holds for Q-learning with a linear learning rate (because with such a learning rate, the action-value estimates will converge to the optimal action values). Thus, we need only consider the case of $\varepsilon \leq \epsilon(1-\gamma)$. Next, we choose ϵ small enough so that the agent's policy is ϵ-optimal only if $Q(r) > Q(l)$. Since the agent will eventually try every action, we know that $Q(l)$ and $Q(r)$ will converge to the optimal value functions $Q^*(l)$ and $Q^*(r)$, respectively (Watkins & Dayan, 1992). Therefore, the agent will continue to experience timesteps for which it follows a non-ϵ-optimal policy until $Q(l) < Q^*(r) = 1$. Of all possible action sequences, taking action a_1 repeatedly is the one that minimizes the number of these timesteps. However, even in this case, we know from the argument in Lemma 39 of (Even-Dar & Mansour, 2003) that an exponential number of such timesteps will occur. □

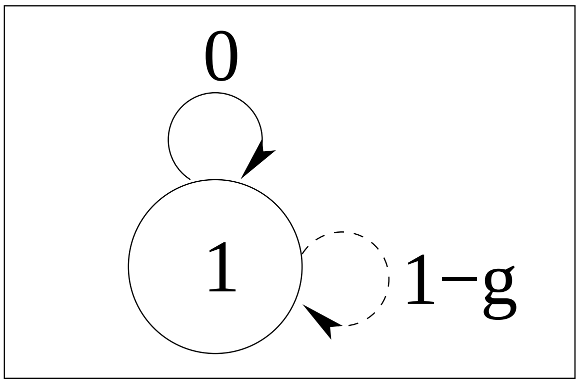


Figure 4.3: The MDP used in the proof of Theorem 10. The states are represented as nodes and the actions as edges (l is a solid edge and r is a dashed edge). The edge for each action is labeled with its corresponding reward ("g" stands for γ in the figure).

In Theorem 9, we argued that the Q-learning algorithm with a constant learning rate and ε-greedy exploration is not PAC-MDP. This argument made use of the well-known fact that Q-learning with a constant learning rate will not converge to the optimal value function. In Theorem 10, we argued that Q-learning with a linear learning rate and ε-greedy exploration is also not PAC-MDP. Here, we made use of the well-known fact that Q-learning with a linear learning rate converges at an exponential rate. In Even-Dar and Mansour (2003), it is shown that Q-learning with a *polynomial learning rate* of the form

$$\alpha_t = 1/(t+1)^w \tag{4.3}$$

where $w \in (1/2, 1)$ and t is the number of timesteps for which the most recent state-action pair has been experienced, converges at a polynomial rate to the optimal value function (assuming that each state-action pair is taken infinitely often). Thus, it is an interesting open problem whether such a learning rate can be used with Q-learning to provide a PAC-MDP algorithm.

Next, we show that the Certainty Equivalence model-based approach to reinforcement learning is not PAC-MDP when combined with ϵ-greedy exploration.

Theorem 11 *(Strehl & Littman, 2006) The Certainty Equivalence Model-based RL algorithm with optimistic initialization and ε-greedy exploration is not a PAC-MDP algorithm.*

Proof: First, we note that the proof of Lemma 25 applies to the Certainty Equivalence approach and thus we consider only the case of $\varepsilon \leq \epsilon(1-\gamma)$.

Consider the following MDP M. M has $n+1$ states s_0, s_1,...,s_n. There are two actions, a_1 and a_2. From s_i, for $i = 1, \ldots, n-1$, action a_1 is deterministic, produces 0 reward, and transitions the agent to state s_{i+1}. From s_i, for $i = 1, \ldots, n-1$, action a_2 is deterministic, produces reward of y, and transitions the agent to state s_0. From s_n both actions are stochastic, producing a reward of 1 with probability z and a reward of 0 with probability $1-z$. They both lead the agent to state s_0. Once in state s_0, learning ends and the agent is placed in state 1. We assume the agent is provided with action-value estimates $Q(s_0, \cdot) = 0$. Such a situation could be simulated to desired accuracy by a more complex MDP (and allowing enough time for $Q(s_0, \cdot)$ to converge), but such an argument is omitted here for simplicity.

Let $\pi(p)$ be the stationary policy in M that takes action a_1 with probability p and takes action a_2 with probability $1-p$ from every state. By solving the Bellman equations, we have that

$$V^{\pi(p)}(s_i) = (p\gamma)^{n-i}z + \frac{1-(p\gamma)^{n-i}}{1-p\gamma}(1-p)y \qquad \text{for } x = 1, \ldots, n. \tag{4.4}$$

Intuitively, if we choose z and y to be constants with $z > y$ and a discount factor close to 1, we can guarantee that near-optimal policies must try to reach the last state s_n. Formally, let $\gamma = (1/2)^{1/(n-1)}$, $z = 3/4$, $y = 1/4$, $\epsilon = 1/11$, $\delta = 1/256$ and $n \geq 10$. Observe that $1/(1-\gamma) = O(n)$, which can be checked by taking the limit of the ratio of n and $1/(1-\gamma)$ as n goes to infinity. Using Equation 4.4 and noting that $y < \gamma^{n-1}z$, we have that the optimal policy is to always take action a_2. Now, we claim that for any value of $p \leq \epsilon(1-\gamma)$, the policy $\pi(p)$ is not ϵ-optimal from state s_1. To prove this claim, first note that

$$V^{\pi(p)}(s_1) \leq p\gamma^{n-1}z + (1-p)y. \tag{4.5}$$

Consider an agent following policy $\pi(p)$ from state s_1. Its first action will be a_1 with probability p and a_2 with probability $1-p$. If it chooses a_1, it will receive zero reward and transition to state s_2. Since the optimal policy is to always take a_1, the most this event can contribute to the value $\pi(p)$, is $\gamma^{n-1}z$, which is the value of the optimal policy. Otherwise, if it chooses a_2, it receives exactly y reward. Now, we prove that

$$V^*(s_1) - p\gamma^{n-1}z + (1-p)y > \epsilon. \tag{4.6}$$

To verify, note that

$$\begin{aligned}
& V^*(s_1) - p\gamma^{n-1}z + (1-p)y \\
&= \gamma^{n-1}z(1-p) + (1-p)y \\
&= (\gamma^{n-1}z - y)(1-p) \\
&= (3/8 - 1/4)(1-p) \\
&\geq (1/8)(1-\epsilon(1-\gamma)) \\
&= (1/8)(109/110) > \epsilon.
\end{aligned}$$

In the fourth step, we used the fact that $p < \epsilon(1-\gamma)$. Now, combining Equation 4.5 and Equation 4.6, we have that $V^*(s_1) - V^{\pi(p)} > \epsilon$ when $p < \epsilon(1-\gamma)$.

Suppose we run the Certainty Equivalence RL algorithm with ϵ-greedy exploration and optimistic initialization in the MDP M. Since the MDP is deterministic in all states except s_n, and we've used optimistic initialization, the agent will first explore every action from every state at least once. It will thus learn the entire MDP except for the reward function for state s_n. It will try both actions from state s_n at least once. Suppose that it observes reward 0 from state s_n for both these actions. This event will happen with probability $(1-z)^2 = 1/16 = \sqrt{\delta}$. At this point, the agent's model says that the reward for either action from state s_n is 0. Thus, the optimal policy for the model will be to take action a_2 from every state. Since it is using ε-exploration, it will follow policy $\pi(\varepsilon)$ until it reaches state s_n and learns more about the reward function. From our above argument and the fact that $\varepsilon \leq \epsilon(1-\gamma)$, we have that the agent's policy is not ϵ-optimal from state s_1. Next, we show that with high probability it will visit state s_1 an exponential number of times before updating its model.

Let a trial consist of the agent starting in state s_1 and following policy $\pi(\varepsilon)$ until it reaches state s_0. We say that the trial is successful if it reaches state s_n. The probability of a single trial being a success is at most $\varepsilon^{n-1} \leq (\epsilon(1-\gamma))^{n-1} \leq (1/11)^{n-1}$. Hence, the expected number of trials is exponential. Using standard arguments, we can also show that with probability at least $\sqrt{\delta}$, the agent will require an exponential (in n) number of trials to obtain a single success. To summarize, we've shown that with probability at least δ the agent will have exponential sample complexity. Therefore, it is not PAC-MDP. $\square$

4.2 PAC-MDP Algorithms and Convergent Algorithms

There has been a great deal of theoretical work analyzing RL algorithms. Most of the early results include proving that under certain conditions various algorithms will, in the limit of infinite experience, compute the optimal value function from which the optimal policy can be extracted (Watkins & Dayan, 1992; Bertsekas & Tsitsiklis, 1996). These convergence results make no performance guarantee after only a finite amount of experience. They also ignore the problem of exploration by assuming that every state and action is taken infinitely often.

An interesting avenue of research is to study the convergence rates of various RL algorithms. Even-Dar and Mansour (2003) studied, for example, the convergence rate of Q-learning. They showed that, under certain conditions, Q-learning converges to a near-optimal value function in a polynomial number of timesteps.

A PAC-MDP algorithm does not, by definition, need to be a convergent algorithm.

In fact, all of the algorithms proven to be PAC-MDP to date are not convergent algorithms. It may be possible to modify any PAC-MDP algorithm to construct a convergent PAC-MDP algorithm, for example by adding ϵ-greedy exploration. However, doing so would require new ideas, as the current PAC-MDP proofs all rely on the fact that the agent's policy changes only finitely many times, while most provably convergent algorithms don't exhibit this property.

4.3 Reducing the Total Computational Complexity

In each of the algorithms, we've assumed that the action values are stored in a priority queue. The use of this data structure results in a cost of $\Theta(\ln(A))$ each time an action value is updated. If there are Z total updates during the entire run of an algorithm, then the associated cost may be as high as $\Theta(Z \ln(A))$. Several of the algorithms we've discussed (RTDP-RMAX, RTDP-IE, Delayed Q-learning, Delayed Q-learning with IE), have the special property that for a fixed state, the action-value estimates are non-increasing over time. In this section, we show that for these algorithms, the total cost can be made to be $\Theta(\frac{1}{(1-\gamma)^2\epsilon})$. This observation likely results in an improvement on the computational complexity bounds (but a corresponding increase in space complexity) for these algorithms, since our bounds for Z are not smaller than $\Theta(\frac{1}{(1-\gamma)^2\epsilon^2})$.

The idea is very simple. First, note that the action value estimates are bounded between 0 and $1/(1-\gamma)$, and are non-increasing. We modify each of the algorithms as follows. For some timestep t, let a^* denote the greedy action, $a^* = \text{argmax}_a Q(s_t, a)$. Instead of requiring the algorithms to choose this greedy action, we allow them to choose any action a' such that $Q(s_t, a') \geq Q(s_t, a^*) - C\epsilon(1-\gamma)$, for some constant C. We call an algorithm "nearly greedy" if it uses such a strategy. Using Proposition 1, it is easy to see that Theorem 1 can be modified to allow for algorithms that choose actions that are nearly greedy. Given this modification, our implementation is as follows. At the beginning of the algorithm, $1/(C\epsilon(1-\gamma)^2)$ buckets (for each state) are created that can hold pointers to the A action values. The first bucket represents the interval $[0, C\epsilon(1-\gamma)]$, the second $(C\epsilon(1-\gamma), 2C\epsilon(1-\gamma)]$, and so on. The ith bucket contains a pointer to the action-value estimate $Q(s, a)$ if and only if $Q(s, a)$ lies in the interval $((i-1)C\epsilon(1-\gamma), iC\epsilon(1-\gamma)]$. If more than one action value lie in the same interval, then a list of the pointers is maintained. It is easy to see that this data structure can support inserts and removals in constant time. In addition, we maintain a pointer to the bucket that contains the maximum valued estimate. Since the action values are non-increasing, the number of operations, during the entire execution of the algorithm, required to maintain this pointer is equivalent to the number of buckets $(O(1/(\epsilon(1-\gamma)^2)))$. This bound is amortized and will negatively impact both the space and per-timestep computational time complexities.

4.4 On the Use of Value Iteration

All of our algorithms involved learning action-value estimates. Many of them do so by solving an internal model. The others can be viewed as incrementally solving an internal model. In all cases, we have used value iteration or partial value iteration (for the incremental algorithms) to solve this internal model. For the complete model-based methods (MBIE-EB, E^3 and R-MAX), any technique for solving MDPs can be used. For instance, policy iteration may be used. Also, in particular, if the discount factor is very close to 1, it may be beneficial to use a fast linear programming algorithm rather than value iteration or policy iteration. For the other algorithms, the value-iteration techniques are fundamental and there is no obvious way to modify them to use other MDP solution techniques.

Chapter 5
Empirical Evaluation

To better quantify the relationship between them, we performed three sets of experiments with each of the algorithms. For each experiment, we recorded the cumulative reward obtained (to measure sample complexity) and Bellman backup computations required[1] (to measure computational complexity) by the agent over a fixed number of timesteps.

Our main working hypotheses are

- Compared to the naïve approach, the Interval Estimation approach to exploration results in a better use of experience. This statement is verified by the fact that when two algorithms are compared that only differ in their exploration strategy (for instance, RTDP-IE and RTDP-RMAX), the one using IE obtains higher cumulative reward. The Bandit MDP domain was specifically chosen to highlight the benefits of the IE approach.

- The incremental model-based approaches (RTDP-RMAX and RTDP-IE) benefit from maintaining a model while using much less computation than algorithms that solve the model (R-MAX and MBIE-EB). Our experiments show that this hypothesis is not universally true but is true most of the time. For instance, there is only one domain (Bandit MDP) where a model-free algorithm obtained greater cumulative reward than RTDP-RMAX. Also, RTDP-IE was never outperformed, in terms of reward, by a model-free algorithm. The total computational cost of the incremental algorithms were lower than their non-incremental counterparts

[1]Formally, we counted the number of multiplication operations used in the Bellman backup.

except in the Bandit MDP domain.[2]

The Q-learning algorithm we experimented with uses optimistic initialization and ε-greedy exploration (see Section 3.1). The learning rate was chosen according to the paper by Even-Dar and Mansour (2003) with parameters $1/2 \leq w \leq 1$. Specifically, the learning rate for a state-action pair experienced n times is

$$\alpha = 1/(n+1)^w. \tag{5.1}$$

Surprisingly, the optimal setting of w within this range for our experiments was either 1/2 or 1. We also tried other constant (less than or equal to 1) multiples of this learning rate and found no improvement. It is worth noting that, consistent with the analysis, we also found that the algorithm always performed best with $\varepsilon = 0$.

All algorithms except Q-learning have a parameter (usually called ϵ_1) that determines the minimum allowable difference in two successive action-value estimates. For consistency, we have set this parameter to be fixed at the value of 10^{-6}. The algorithms that use IE as their method for exploration have the property that as their parameter m is increased,[3] the performance (in terms of cumulative reward obtained) of the algorithm typically increases as well. Thus, we set this parameter to $m = 100,000$ for the incremental algorithms RTDP-IE and DQL-IE. It was computationally infeasible to use this large of a value of m for MBIE-EB so $m = 10,000$ was used. Each algorithm used a discount factor of $\gamma = 0.95$.

5.1 Bandit MDP

The first set of experiments consisted of an MDP similar to the k-armed bandit problem(Berry & Fristedt, 1985) with $k = 6$, except that the noise of the arms was modeled

[2]This does not contradict our claims that the incremental versions have lower computational complexity, because those claims apply to the worst-case per-step complexity rather than the total amortized complexity, which is what was measured in the experiments.

[3]For MBIE-EB and RTDP-IE, m is the maximum number of samples per state-action pair that is used for estimating the model. For Delayed Q-learning with IE, m is the maximum number of successive attempted updates for a single state-action pair the algorithm will attempt until it gives up.

in the transition function rather than the reward function. Specifically, there were 7 states ($S = \{0, \ldots, 6\}$), with 0 as the start state and 6 actions. Taking action $j \in \{1, \ldots, 6\}$ from state 0 results in a transition to state j with probability $1/j$ and a transition back to state 0 with probability $1 - 1/j$. From state $i > 0$, under each action, the agent is transitioned to state 0. Choosing action 1 from any state $i > 0$ results in a reward of $(3/2)^i$ (all other rewards are 0). These dynamics were created so that it is better to choose the action with the lowest payoff probability (leaving state 0). To recap, from state 0, each action behaves like pulling an arm. The arm "pays off" if the agent is transitioned away from state 0. Once in another state, the agent is free to choose action 1 and obtain non-zero reward. A single experiment consisted of running a given algorithm in the MDP for 100,000 timesteps. Each experiment was repeated 1000 times and the results averaged. For each algorithm, the parameters were chosen by running the algorithm multiple times beforehand with different parameter settings (chosen by a uniform grid) and choosing the parameter values that maximized the average cumulative reward (over 100 experiments per parameter setting).[4]

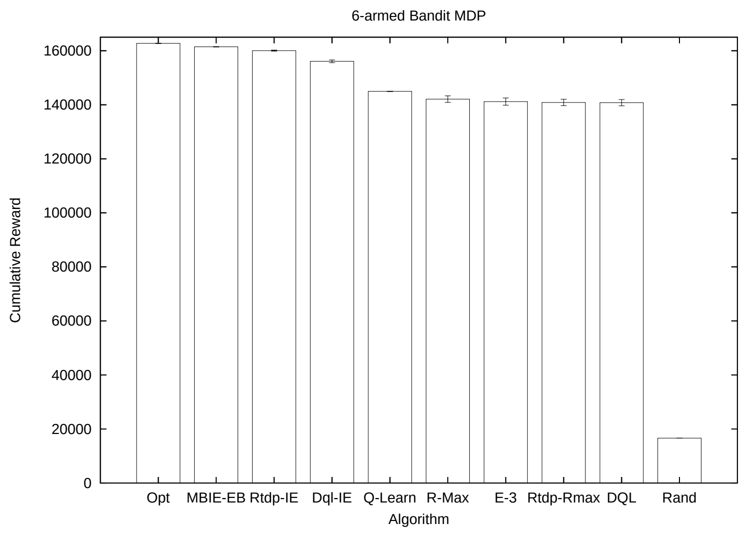


Figure 5.1: Results on the 6-armed Bandit MDP

Figure 5.1 displays the average cumulative reward of each algorithm. In Table 5.1, we provide the parameter settings of each algorithm along with the average cumulative

[4]This method of evaluation may favor algorithms that perform very well for a small range of parameters over more robust algorithms that perform well over a larger range.

Algorithm	Param	Reward	Backups
Optimal	—	$163 \cdot 10^3$	0
MBIE-EB	$\beta = 7.69$	$161 \cdot 10^3$	$397 \cdot 10^5$
RTDP-IE	$\beta = 0.34$	$160 \cdot 10^3$	$543 \cdot 10^1$
DQL-IE	$\beta = 0.9$	$156 \cdot 10^3$	$100 \cdot 10^2$
Q-learning	$w = 0.5$	$145 \cdot 10^3$	$100 \cdot 10^2$
RMAX	$m = 34$	$142 \cdot 10^3$	$160 \cdot 10^1$
E-Cubed	$(m, thresh) = (25, 0.167)$	$141 \cdot 10^3$	$220 \cdot 10^1$
RTDP-RMAX	$m = 34$	$141 \cdot 10^3$	$387 \cdot 10^1$
DQL	$m = 5$	$141 \cdot 10^3$	$108 \cdot 10^1$
Random	—	$17 \cdot 10^3$	0

Table 5.1: Results in the Bandit MDP. The algorithms are listed in decreasing order by average cumulative reward (third column). Only three significant digits are displayed. Within this range, all differences were statistically significant.

reward and number of Bellman backup computations used by the algorithm.

First, consider the average cumulative reward achieved by the algorithms. This domain illustrates the advantage of using the Interval Estimation approach over the naïve approach to exploration, even when the uncertainty of the domain appears in the transition distribution rather than the reward distribution (as it does in the simpler K-armed bandit problem). Since the sequential nature of this problem is close to minimal, the advantage of the model-based approaches over the incremental approach is small in this problem. When we consider the computation time (measured by the Bellman Backups in the table above) we see that MBIE-EB, in particular, is very slow compared to the other algorithms. In general, the IE approach required more computation than the naïve approach and the algorithms that solved their internal models took more computation than the incremental algorithms. However, we note that the parameters for each algorithm were optimized to yield maximum reward ignoring computation time. We could, for example, reduce the computation time of RTDP-IE and DQL-IE (short for Delayed Q-learning with IE)[5] to almost any value by using a smaller value for m. Finally, we note that each of the algorithms (except the algorithm that chose actions uniformly at random) found the optimal arm and pulled that one most often.

[5]Since m was set to be as large as the number of timesteps, DQL-IE necessarily performs a single Bellman backup computation per timestep.

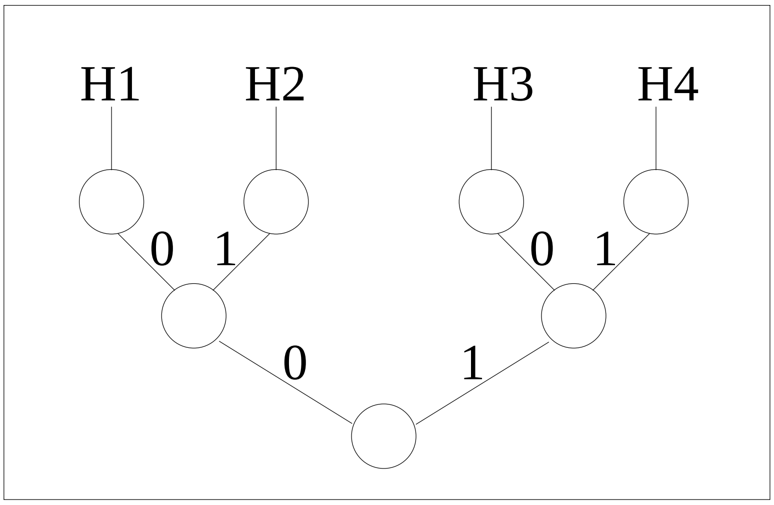


Figure 5.2: Hallways MDP diagram

5.2 Hallways MDP

The Hallways domain consists of four hallways, each of length ten, and two actions, 0 and 1. There is an initial start state s_0. From that state the first two action choices determine the hallway that the agent chooses to go down (00 for hallway H1, 01 for hallway H2, 10 for hallway H3, and 11 for hallway H4). Figure 5.2 shows a rough diagram of this domain where states are represented as nodes and actions are represented as edges labeled by action ID. Once in a hallway, action 0 moves the agent forward and action 1 leaves the agent in the same state. When at the end of the hallway, the agent may return to the state s_0 by performing action 0. All rewards are zero except on those transitions that take an agent from the end of hallway Hi to state s_0 under action 0, for $i \in \{1, 2, 3, 4\}$. The reward for this transition is stochastic: with probability $(i - 1)/i$ the reward is 0 and with probability $1/i$ the reward is $(3/2)^{i+5}$. This reward was chosen so that the optimal policy is always to go down hallway 4 but this is also the least likely to result in non-zero reward (so that exploration is necessary). In total there are 47 states. This domain was chosen because it combines the stochastic reward element of the K-armed bandit problem with the necessity of relating the value each state to the value of its successor.

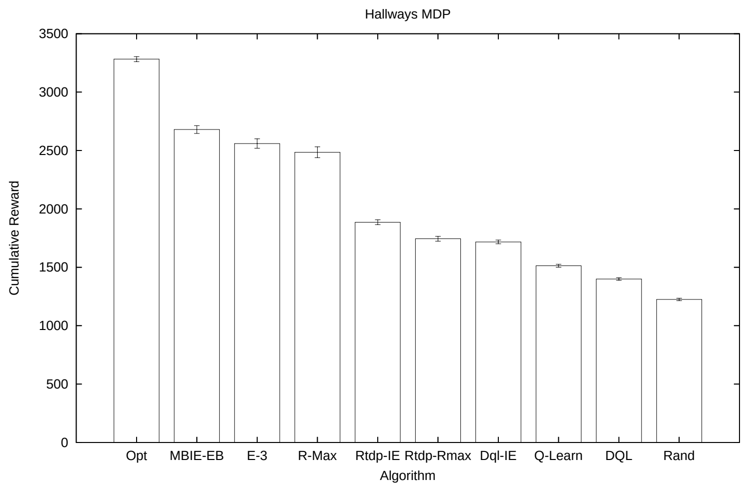


Figure 5.3: Results on the Hallways MDP

Figure 5.3 displays the average cumulative reward of each algorithm. In the following table, we provide the parameter settings of each algorithm along with the average cumulative reward and number of Bellman backup computations used by the algorithm.

Algorithm	Param	Reward	Backups
Optimal	—	$33 \cdot 10^2$	0
MBIE-EB	$\beta = 0.6$	$27 \cdot 10^2$	$79 \cdot 10^6$
E-Cubed	$(m, thresh) = (8, 0.82)$	$26 \cdot 10^2$	$64 \cdot 10^3$
RMAX	$m = 7$	$25 \cdot 10^2$	$53 \cdot 10^3$
RTDP-IE	$\beta = 0.35$	$19 \cdot 10^2$	$10 \cdot 10^3$
RTDP-RMAX	$m = 13$	$17 \cdot 10^2$	$88 \cdot 10^2$
DQL-IE	$\beta = 0.17$	$17 \cdot 10^2$	$10 \cdot 10^3$
Q-learning	$w = 1$	$15 \cdot 10^2$	$10 \cdot 10^3$
DQL	$m = 1$	$14 \cdot 10^2$	$10 \cdot 10^3$
Random	—	$12 \cdot 10^2$	0

Table 5.2: Results in the Hallways MDP. The algorithms are listed in decreasing order by average cumulative reward (third column). Only two significant digits are displayed. Within this range, all differences were statistically significant except the reward values for the E-Cubed and RMAX rows.

Due to the highly sequential nature of the MDP, the full model-based algorithms (MBIE-EB, E^3, and R-MAX) performed better than the incremental algorithms. Given this, the IE approach to exploration generally performed better than the naïve approach.

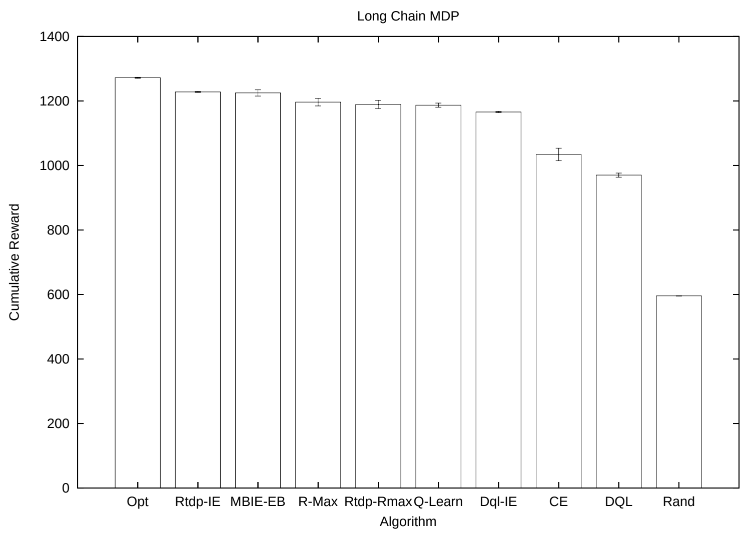


Figure 5.4: Results on the Long Chain MDP

5.3 LongChain MDP

The final MDP we tested is an approximation of the MDP used in the proof of Theorem 11, with 20 states. Instead of using an episodic MDP as described in the proof we add a transition from the terminal state s_0 to the start state with probability 0.025 (under both actions, with probability $1 - 0.025$, the agent stays in the terminal state). Since this MDP was used to prove that the Certainty Equivalence approach was not PAC-MDP, we also tested this algorithm along with the others.

Figure 5.4 displays the average cumulative reward of each algorithm (the results for E^3 were eliminated to make room for the results of the Certainty Equivalence (CE) method, but they appear in the table). In the following table, we provide the parameter settings of each algorithm along with the average cumulative reward and number of Bellman backup com putations used by the algorithm.

As expected, the Certainty-Equivalence approach performed poorly in this domain, even though it computes an extremely accurate model of the transitions for the region of the MDP it visits.

Algorithm	Param	Reward	Backups
Optimal	—	$127 \cdot 10^1$	0
RTDP-IE	$\beta = 0.068$	$123 \cdot 10^1$	$549 \cdot 10^2$
MBIE-EB	$\beta = 0.16$	$123 \cdot 10^1$	$403 \cdot 10^5$
RMAX	$m = 2$	$120 \cdot 10^1$	$767 \cdot 10^1$
E-Cubed	$(m, thresh) = (2, 0.4)$	$119 \cdot 10^1$	$758 \cdot 10^1$
RTDP-RMAX	$m = 2$	$119 \cdot 10^1$	$250 \cdot 10^1$
Q-learning	$w = 1$	$119 \cdot 10^1$	$100 \cdot 10^2$
DQL-IE	$\beta = 2.33$	$117 \cdot 10^1$	$100 \cdot 10^2$
Certainty-Equivalence (CE)	—	$103 \cdot 10^1$	$140 \cdot 10^5$
DQL	$m = 10$	$970 \cdot 10^0$	$272 \cdot 10^1$
Random	—	$596 \cdot 10^0$	0

Table 5.3: Results in the LongChain MDP. The algorithms are listed in decreasing order by average cumulative reward (third column). Only three significant digits are displayed. Within this range, all differences were statistically significant except the number of backups (column 4) for the E-Cubed and RMAX rows.

5.4 Summary of Empirical Results

Reinforcement-learning algorithms are extremely difficult to test experimentally. The sample complexity, as we've defined it, is impossible to compute and difficult (computationally) to estimate. We chose to measure cumulative reward instead since it is the ultimate goal of an RL agent to maximize reward. Unfortunately, this measure is extremely noisy and therefore each experiment had to be repeated many times to yield accurate measures. Also, several of the algorithms (MBIE-EB, in particular) were very slow. Thus, we were limited to domains with very small state and action spaces. Given these issues, the experimental results do not provide the complete story behind the performance of the algorithms (in fact, no set of experiments could). However, several observations are clear. The first is that the algorithms that use the Interval Estimation approach to exploration consistently outperform those that use the naïve approach when we ignore computation time. In particular, MBIE-EB is superior to all other algorithms by this measure.[6] In two out of three domains, the incremental model-based approaches (RTDP-IE and RTDP-RMAX) used much less computation

[6]We have not displayed the results, but we also tested the original MBIE algorithm from Section 2.5. This algorithm consistently provided a minor increase in performance against MBIE-EB in terms of cumulative reward, but used even more computation time.

than their non-incremental versions but often achieved almost as much cumulative reward. In fact, all the algorithms did fairly well on most domains in terms of obtaining reward. Many of the new algorithms achieved results better than that of Q-learning. This suggests that the theoretical results have some practical implications. One notable exception appears to be the Delayed Q-learning algorithm. We see here that waiting for m visits to a state-action pair before any update appears detrimental even though it is sufficient for the asymptotic PAC-MDP sample complexity bounds. This certainly suggests that Delayed Q-learning is not an extremely practical algorithms, which is not entirely surprising. However, the Delayed Q-learning with IE algorithm is more practical and also comes with the same theoretical guarantees.

Chapter 6

Extensions

The main drawback of the algorithms presented in this thesis is that their sample complexities scale polynomially with the size of the state and action spaces, respectively. This quality is clearly necessary because an agent cannot learn about states it has never visited. However, many RL algorithms make implicit or explicit assumptions about their domains that allow an agent to generalize across various dimensions. The most well-studied type of generalization is where a function approximator is trained to approximate the value of each state (Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996). In the following two sections we discuss a different type of generalization, namely where we try to learn the transition and reward distributions and generalize across states and actions.

6.1 Factored-State Spaces

Definition 13 *A* **factored-state MDP** *(or* **f-MDP***) is an MDP where the states are represented as vectors of n components $X = \{X_1, X_2, \ldots, X_n\}$. Each component X_i (called a* **state variable** *or* **state factor***) may be one of finitely many values from the set $\mathcal{D}(X_i)$. In other words, each state can be written in the form $x = \langle x_1, \ldots, x_n \rangle$, where $x_i \in \mathcal{D}(X_i)$.*

The definition of factored-state MDPs is motivated by the desire to achieve learning in very large state spaces. The number of states of a factored-state MDP M is *exponential* in the number n of state variables.

6.1.1 Restrictions on the Transition Model

Factored-state MDPs are most useful when there are restrictions on the allowable transition functions. Traditionally, researchers have studied transition models that can be represented by *dynamic Bayesian networks (DBNs)* for each action of the MDP. Such a representation has been shown to be powerful enough to support fast learning (Kearns & Koller, 1999). However, this representation is unable to model some important conditional independencies. Therefore, we develop a more general and more powerful model. For any factored state x, let x_i denote the ith component of x for $i = 1, \dots, n$. Next, we introduce a model that yields a compact transition representation by allowing the transition probabilities for the factors of the next state x', $P(x'_i|x, a)$, to depend on only a subset of the state factors of the current state x.

Assumption 1 *Let x, x' be two states of a factored-state MDP M, and let a be an action. The transition distribution function satisfies the following conditional independence condition:*

$$T(x'|x, a) = \prod_i P(x'_i|x, a). \tag{6.1}$$

This assumption ensures that the values of each state variable after a transition are determined independently of each other, conditioned on the previous state. We consider transition functions that are structured as follows.

Definition 14 *Let $\mathcal{I}$ be a set of* **dependency identifiers**.

Definition 15 *Let $D : \mathcal{S} \times \mathcal{A} \times X \to \mathcal{I}$ be a* **dependency function**.

Assumption 2 *Let $s, s' \in \mathcal{S}$ be two states and $a \in \mathcal{A}$ be an action. We assume that $P(s'_i|s, a) = P(s'_i|D(s, a, X_i))$. Thus, the transition probability from (s, a) to s' can be written as*

$$T(s'|s, a) = \prod_{i=1}^{n} P(s'_i|D(s, a, X_i)) \tag{6.2}$$

The dependency function approach yields a compact representation of the underlying transition function by allowing commonalities among component distributions with shared dependency function behavior. It also generalizes other approaches, such as those using dynamic Bayes networks (Kearns & Koller, 1999), and those incorporating decision trees (Boutilier et al., 1996) to represent abstraction. Several important definitions follow.

Definition 16 *For each $(s, a) \in \mathcal{S} \times \mathcal{A}$, let*

$$D_{s,a} := \{(X_i, j) \in X \times \mathcal{I} \mid j = D(s, a, X_i)\} \tag{6.3}$$

be the **relevant dependency pairs for state-action pair** (s, a).

Definition 17 *Let $\mathcal{Q} = \cup_{(s,a)\in\mathcal{S}\times\mathcal{A}} D_{s,a}$ be the set of all* **transition components**. *Let N denote $|\mathcal{Q}|$, the number of transition components.*

Note that each transition component, (X_i, j), corresponds to an independent probability distribution over the set $\mathcal{D}(X_i)$ that potentially needs to be estimated by the agent. We will provide algorithms whose sample complexity depends only linearly on $\sum_{(X_i,j)\in\mathcal{Q}} |\mathcal{D}(X_i)|$, the number of parameters of the compact representation.

6.1.2 Factored Rmax

Rmax is a reinforcement-learning algorithm introduced by (Brafman & Tennenholtz, 2002) and shown to have PAC-MDP sample complexity by (Kakade, 2003; Brafman & Tennenholtz, 2002) showed it was PAC-MDP in a slightly different setting). Factored Rmax (or **f-Rmax**) is the direct generalization to factored-state MDPs (Guestrin et al., 2002). Factored Rmax is model-based, in that it maintains a model M' of the underlying f-MDP, and at each step, acts according to an optimal policy of its model.

To motivate the model used by Factored Rmax, we first describe at a high level the main intuition of the algorithm. Consider a fixed state factor X_i and dependency identifier $j \in \mathcal{I}$ such that $D(s, a, X_i) = j$ for some state s and action a. Let $x_i \in \mathcal{D}(X_i)$ be any value in the domain of X_i. There exists an associated probability $P(x_i|j)$. We call the corresponding distribution, $P(\cdot|j)$ for (X_i, j), a *transition component*, which is defined formally in Section 6.1.1. The agent doesn't have access to this distribution however, and it must be learned. The main idea behind model-based approaches for f-MDPs is to use the agent's experience to compute an approximation to the unknown distribution $P(\cdot|j)$. However, when the agent's experience is limited, the empirical distribution often produces a very poor approximation. The trick behind f-Rmax is to use the agent's experience only when there is enough of it to ensure decent accuracy, with high probability.

Let $m = (m_1, \ldots, m_n)$ be some user-defined vector of positive integers that is provided to f-Rmax as input at the beginning of a run. For each transition component (X_i, j), f-Rmax maintains a count $n(X_i, j)$ of the number of times it has taken an action a from a state s for which $D(s, a, X_i) = j$. For a given state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, $D_{s,a}$ is the set of relevant dependency identifies $j \in \mathcal{I}$ such that $j = D(s, a, X_i)$ for some factor X_i. If the associated counts, $n(X_i, j)$, are each at least m_i, respectively, then we say that (s, a) is a *known* state-action pair and use the reward distribution and the empirical transition distribution estimate for (s, a). Otherwise, the f-Rmax agent assumes that the value of taking action a from state s is $1/(1 - \gamma)$, the maximum possible value.

Intuitively, the model used by f-Rmax provides a large exploration bonus for reaching a state-action pair that is unknown (meaning it has some relevant transition component X_i that has not been experienced m_i times). This will encourage the agent to increase the counts $n(X_i, j)$, causing effective exploration of the state space until many transition components are known. After much experience, the empirical distribution is used, and the agent acts according to a near-optimal policy. We will show that if the m_i are set large enough, but still polynomial in the relevant quantities (with the number of states being replaced by the number of transition components), then factored Rmax is PAC-fMDP in the sample complexity framework.

Formally, Factored Rmax solves the following set of equations to compute its state-action value estimates:

$$\begin{aligned} Q(s,a) &= 1/(1-\gamma), \text{if } \exists X_i,\ n(D(s,a,X_i)) < m_i \\ Q(s,a) &= R(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} Q(s',a'), \end{aligned}$$

otherwise.

6.1.3 Analysis of Factored Rmax

The main result of this section is the following theorem, which implies that f-Rmax is PAC-fMDP.

Theorem 12 *Suppose that $0 \leq \epsilon < \frac{1}{1-\gamma}$ and $0 \leq \delta < 1$ are two real numbers and $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ is any factored-state MDP with dependency function D and dependency identifiers $\mathcal{I}$. Let n be the number of state factors and $\mathcal{Q}$ be the set of transition components with $N = |\mathcal{Q}|$. There exists inputs $m = (m_1, \ldots, m_n)$ satisfying $m_i = m_i(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{n^2(|\mathcal{D}(X_i)| + \ln(N/\delta))}{\epsilon^2(1-\gamma)^4}\right)$, such that if f-Rmax is executed on M with inputs m, then the following holds. Let $\mathcal{A}_t$ denote f-Rmax's policy at time t and s_t denote the state at time t. With probability at least $1-\delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ is true for all but*

$$O\left(\frac{n^2(\Psi + N\ln(N/\delta))}{\epsilon^3(1-\gamma)^6} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)}\right), \tag{6.4}$$

timesteps t, where $\Psi = \sum_{(X_i,j)\in\mathcal{Q}} |\mathcal{D}(X_i)|$.

Factored Rmax models the unknown environment using the certainty-equivalence method.

Certainty-Equivalence Model

Let $(X_i, j) \in \mathcal{Q}$ be a fixed transition component that assigns probabilities, $P(x_k|j)$, for all $x_k \in \mathcal{D}(X_i)$. The maximum likelihood estimate of these probabilities from m_i samples has the following formula:

$$\hat{P}(x_k|j) = \frac{\#\ of\ samples\ equal\ to\ x_k}{\#\ of\ samples = m_i}. \tag{6.5}$$

We note that a learning algorithm obtains samples for (X_i, j) whenever it takes an action a from a state s for which $(X_i, j) \in D_{s,a}$. The *empirical model* (also called the *certainty-equivalence model*) is the transition model defined by using the maximum likelihood estimates for each transition component:

$$\hat{T}(s'|s,a) = \prod_{i=1}^{n} \hat{P}(s'_i|D(s,a,X_i)). \tag{6.6}$$

Analysis Details

We utilize Theorem 1 to prove that f-Rmax is PAC-MDP. The key insight is that after an adequate number of samples have been gathered for a given transition component, the resulting empirical transition probability distribution is close to the true one, with high probability. However, transitions in a factored-state MDP involve n transition components since there are n state factors, all of whose transitions are independently computed (see Equation 6.1). Next, we relate accuracy in the transition model with accuracy in the transition components.

We seek to bound the L_1 distance between an approximate transition distribution of the factored-state MDP to the true transition model.

Lemma 26 *Let $X = \langle x(i,j)\rangle$ and $Y = \langle y(i,j)\rangle$ be any two stochastic matrices of size $m \times n$. Let $x(\cdot,j)$ denote the jth column of the matrix X, which is a discrete probability distribution over m elements. For stochastic matrix X, let P_X denote the product distribution of size m^n defined as $P_X(i_1,\ldots,i_n) = x(i_1,1)\cdots x(i_m,m)$. If*

$$||x(\cdot,j) - y(\cdot,j)||_1 \le 2\epsilon \;\text{ for all } j = 1,\ldots,n,$$

for $0 \le \epsilon \le 1$, then

$$\sum_{i_1=1}^{m} \cdots \sum_{i_n=1}^{m} |P_X(i_1,\ldots,i_n) - P_Y(i_1,\ldots,i_n)| \le 2 - 2(1-\epsilon)^n$$

Proof: Using the transformation $y(i,j) = x(i,j) + \alpha(i,j)$, we seek to maximize the function $f(X, \alpha(\cdot,\cdot)) := \sum_{i_1=1}^{m} \cdots \sum_{i_n=1}^{m} |x(i_1,1)\cdots x(i_n,n) - (x(i_1,1)+\alpha(i_1,1))\cdots(x(i_n,n)+$

$\alpha(i_n, n))|$, under the lemma's constraints. Fix any column index j and consider the partial derivative with respect to the ith component, $\partial f/\partial x(i,j)$. It is clear that this derivative does not depend[1] on $x(\cdot, j)$. Suppose there is an element in the jth column, $x(i,j)$, such that $\epsilon < x(i,j) < 1-\epsilon$. Then there must be another distinct element $x(i',j)$ such that $x(i',j) > 0$. Without loss of generality, suppose that $\partial f/\partial x(i,j) \geq \partial f/\partial x(i',j)$. The value of f will not decrease if we simultaneously increase $x(i,j)$ and decrease $x(i',j)$ by as much as possible (until $x(i,j)+\alpha(i,j) = 1$ or $x(i',j)+\alpha(i',j) = 0$). By a similar argument if there are two or more nonzero elements in the jth column that add up to ϵ, then we can increase the one with largest partial derivative to ϵ and decrease the others to zero. In conclusion, we can restrict ourselves to matrices X whose columns are one of the two following forms: (1) there is one element with value $1-\epsilon$ and another one with value ϵ, or (2) there is a single element with value one and the rest are zero-valued.[2] By symmetry, if column j of matrix X is of form (1) with indices i_1, i_2 such that $x(i_1, j) = 1-\epsilon$, $x(i_2, j) = \epsilon$, then column j of matrix Y is of form (2) with $\alpha(i_1, j) = \epsilon$ and $\alpha(i_2, j) = -\epsilon$, and vice versa.

We have shown that we can restrict the maximization of f over stochastic matrices X and Y of the following form:

$$X = \begin{pmatrix} 1-\epsilon & \dots & 1-\epsilon & 1 & \dots & 1 \\ \epsilon & \dots & \epsilon & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \end{pmatrix}$$

$$Y = \begin{pmatrix} 1 & \dots & 1 & 1-\epsilon & \dots & 1-\epsilon \\ 0 & \dots & 0 & \epsilon & \dots & \epsilon \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \end{pmatrix}$$

[1]The absolute value signs can be removed by noting that there is some setting of $\alpha(\cdot,\cdot)$ that maximizes f.

[2]Technically, we have left open the possibility that one element has value $1-\epsilon'$ and another has value ϵ', where $0 < \epsilon' < \epsilon$. However, it is easy to show that we can increase f in such a case.

Suppose there are t_1 columns of type (1) and $t_2 = n - t_1$ columns of type (2) in X. Then, we have that

$$f(X,Y) = |(1-\epsilon)^{t_1} - (1-\epsilon)^{t_2}| + (1 - (1-\epsilon)^{t_1}) + (1 - (1-\epsilon)^{t_2}). \tag{6.7}$$

The first term in Equation 6.7 follows from choosing elements only in the first row ($i_1 = \cdots = i_n = 1$). The second term results from choosing non-zero terms in X that are zero in Y and the third term from choosing non-zero terms in Y that are zero in X. Without loss of generality, if $t_1 > t_2$, then from Equation 6.7 we have that $f(X,Y) = 2 - 2(1-\epsilon)^{t_1} \leq 2 - 2(1-\epsilon)^n$, as desired. □

Corollary 2 *Let M be any factored-state MDP. Suppose that for each transition component $P(\cdot|j)$ we have an estimate $\hat{P}(\cdot|j)$ such that $||P(\cdot|j) - \hat{P}(\cdot|j)||_1 \leq \epsilon/n$. Then for all state-action pairs (s,a) we have that*

$$||T(s,a) - \hat{T}(s,a)||_1 \leq \epsilon. \tag{6.8}$$

Proof: Follows directly from the fact that $1 - (1-\epsilon)^n \leq \epsilon n$. □

Corollary 2 shows that L_1 accuracy of the transition components of the model implies L_1 accuracy of the resulting empirical transition distributions. Lemma 4 shows that L_1 accuracy in the transition distributions of a model combined with the true reward distributions can be used to compute accurate value functions for any policy. Thus, we are left with answering the question: *how many samples are needed to estimate a given transition component (discrete probability distribution) to a desired L_1 accuracy.* The following theorem is helpful in this matter.

Theorem 13 *(from (Weissman et al., 2003)) Let P a probability distribution on the set $\mathcal{A} = \{1, 2, \ldots, a\}$. Let $X_1, X_2, \ldots, X_m$ be independent identically distributed random variables according to P. Let $\hat{P}$ denote the empirical distribution computed by using the X_i's. Then for all $\epsilon > 0$,*

$$Pr(||P - \hat{P}||_1 \geq \epsilon) \leq 2^a e^{-m\epsilon^2/2}. \tag{6.9}$$

Definition 18 *For f-Rmax we define the "known" state-action pairs K_t, at time t, to be*

$$K_t := \{(s, a) \in \mathcal{S} \times \mathsf{A} | n(X_i, j) \geq m_i \text{ for all } (j, X_i) \in D_{s,a}\}. \tag{6.10}$$

If t is contextually defined, we use the simpler notation K.

The following event will be used in our proof that f-Rmax is PAC-fMDP. We will provide a sufficient value of the parameter vector m to guarantee that the event occurs, with high probability. In words, the condition says that the value of any state s, under any policy, in the empirical known state-action MDP $\hat{M}_{K_t}$ (see Definition 5) is ϵ-close to its value in the true known state-action MDP M_{K_t}.

Event A1 *For all stationary policies π, timesteps t and states s during execution of the f-Rmax algorithm on some f-MDP M, $|V^{\pi}_{M_{K_t}}(s) - V^{\pi}_{\hat{M}_{K_t}}(s)| \leq \epsilon$.*

Lemma 27 *There exists a constant C such that if f-Rmax with parameters $m = \langle m_i \rangle$ is executed on any MDP $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ and m_i, for $i = 1, \ldots, n$, satisfies*

$$m_i \geq C \left(\frac{n^2(|\mathcal{D}(X_i)| + \ln(N/\delta))}{{\epsilon_1}^2(1-\gamma)^4} \right),$$

then event A1 will occur with probability at least $1 - \delta$.

Proof: Event A1 occurs if f-Rmax maintains a close approximation of its known state-action MDP. On any fixed timestep t, the transition distributions that f-Rmax uses are the empirical estimates as described in Section 6.1.3, using only the first m_i samples (of immediate reward and next state pairs) for each $(X_i, j) \in \cup_{(s,a)\in K}\{D_{s,a}\}$. Intuitively, as long as each m_i is large enough, the empirical estimates for these state-action pairs will be accurate, with high probability.[3] Combining Corollary 2 with

[3]There is a minor technicality here. The samples, in the form of next state factors, experienced by an online agent in an f-MDP are not necessarily independent samples. The reason for this is that the learning environment or the agent could prevent future experiences of state factors based on previously observed outcomes. Nevertheless, all the tail inequality bounds that hold for independent samples also hold for online samples in f-MDPs, a fact that is due to the Markov property. There is an extended discussion and formal proof of this in the context of general MDPs in our forthcoming paper (Strehl & Littman, 2007) that extends to the factored case.

Lemma 4 reveals that it is sufficient to obtain $C\Big(\epsilon(1-\gamma)^2/n\Big)$-accurate (in L$_1$ distance) transition components. From Theorem 13 we can guarantee the empirical transition distribution is accurate enough, with probability at least $1-\delta'$, as long as $2^{|\mathcal{D}(X_i)|}e^{-m_i\epsilon^2(1-\gamma)^4/(2n^2)} \leq \delta'$. Using this expression, we find that it is sufficient to choose m such that

$$m_i \propto \frac{n^2(|\mathcal{D}(X_i)| + \ln(1/\delta'))}{\epsilon^2(1-\gamma)^4}. \tag{6.11}$$

Thus, as long as m_i is large enough, we can guarantee that the empirical distribution for a single transition component will be sufficiently accurate, with high probability. However, to apply the simulation bounds of Lemma 4, we require accuracy for all transition components. To ensure a total failure probability of δ, we set $\delta' = \delta/N$ in the above equations and apply the union bound over all transition components. □

Proof of Main Theorem

Proof: (of Theorem 12). We apply Theorem 1. Assume that Event $A1$ occurs. Consider some fixed time t. First, we verify condition (1) of the theorem. We have that $V_t(s) = V^*_{\hat{M}_{K_t}}(s) \geq V^*_{M_{K_t}}(s) - \epsilon \geq V^*(s) - \epsilon$. The first equality follows from the fact that action-values used by f-Rmax are the result of a solution of its internal model. The first inequality follows from Event A1 and the second from the fact that M_{K_t} can be obtained from M by removing certain states and replacing them with a maximally rewarding state whose actions are self-loops, an operation that only increases the value of any state. Next, we note that condition (2) of the theorem follows from Event A1. Observe that the learning complexity, $\zeta(\epsilon, \delta)$, satisfies $\zeta(\epsilon, \delta) \leq \sum_{(X_i,j)\in\mathcal{Q}} m_i$. This is true because each time an escape occurs, some $(s,a) \notin K$ is experienced. However, once all the transition components (X_i, j) for (s,a) are experienced m_i times, respectively, (s,a) becomes part of and never leaves the set K. To guarantee that Event $A1$ occurs with probability at least $1-\delta$, we use Lemma 28 to set m. □

6.1.4 Factored IE

The *Factored IE* (or *f-IE*) algorithm is similar to f-Rmax in that it maintains empirical estimates for the transition components as described in Section 6.1.3. The main difference is that f-IE uses the empirical estimates for each transition component even if the agent has little experience (in the form of samples) with respect to that component. Like f-Rmax, f-IE has a parameter m_i for each factor X_i and it uses only the first m_i samples for each transition component $(X_i, j) \in \mathcal{Q}$ to compute its empirical estimate (all additional observed samples are discarded).[4] However, when m_i samples are yet to be obtained, f-IE still computes an empirical estimate $\hat{Pr}(\cdot|j)$ using all the observed samples. This is in contrast to the f-Rmax algorithm, which ignores such estimates. Thus, f-IE makes better use of the agent's limited experience.

For a specified state-action pair (s, a), let c_i denote the count $n(D(s, a, X_i))$ that is maintained by both the f-Rmax and f-IE algorithms. Recall that f-Rmax solves the following set of equations to compute the policy it follows:

$$
\begin{aligned}
Q(s,a) &= 1/(1-\gamma), \text{if } \exists X_i,\ c_i < m_i \\
Q(s,a) &= R(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} Q(s',a'),
\end{aligned}
$$

otherwise.

The algorithm f-IE solves a similar set of equations:

$$
\begin{aligned}
Q(s,a) &= 1/(1-\gamma), \text{if } \exists X_i,\ c_i = 0 \\
Q(s,a) &= R(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} Q(s',a') \\
&\quad + eb(c_1, c_2, \ldots, c_n) \qquad\qquad \text{otherwise.}
\end{aligned}
$$

[4]This condition was needed for our analysis to go through. Experimentally, we have found that the algorithm has reduced sample complexity but increased computational complexity without this restriction.

where $eb : \mathbb{Z}^n \to \mathbb{R}$ is a function of the form

$$eb(c_1, c_2, \ldots, c_n) := \max_{(X_i,j)\in D(s,a)} \frac{\beta_i}{\sqrt{c_i}}, \tag{6.12}$$

for some constants β_i, $i = 1, \ldots, n$. We think of this function as an *exploration bonus* that provides incentive for obtaining samples from transition components that are poorly modeled and therefore have a low count, c_i.

6.1.5 Analysis of Factored IE

The main result of this section is the following theorem.

Theorem 14 *Suppose that* $0 \leq \epsilon < \frac{1}{1-\gamma}$ *and* $0 \leq \delta < 1$ *are two real numbers and* $M = \langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$ *is any factored-state MDP with dependency function* D *and dependency identifiers* $\mathcal{I}$. *Let* n *be the number of state factors and* $\mathcal{Q}$ *be the set of transition components with* $N = |\mathcal{Q}|$. *There exists inputs* $m = (m_1, \ldots, m_n)$ *and* $\beta = (\beta_1, \ldots, \beta_n)$, *satisfying* $m_i = m_i(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{n^2(|\mathcal{D}(X_i)|+\ln(Nn/(\epsilon(1-\gamma)\delta)))}{\epsilon^2(1-\gamma)^4}\right)$ *and* $\beta_i = \frac{n}{1-\gamma}\sqrt{2\ln(Nm_i/\delta) + 2\ln(2)|\mathcal{D}(X_i)|}$, *such that if f-IE is executed on* M *with inputs* m *and* β, *then the following holds. Let* $\mathcal{A}_t$ *denote f-IE's policy at time* t *and* s_t *denote the state at time* t. *With probability at least* $1-\delta$, $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ *is true for all but*

$$O\left(\frac{n^2(\Psi + N\ln(\frac{Nn}{\epsilon(1-\gamma)\delta}))}{\epsilon^3(1-\gamma)^6} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)}\right), \tag{6.13}$$

timesteps t, *where* $\Psi = \sum_{(X_i,j)\in\mathcal{Q}} |\mathcal{D}(X_i)|$.

Analysis Details

Recall that for a fixed transition component (X_i, j), the f-IE algorithm maintains a count $n(X_i, j)$ that is equal to the number of samples obtained by the agent for estimation of the corresponding distribution. Since the algorithm will only use the first m_i samples, $n(X_i, j) \leq m_i$.

Event A2 *For all transition components $(X_i, j) \in \mathcal{Q}$, the following holds during execution of the f-IE algorithm on MDP M,*

$$||\hat{P}(\cdot|j) - P(\cdot|j)||_1 \leq \frac{\sqrt{2\ln(Nm_i/\delta) + 2\ln(2)|\mathcal{D}(X_i)|}}{\sqrt{n(X_i, j)}}, \tag{6.14}$$

Lemma 28 *The event A2 will occur with probability at least $1 - \delta$.*

Proof: Fix a transition component $(X_i, j) \in \mathcal{Q}$. Fix a moment during execution of f-IE in an f-MDP M. By Theorem 13, we have that $P(||P(\cdot|j) - \hat{P}(\cdot|j)||_1 \geq \alpha) \leq 2^{|\mathcal{D}(X_i)|}e^{-n(X_i,j)\alpha^2/2}$. Setting the right-hand side to be at most $\delta/(Nm_i)$ and solving for α proves that with probability at least $1 - \delta/(Nm_i)$, we will have that

$$||\hat{P}(\cdot|j) - P(\cdot|j)||_1 \leq \frac{\sqrt{2\ln(Nm_i/\delta) + 2\ln(2)|\mathcal{D}(X_i)|}}{\sqrt{n(X_i, j)}}, \tag{6.15}$$

To guarantee that this holds for all transition components we proceed with two applications of the union bound: first for a fixed transition component over all possible values of $n(X_i, j)$ and then for a fixed factor over all transition components. Let $F(X_i, j, k)$ denote the probability that Equation 6.15 does not hold for some timestep such that $n(X_i, j) = k$ holds. We have that

$$\sum_{(X_i,j)\in\mathcal{Q}} \sum_{k=1}^{m_i} F(X_i, j, k) \leq \sum_{(X_i,j)\in\mathcal{Q}} \sum_{k=1}^{m_i} \delta/(Nm_i) = \delta$$

□

Lemma 29 *If Event A2 occurs, then the following always holds during execution of f-IE: $||T(s, a) - \hat{T}(s, a)||_1 \leq$*

$$\max_{(X_i,j)\in D(s,a)} \frac{n\sqrt{2\ln(Nm_i/\delta) + 2\ln(2)|\mathcal{D}(X_i)|}}{\sqrt{n(X_i, j)}}, \tag{6.16}$$

for all $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Proof: The claim follows directly from Corollary 2. □

Lemma 30 *If Event A2 occurs and*

$$\beta_i \geq \frac{n}{1-\gamma}\sqrt{2\ln\left(Nm_i/\delta\right) + 2\ln(2)|\mathcal{D}(X_i)|}, \tag{6.17}$$

then the following always holds during execution of f-IE:

$$Q(s,a) \geq Q^*(s,a) \tag{6.18}$$

Proof: Recall that f-IE computes it action-value estimates, $Q(s,a)$, by solving its internal model. We prove the claim by induction on the number of steps of value iteration. Let $Q^{(i)}(s,a)$ denote the result of running value iteration of f-IE's model for i iterations. We let $Q^{(0)} = 1/(1-\gamma)$. Assume that the claim holds for some value $t-1$. We have that

$$\begin{aligned}
&Q^*(s,a) - Q^{(t)}(s,a) \\
&\leq \; Q^*(s,a) - R(s,a) - \gamma\sum_{s'}\hat{T}(s'|s,a)V^*(s') \\
&\quad\; -eb(c_1, c_2, \ldots, c_n) \\
&\leq \; \frac{1}{1-\gamma}\sum_{s'}\left(T(s'|s,a) - \hat{T}(s'|s,a)\right) \\
&\quad\; - \max_{(X_i,j)\in D(s,a)} \frac{\beta_i}{\sqrt{n(X_i,j)}} \\
&\leq \; 0.
\end{aligned}$$

The first inequality results from the induction hypothesis and the fact that $Q^{(t)}(s,a) = R(s,a) + \gamma\sum_{s'}\hat{T}(s'|s,a)\max_{a'}Q^{(t-1)}(s',a')$. The second inequality follows from the fact that $V^*(s) \leq 1/(1-\gamma)$ holds for all states s. The final inequality used Lemma 29 and Equation 6.17. $\square$

Proof of Main Theorem

Proof: (of Theorem 14). We apply Theorem 1. Assume that Event $A2$ occurs. Define the set of "known" state-action pairs K_t, at time t, to be the same as for f-Rmax:

$$K_t := \{(s,a) \in \mathcal{S} \times \mathsf{A} | n(X_i, j) \geq m_i \text{ for all } (j, X_i) \in D_{s,a}\}. \tag{6.19}$$

Consider some fixed time t. Condition (1) of the theorem holds by Lemma 30.

Next, we sketch a proof that condition (2) of the theorem holds. f-IE computes its action-value estimates by solving the empirical model with exploration bonuses added to the reward function. We need to show that it is close to the MDP M_{K_t}, which is identical to f-IE's model except that the true transition distribution is used instead of the empirical estimate for those state-action pairs in K_t and the exploration bonuses are discarded for those state-action pairs. If each exploration bonus is less than $\epsilon(1-\gamma)/2$ then the value function of the model is $\epsilon/2$-close to the value function of the model without the exploration bonuses (because any one-step reward gets multiplied by $1/(1-\gamma)$ if accrued over an infinite horizon). Thus we need to choose m so that

$$\beta_i / m_i \leq \epsilon(1-\gamma)/2 \text{ for each } i. \tag{6.20}$$

Also, by Lemma 4, we can guarantee the the value functions for the two models are ϵ-accurate as long as the transition function is $C\epsilon(1-\gamma)^2$-accurate for some constant. From Lemma 29, it is sufficient to ensure that

$$\frac{n\sqrt{2\ln\left(N m_i/\delta\right) + 2\ln(2)|\mathcal{D}(X_i)|}}{\sqrt{m_i}} \leq C\epsilon(1-\gamma)^2 \text{ for each } i, \tag{6.21}$$

holds. Ignoring constants, the conditions specified by Equations 6.20 and 6.21 are equivalent and are satisfied by

$$m_i \propto \frac{n^2(|\mathcal{D}(X_i)| + \ln\left(Nn/(\epsilon(1-\gamma)\delta)\right))}{\epsilon^2(1-\gamma)^4}. \tag{6.22}$$

Finally, note that the learning complexity, $\zeta(\epsilon, \delta) \leq \sum_{(X_i,j) \in \mathcal{Q}} m_i$. This is true because each time an escape occurs, some $(s, a) \notin K$ is experienced. However, once all the transition components (X_i, j) for (s, a) are experienced m_i times, respectively, (s, a) becomes part of and never leaves the set K. $\square$

6.2 Infinite State Spaces

In many real-world domains for which we'd like to apply reinforcement-learning techniques, the most straight-forward way to model the environment is with infinitely many states and actions. Solving this problem is currently open, although some progress has been made (Antos et al., 2006). We don't present any results for large state space in this thesis, but we note that Theorem 1 doesn't require a flat state space and can conceivably be applied to infinite state spaces.

Conclustion

In this thesis we have examined the problem of exploration verses exploitation in finite-state Markov Decision Processes. We show that both model-based and model-free algorithms can be designed so that they satisfy a theoretical condition we've termed PAC-MDP. The important aspect of these algorithms is that they act near optimally on all but a few timesteps. The analysis of all the algorithms in this thesis made use of a common framework, which can potentially help with the analysis of future algorithms in the field. Important open problems include closing the gap between the upper and lower sample complexity bounds and extending the results to MDPs with an infinite number of states and actions.

Bibliography

Antos, A., Szepesvàri, C., & Munos, R. (2006). Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *The Nineteenth Annual Conference on Learning Theory* (pp. 574–588).

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*, 81–138.

Berry, D. A., & Fristedt, B. (1985). *Bandit problems: Sequential allocation of experiments*. London, UK: Chapman and Hall.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI 96)* (pp. 115–123). Portland, OR.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to algorithms*. Cambridge, MA: The MIT Press.

Even-Dar, E., Mannor, S., & Mansour, Y. (2003). Action elimination and stopping conditions for reinforcement learning. *The Twentieth International Conference on Machine Learning (ICML 2003)* (pp. 162–169).

Even-Dar, E., & Mansour, Y. (2003). Learning rates for Q-learning. *Journal of Machine Learning Research*, *5*, 1–25.

Fiechter, C.-N. (1997). Expected mistake bound model for on-line reinforcement learning. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 116–124).

Givan, R., Leach, S., & Dean, T. (2000). Bounded-parameter Markov decision processes. *Artificial Intelligence*, *122*, 71–109.

Guestrin, C., Patrascu, R., & Schuurmans, D. (2002). Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. *Proceedings of the International Conference on Machine Learning* (pp. 235–242).

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58*, 13–30.

Kaelbling, L. P. (1993). *Learning in embedded systems.* Cambridge, MA: The MIT Press.

Kakade, S. M. (2003). *On the sample complexity of reinforcement learning.* Doctoral dissertation, Gatsby Computational Neuroscience Unit, University College London.

Kearns, M., & Singh, S. (1999). Finite-sample convergence rates for Q-learning and indirect algorithms. *Advances in Neural Information Processing Systems 11* (pp. 996–1002). The MIT Press.

Kearns, M. J., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 740–747).

Kearns, M. J., & Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, *49*, 209–232.

Kearns, M. J., & Vazirani, U. V. (1994a). *An introduction to computational learning theory.* Cambridge, Massachusetts: The MIT Press.

Kearns, M. J., & Vazirani, U. V. (1994b). *An introduction to computational learning theory.* Cambridge, Massachusetts: The MIT Press.

Littman, M. L., Dean, T. L., & Kaelbling, L. P. (1995). On the complexity of solving Markov decision problems. *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)* (pp. 394–402). Montreal, Québec, Canada.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, *13*, 103–130.

Nilim, A., & Ghaoui, L. E. (2004). Robustness in Markov decision problems with uncertain transition matrices. *Advances in Neural Information Processing Systems 16 (NIPS-03)*.

Puterman, M. L. (1994). *Markov decision processes—discrete stochastic dynamic programming*. New York, NY: John Wiley & Sons, Inc.

Singh, S. P., & Yee, R. C. (1994). An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, *16*, 227.

Strehl, A. L., Li, L., & Littman, M. L. (2006a). Incremental model-based learners with formal learning-time guarantees. *UAI-06: Proceedings of the 22nd conference on Uncertainty in Artificial Intelligence* (pp. 485–493). Cambridge, Massachusetts.

Strehl, A. L., Li, L., & Littman, M. L. (2006b). Pac reinforcement learning bounds for rtdp and rand-rtdp. *AAAI-06 Workshop on Learning for Search*.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., & Littman, M. L. (2006c). PAC model-free reinforcement learning. *ICML-06: Proceedings of the 23rd international conference on Machine learning* (pp. 881–888).

Strehl, A. L., & Littman, M. L. (2004). An empirical evaluation of interval estimation for Markov decision processes. *The 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2004)* (pp. 128–135).

Strehl, A. L., & Littman, M. L. (2005). A theoretical analysis of model-based interval estimation. *Proceedings of the Twenty-second International Conference on Machine Learning (ICML-05)* (pp. 857–864).

Strehl, A. L., & Littman, M. L. (2006). Reinforcement learning in general MDPs: PAC analysis. Working paper.

Strehl, A. L., & Littman, M. L. (2007). An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*. in press.

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.

Weissman, T., Ordentlich, E., Seroussi, G., Verdu, S., & Weinberger, M. J. (2003). *Inequalities for the L1 deviation of the empirical distribution* (Technical Report HPL-2003-97R1). Hewlett-Packard Labs.

Wiering, M., & Schmidhuber, J. (1998). Efficient model-based exploration. *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior (SAB'98)* (pp. 223–228).

Williams, R. J., & Baird, III, L. C. (1993). *Tight performance bounds on greedy policies based on imperfect value functions* (Technical Report NU-CCS-93-14). Northeastern University, College of Computer Science, Boston, MA.

Vita

Alexander L. Strehl

1980	Born March 24 in York, Pennsylvania.
1998-2002	Attended College of Charleston, South Carolina;
	majored in Computer Information Systems and Mathematics.
2002	B.S., College of Charleston.
2002-2007	Attend Rutgers University, New Jersey; majored in Computer Science.
2002	Teaching Assistantship, Department of Computer Science.
2003-2006	Research Assistantship, Department of Computer Science.
2006-2007	Bevier fellowship, Rutgers University.
2007	PH.D. in Computer Science.

Publications

1. Alexander L. Strehl, Lihong Li, and Michael L. Littman. **Incremental Model-based Learners With Formal Learning-Time Guarantees.** *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence* (UAI 2006), 2006, 485-493.
2. Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. **PAC Model-Free Reinforcement Learning.** *Proceedings of the 23rd International Conference on Machine Learning* (ICML 2006), 2006, 881-888.
3. Alexander L. Strehl, Christopher Mesterharm, Michael L. Littman, and Haym Hirsh. (2006). **Experience-Efficient Learning in Associative Bandit Problems.** *Proceedings of the 23rd International Conference on Machine Learning* (ICML 2006), 2006, 889-896.
4. Alexander L. Strehl and Michael L. Littman. **A Theoretical Analysis of Model-Based Interval Estimation.** *Proceedings of the 22nd International Conference on Machine Learning* (ICML 2005), 2005, 857-864.
5. Alexander L. Strehl and Michael L. Littman. **An Empirical Evaluation of Interval Estimation for Markov Decision Processes**. *Proceedings of the 16th International Conference on Tools with Artificial Intelligence* (ICTAI), 2004, 128-135.