Probably Approximately Corrct (PAC) Exploration in Reinforcement Learning

Alexander L. Strehl
strehl@yahoo-inc.com

Abstract

Reinforcement Learning (RL) Markov Decision Processes is studied with an emphasis on the well-studied exploration problem. We first formulate and discuss a definition of "efficient" algorithms that is termed Probably Approximately Correct (PAC) in RL. Next we provide general sufficient conditions for such an algorithm that applies to several different modeling assumptions. The conditions can be used to demonstrate that efficient learning is possible in finite MDPs, with either a model-based or model-free approach, in factored MDPs, and in continuous MDPs with linear dynamics.

1 Introduction

In the reinforcement-learning (RL) problem (Sutton & Barto 1998), an agent acts in an unknown or incompletely known environment with the goal of maximizing an external reward signal. In the most standard mathematical formulation of the problem, the environment is modeled as a Markov Decision Process (MDP) and the goal of the agent is to obtain near-optimal discounted return. Over the years, many algorithms have been proposed for this problem, but analyses of their performances have been relatively scarce. In fact, until recently, most theoretical guarantees have been that certain algorithms will discover an optimal policy in the limit, after an infinite amount of experience. In contrast, several attempts have been made to study "Probably Approximately Correct" or PAC-MDP algorithms, which exhibit near-optimal behavior in polynomial time and experience. This paper discusses several extensions of those results.

We present a theorem that provides sufficient conditions for an algorithm to be PAC-MDP. We examine these conditions and show how they can be applied to prove that efficient learning is possible in three interesting scenerios: finite MDPs (i.e. the "Tabular case"), factored MDPs, and continuous MDPs with linear dynamics.

2 Notation

This section introduces the Markov Decision Process (MDP) notation used throughout the paper; see (Sutton & Barto 1998) for an introduction. Let $\mathcal{P}_S$ denote the set of all probability distributions over the set S. An MDP M is a five tuple $\langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a set called the state space, A is a set called the action space, $T : \mathcal{S} \times \mathsf{A} \to \mathcal{P}_{\mathcal{S}}$ is the transition distribution, $\mathcal{R} : \mathcal{S} \times \mathsf{A} \to \mathcal{P}_{\mathbb{R}}$ is the reward distribution, and $0 \leq \gamma < 1$ is a discount factor on the summed sequence of rewards. We call the elements of $\mathcal{S}$ and A states and actions, respectively, and use S and A to denote the number of states and the number of 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 $\mathcal{R}(s,a)$.

We assume 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. In Section 2.1, we provide one possible formalization of this objective within the PAC-MDP framework. 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.

A *policy* is any strategy for choosing actions. A sta-

tionary 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[1] all lie in the interval $[0, 1]$. For any policy π, let $V_M^\pi(s)$ ($Q_M^\pi(s, a)$) denote the discounted, infinite-horizon value (action-value) function for π in M (which may be omitted from the notation) from state s. If H is a positive integer, let $V_M^\pi(s, H)$ 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, $V_M^\pi(c_t) = E[\sum_{j=0}^{\infty} \gamma^j r_{t+j} | c_t]$ and $V_M^\pi(c_t, H) = E[\sum_{j=0}^{H-1} \gamma^j r_{t+j} | c_t]$. 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.

2.1 Learning Efficiently

To formalize the notion of "efficient learning", we allow the learning algorithm to receive two additional inputs, ϵ and δ, both positive real numbers. The first parameter, ϵ, controls the quality of behavior we require of the algorithm (how close to optimality do we desire) and the second parameter, δ, is a measure of confidence (how certain do we want to be of the algorithm's performance). As these parameters decrease, greater exploration and learning is necessary, as more is expected of the algorithms.

In the following definition, we view an algorithm as simply some non-stationary (in terms of the current state) policy that, on each timestep, takes as input an entire history or trajectory through the MDP (its actual history) and outputs an action (which the agent then executes). Formally, we define the policy of any algorithm $\mathcal{A}$ at a fixed instance of time t to be a function $\mathcal{A}_t : \{\mathcal{S} \times \mathsf{A} \times [0, 1]\}^* \to \mathsf{A}$, that maps future paths to future actions[2].

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 $\mathcal{A}$ 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, $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$).*

Note that the sample complexity of an algorithm is dependent on some infinite-length path through the MDP. We believe this definition captures the essence of measuring learning. It directly measures the number of times the agent acts poorly (quantified by ϵ) and we view "fast" learners as those that act poorly as few times as possible. Based on this intuition, we define what it means to be an "efficient" learning algorithm

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 $\epsilon > 0$ and $0 < \delta < 1$, 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 $1-\delta$. It is simply* **PAC-MDP** *if we relax the definition to have no computational complexity requirement.*

The terminology, PAC, in this definition is borrowed from (Valiant 1984), a classic paper dealing with supervised learning. 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 or how poorly it acts on those timesteps. This definition is in contrast to Valiant's PAC notion, which is more "off-line" in that it requires the algorithm to make all of its mistakes ahead of time before identifying a near-optimal policy.

Please see (Kakade 2003) for a full motivation of this performance measure. Efficient PAC-learnability in the sample-complexity framework from above implies PAC-learnability in a more realistic framework called *Average Loss* that measures the actual return (sum of rewards) achieved by the agent against the expected return of the optimal policy (Strehl & Littman 2005). The analysis of R-MAX by (Kakade 2003) and of MBIE by (Strehl & Littman 2005) use the same definition as above. The analysis of R-MAX by (Brafman & Tennenholtz 2002) and of E^3 by (Kearns & Singh 2002) use slightly different definitions of efficient learning[3].

2.2 General Framework

We now develop some theoretical machinery to prove PAC-MDP statements about various algorithms. Our

[1] It is straightforward to generalize to the case where the rewards are bounded above and below by known but arbitrary constants, say R_{min} and R_{max}, respectively.

[2] The action of an agent on timestep t is given by the function evaluated at the empty history, $\mathcal{A}_t(\emptyset)$.

[3] (Kearns & Singh 2002) dealt with discounted and undiscounted MDPs differently. In the discounted case the agent is required to halt after a polynomial amount of time and output a near-optimal policy from the current state, with high probability. (Kakade 2003) discusses the relationship between this notion and Definiton 1.

theory will be focused on algorithms that maintain a set of action values, $Q(s,a)$, for each state-action pair (denoted $Q_t(s,a)$ at time t)[4]. We also assume an algorithm always chooses actions greedily with respect to the action values. 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 3 *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 $a_t := \text{argmax}_{a \in \mathsf{A}} Q_t(s_t,a)$, where s_t is the tth state reached by the agent.*

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

Definition 4 *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 $R_K(z_{s,a},\cdot) = Q(s,a)(1-\gamma)$.*

The known state-action MDP is a generalization of the standard notions of a "known state MDP" of (Kearns & 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 5 *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 (t)th 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 $(s,a) \notin K_t$.*

[4]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).

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, Li, & Littman 2006) 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 $1-2\delta$.

3 Finite MDPs

3.1 A model-based algorithm

In this section, we discuss the R-MAX algorithm(Brafman & Tennenholtz 2002). R-MAX is a model-based algorithm for finite MDPs. It uses all of its past experience (in the form of tuples of current state, current action, reward, and next state) to maintain an explicit MDP estimate, $\hat{T}$ and $\hat{R}$ of its unknown environment using maximum-likelihood estimation. It also requires an 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:

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

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.

We can use Theorem 1 to prove the following:

Theorem 2 *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* $m(\frac{1}{\epsilon}, \frac{1}{\delta}) = O\left(\frac{S+\ln(SA/\delta)}{\epsilon^2(1-\gamma)^4}\right)$ *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* $1-\delta$*,* $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ *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*.*

3.2 A model-free algorithm

The *Delayed Q-learning* algorithm was introduced by (Strehl *et al.* 2006) 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 plus a small exploration bonus of ϵ_1.

Although we don't have room to go into the details of the algorithm or its analysis we mention that Theorem 1 can be used to prove the following theorem.

Theorem 3 *(Strehl* et al. *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* $m = m(\frac{1}{\epsilon}, \frac{1}{\delta})$ *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* $1-\delta$*,* $V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \epsilon$ *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*.*

4 Factored MDPs

Definition 6 *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.

We assume that the transition function factors so that

$$T(x'|x,a) = \prod_i P(x'_i|x,a). \tag{1}$$

We also assume that the component transition distributions $P(\cdot|x,a)$ can be compactly represented by a *Bayesian Dynamic Network* (DBN) (Boutilier, Dean, & Hanks 1999). Using Theorem 1, we can prove that a modified version of Factored R-MAX (Guestrin, Patrascu, & Schuurmans 2002) is PAC-MDP. The main difference between Factored R-MAX and R-MAX is that instead of learning each transition probability $(Ts'|s,a)$separately (leading to $S^2 * A$ independent variables to learn) Factored R-MAX estimates each transition component $(P(x'_i|x,a))$ from data and keeps track of which components are the same (according to the graphical structure of the DBN). For the full details please see (Strehl 2007).

5 Continuous MDPs with linear dynamics

In the previous sections we have assumed that the number of states in the underlying MDP is finite. Here we relax that assumption and consider MDPs whose states are represented as real vectors.

The model we use is slightly modified from the model described by (Abbeel & Ng 2005). Let $\mathcal{P}_S$ denote the set of all (measurable) probability distributions over the set S. The environment is described by a discounted MDP $M = \langle S, A, T, R, \gamma \rangle$, where $S = \mathbb{R}^{n_S}$ is the state space, $A = \mathbb{R}^{n_A}$ is the action space, $T : S \times A \to P_S$ is the unknown transition dynamics, $\gamma \in [0,1)$ is the discount factor, and $R : S \times A \to \mathbb{R}$ is the known reward function.[5] For each timestep t, let $x_t \in S$ denote

[5] All of our results can easily be extended to the case of an unknown reward function with a suitable linearity assumption.

the current state and $u_t \in A$ the current action. The transition dynamics T satisfy

$$x_{t+1} = M\phi(x_t, u_t) + w_t, \quad (2)$$

where $x_{t+1} \in S$, $\phi(\cdot,\cdot) : \mathbb{R}^{n_S+n_A} \to \mathbb{R}^n$ is a (basis or kernel) function satisfying $||\phi(\cdot,\cdot)|| \leq 1$, and M is an $n_S \times n$ matrix. We assume that the 2-norm of each row of M is bounded by 1.[6] Each component of the noise term $w_t \in \mathbb{R}^{n_S}$ is chosen i.i.d. from a normal distribution with mean 0 and variance σ^2 for a known constant σ. If an MDP satisfies the above conditions we say that it is *linearly parameterized*, because the next-state x_{t+1} is a linear function of the vector $\phi(x_t, u_t)$ (which describes the current state and action) plus a noise term. We assume that the learner (also called the *agent*) receives n_S, n_A, n, R, $\phi(\cdot,\cdot)$, σ, and γ as input, with T initially being unknown.

We begin by creating a supervised regression problem, $f_i(\phi(s,a))$, for each state-component i that attempts to predict the ith component of the state resulting from taking any action a from any initial state s, that is, we learn a function $f_i : \mathbb{R}^n \to \mathbb{R}$ for each $i \in \{1, \ldots, n_S\}$. The learned function approximators can be combined to form a model (in the form of a linearly parameterized MDP) of the unknown environment. We then solve this model and act according to the optimal policy of the model. Exploration is achieved by keeping confidence bounds on the predictions made by f_i and when we are not sufficiently confident about a given prediction for state s and action a, we assume (in the model) that (s,a) leads to a maximally rewarding state. Thus, the algorithm can be viewed as a direct generalization of R-Max to linearly parameterized MDPs. Using Theorem 1, we can prove that it is PAC-MDP.

6 Conclusion

We have developed a general theory that allows the solution of a variety of interesting reinforcement learning problems in a formal manner. Future work consists of applying the theory to more realistic problems and developing large-scale systems. One of the fundamental difficulties of our approach presented is to overcome the computational burden of planning in the model-based approaches.

References

Abbeel, P., and Ng, A. Y. 2005. Exploration and apprenticeship learning in reinforcement learning. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, 1–8. New York, NY, USA: ACM Press.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

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

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

Kakade, S. M. 2003. *On the Sample Complexity of Reinforcement Learning*. Ph.D. Dissertation, Gatsby Computational Neuroscience Unit, University College London.

Kearns, M. J., and Singh, S. P. 2002. Near-optimal reinforcement learning in polynomial time. *Machine Learning* 49(2–3):209–232.

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

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

Strehl, A. L.; Li, L.; and Littman, M. L. 2006. Incremental model-based learners with formal learning-time guarantees. In *UAI-06: Proceedings of the 22nd conference on Uncertainty in Artificial Intelligence*, 485–493.

Strehl, A. L. 2007. Model-based reinforcement learning in factored-state MDPs. In *ADPRL 2007: Proceedings of the 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. In press.

Sutton, R. S., and 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(11):1134–1142.

[6] The algorithm can be modified to deal with bounds (on the norms of the rows of M) that are larger than one.