Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Efficient PAC-Optimal Exploration in Concurrent, Continuous State MDPs with Delayed Updates

Jason Pazis
Laboratory for Information and Decision Systems*
Massachusetts Institute of Technology
Cambridge, MA 02139
jpazis@mit.edu

Ronald Parr
Department of Computer Science
Duke University
Durham, NC 27708
parr@cs.duke.edu

Abstract

We present a new, efficient PAC optimal exploration algorithm that is able to explore in multiple, continuous or discrete state MDPs simultaneously. Our algorithm does not assume that value function updates can be completed instantaneously, and maintains PAC guarantees in realtime environments. Not only do we extend the applicability of PAC optimal exploration algorithms to new, realistic settings, but even when instant value function updates are possible, our bounds present a significant improvement over previous single MDP exploration bounds, and a drastic improvement over previous concurrent PAC bounds. We also present TCE, a new, fine grained metric for the cost of exploration.

1 Introduction and Motivation

PAC-optimal exploration algorithms have received significant attention from the Reinforcement learning (RL) community over the last few years. Considering how important efficient exploration is for RL, this attention is well deserved. Unlike non-sequential machine learning, in RL it is often the case that samples have to be obtained by sequential interaction with a Markov decision process (MDP). As a result, sampling certain regions that the optimal policy visits often can be quite improbable unless we are actively trying to reach those regions. Furthermore, even in cases where we have a generative model of the environment, it may be the case that only a tiny percentage of the state space is reachable from the starting state; in that situation, learning by collecting samples from the entire space would be inefficient.

The first major roadblock preventing most PAC-optimal exploration methods from being applicable in practice is that they scale poorly to domains of realistic size. Not only do they assume that the state-action space is discrete and finite, which renders them inapplicable to continuous space domains, but their sample complexity becomes prohibitive in all but the smallest problems. Both of these drawbacks are a result of their inability to generalize, a weakness that our algorithm does not share.

In many real world problems value function updates are not instantaneous. Consider the following two examples: 1) A fast-paced realtime system, whose time-step is too short to allow updating the value function between every step. 2) A mobile system where instead of computing the value function locally we send new samples to a remote platform which then returns the updated value function with some delay. Delayed value function updates are the norm rather than the exception in real life, yet they are not adequately addressed by current PAC-optimal exploration methods. As we will show in our analysis, the worst case sample complexity of our algorithm degrades gracefully with increasing value function update delays.

Another situation that appears often in real life is when we want to explore in multiple MDPs at the same time. This scenario encompasses exploring in multiple "copies" of the same MDP simultaneously, or exploring in multiple MDPs with varying degrees of similarity. While both of these problems are interesting, the second one presents additional difficulties. It might be tempting to try to cluster MDPs into discrete clusters, and then treat each cluster as if all MDPs in the cluster are identical. Unfortunately this approach runs into the same scalability issues as discrete state-action exploration algorithms that lack the ability to generalize. There are many cases where we are dealing with MDPs that are different enough that we don't want to treat them as identical, but we would still like to be able to take advantage of any similarities present. Consider for example the case where we have multiple MDPs that are similar in most of the state-action space, but differ drastically in a small but important section. One of the ways this situation can occur is when we have multiple MDPs with the same dynamics but different goals. As we will see, generalizing over MDPs is no different than generalizing over state-actions. Our approach to generalization will allow us to handle multiple MDPs seamlessly, taking advantage of any degree of similarity present.

Our focus in this paper is on the *online, discounted exploration* setting. The most commonly used metric in this setting is the number of time-steps t such that $V^{\pi}(s_t) < V^{*}(s_t) - \epsilon$, commonly referred to as "sample complexity" (Kakade 2003). One of the main drawbacks of sample complexity is that it is very coarse: It only counts the number of significantly suboptimal steps, and does not discriminate between the magnitude of different errors. In addition

*This research was performed while the first author was a graduate student with the computer science department at Duke University.

to our algorithmic contribution, we present TCE, a new, fine grained metric for the cost of exploration. As we will show in section 3, TCE bounds imply sample complexity bounds.

Regret is a metric that shares the fine-grained nature of TCE. While it has gained significant traction in other exploration settings, it is unsuitable for our setting. Regret for discounted MDPs is defined as the difference between the expected accumulated discounted reward of an optimal policy and the discounted accumulated reward achieved by the algorithm in question. For MDPs with non-negative rewards, regret for the discounted setting is upper bounded by $\frac{R_{\max}}{1-\gamma}$. There are at least two ways in which regret in discounted MDPs can be arbitrarily close to this upper bound: 1) A common and arguably necessary (Jaksch, Ortner, and Auer 2010) assumption in regret analysis is a finite mixing time/diameter. By assuming a finite diameter, we assume that our algorithm cannot get trapped in a "bad" set of states. Unfortunately this is a very strong assumption that is not true for many real world domains (such as for any robotic system that can get damaged or trapped, any living organism that can get injured or simply bored and stop using our system, and any financial entity that can go broke) As a simple example of why the lack of finite diameter can yield regret arbitrarily close to $\frac{R_{\max}}{1-\gamma}$, consider an MDP with two states: The starting state has two actions, the first of which is self-transitioning with a reward of $R_{\max}$, while the second has zero reward and transitions to an absorbing state with zero reward. If the algorithm is unlucky and chooses the second action first, its regret will be $\frac{R_{\max}}{1-\gamma}$. 2) Even if we assume a finite diameter, it is easy to design discounted MDPs for which regret is arbitrarily close to $\frac{R_{\max}}{1-\gamma}$. Consider an MDP where the starting state has two actions: The first is self-transitioning with reward $R_{\max}$, while the second one has zero reward and transitions to an n long chain of states each of which has zero reward and the last of which transitions back to the starting state. If the algorithm is unlucky and chooses the second action first, its regret will be at least $(1-\gamma^n)\frac{R_{\max}}{1-\gamma}$, where n can be as large as $|S|$. The situation can become even worse if we add stochasticity.

2 Background

A *Markov Decision Process* (MDP) is a 5-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\boldsymbol{\mathcal{S}}$ is the state space of the process, $\boldsymbol{\mathcal{A}}$ is the action space, $\boldsymbol{P}$ is a Markovian transition model $\big(p(s'|s,a)$ denotes the probability density of a transition to state s' when taking action a in state $s\big)$, $\boldsymbol{R}$ is a reward function $\big(R(s,a,s')$ is the reward for taking action a in state s and transitioning to state $s'\big)$, and $\boldsymbol{\gamma} \in [0,1)$ is a discount factor for future rewards. We will use $\boldsymbol{\mathcal{A}_s}$ to denote the set of actions available at state s. A *deterministic policy* $\boldsymbol{\pi}$ for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action choice in state s. The value $\boldsymbol{V^{\pi}(s)}$ of a state s under a policy π is defined as the expected, accumulated, discounted reward when the process begins in state s and all decisions are made according to policy π. There exists an optimal policy $\boldsymbol{\pi^*}$ for choosing actions which yields the optimal value function $V^*(s)$, defined recursively via the Bellman optimality equation:

$$\boldsymbol{V^*(s)} = \sup_a \{ \int_{s'} p(s'|s,a) \left(R(s,a,s') + \gamma V^*(s')\right)\}.$$

Similarly, the value $\boldsymbol{Q^{\pi}(s,a)}$ of a state-action pair (s,a) under a policy π is defined as the expected, accumulated, discounted reward when the process begins in state s by taking action a and all decisions thereafter are made according to policy π. The Bellman optimality equation for Q becomes:

$$\boldsymbol{Q^*(s,a)} = \int_{s'} p(s'|s,a) \left(R(s,a,s') + \gamma \sup_{a'} \{Q^*(s',a')\} \right).$$

For a fixed policy π the Bellman operator for Q is defined as:

$$\boldsymbol{B^{\pi}Q(s,a)} = \int_{s'} p(s'|s,a) \Big(R(s,a,s') + \gamma Q(s', \pi(s')) \Big).$$

In reinforcement learning (Sutton and Barto 1998), a learner interacts with a stochastic process modeled as an MDP and typically observes the state and immediate reward at every step; however, the transition model P and reward function R are not known. The goal is to learn a near optimal policy using experience collected through interaction with the process. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the reward received r, the resulting next state s', and $\mathcal{A}_{s'}$, the set of available actions in s', essentially sampling the transition model and reward function of the process. Thus experience comes in the form of $(s, a, r, s', \mathcal{A}_{s'})$ samples.

There have been many definitions of sample complexity in RL. For PAC-optimal exploration algorithms in discounted MDPs, the most commonly used one is the following (Kakade 2003):

Definition 2.1. *Let π be an arbitrarily complex, possibly non-stationary, possibly history dependent policy (such as the policy followed by an exploration algorithm), and let $(s_1, s_2, s_3, \dots)$ be a random path generated by π. The* **sample complexity** *of exploration is the number of time-steps t such that $V^{\pi}(s_t) < V^*(s_t) - \epsilon$.*

We will be defining a more fine-grained sample complexity metric in section 3.

The notion of efficient PAC optimal exploration has also evolved over the years. We will be using the following definition (Strehl, Li, and Littman 2009):

Definition 2.2. *An algorithm is said to be* **efficient PAC-MDP** *(Probably Approximately Correct in Markov Decision Processes) if, for any $\epsilon > 0$ and $0 < \delta < 1$, its sample complexity, its per-timestep computational complexity, and its space complexity, are less than some polynomial in the relevant quantities $(S, A, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma})$, with probability at least $1-\delta$.*

Over the years, some researchers have dropped the polynomial per-timestep computational and space complexity requirements. While it is theoretically interesting to see what we can achieve when computational and space complexity requirements are lifted, algorithms whose computational and/or space complexity is super-polynomial in the relevant quantities are of limited use in practical applications.

3 The Total Cost of Exploration (TCE)

When interacting with the real world we typically cannot assume a finite diameter, yet we may care about more than just the number of highly suboptimal steps an exploration algorithm might incur. For example, if we are using our algorithm to explore using a robot or some other physical system, it would be more useful to have an upper bound on the amount of wear and tear of the robot, rather than a bound on the number of times the robot will be completely destroyed.

We now define a metric that does not require unrealistic assumptions, is suitable for measuring exploration performance in discounted MDPs, and provides fine-grained information:

Definition 3.1. *Let π be an arbitrarily complex, possibly non-stationary, possibly history dependent policy (such as the policy followed by an exploration algorithm), $(s_1, s_2, s_3, \dots)$ be a random path generated by π, ϵ a positive constant, $\mathcal{T}$ the (possibly infinite) set of time steps for which $V^\pi(s_t) < V^*(s_t) - \epsilon$, and define*[1]*:*

$$\epsilon_e(t) = V^*(s_t) - V^\pi(s_t) - \epsilon, \ \forall\, t \in \mathcal{T}.$$
$$\epsilon_e(t) = 0, \ \forall\, t \notin \mathcal{T}.$$

The Total Cost of Exploration (**TCE**) *is defined as the undiscounted infinite sum:*

$$\sum_{t=0}^{\infty} \epsilon_e(t).$$

It is now easy to see that for any policy for which the TCE is bounded, the number of at least ϵ-suboptimal steps is also bounded:

Lemma 3.2. *For any policy π for which the TCE is bounded, there will be at most:*

$$\frac{\sum_{t=0}^{\infty} \epsilon_e(t)}{\epsilon_{threshold}}$$

steps such that:

$$V^\pi(s_t) < V^*(s_t) - \epsilon - \epsilon_{threshold},$$

where $\epsilon_{threshold}$ is any positive constant.

Unfortunately upper "number of suboptimal steps" sample complexity bounds do not translate to tight TCE bounds. Given a sample complexity bound for an exploration algorithm for which $V^\pi(s_t) < V^*(s_t) - \epsilon$ for at most n time steps, the best we can say is that $\sum_{t=0}^{\infty} \epsilon_e(t) \leq n(V_{\max} - \epsilon)$.

Note that just like sample complexity, TCE does not make any assumptions on the algorithm or the MDP to which it is applied. For example, no assumptions are made as to whether the algorithm distinguishes between "known" and "unknown" states, or whether the MDP has a finite diameter. While beyond the scope of this work, many existing algorithms could be analyzed in terms of TCE.

[1]Note that $V^\pi(s_t)$ denotes the expected, discounted, accumulated reward of the arbitrarily complex policy π from state s at time t, rather than the expectation of some stationary snapshot of π.

Lower bound

Sample complexity is known to be log-linear in the size of the state-action space $|SA|$ (Strehl, Li, and Littman 2009), quadratic in $\frac{1}{\epsilon}$, and cubic in $\frac{1}{1-\gamma}$ (Lattimore and Hutter 2012), for an overall lower worst case bound of:

$$\Omega\left(\frac{|SA|\log|SA|}{\epsilon^2(1-\gamma)^3}\right).$$

This implies a lower worst case bound of:

$$\Omega\left(\frac{|SA|\log|SA|}{\epsilon(1-\gamma)^3}\right)$$

for TCE (if a better bound could be established, lemma 3.2 would imply a bound better than $\Omega\left(\frac{|SA|\log|SA|}{\epsilon^2(1-\gamma)^3}\right)$ for sample complexity which is a contradiction). In theorem 8.5 we come within $\frac{1}{1-\gamma}$ of this lower bound (ignoring logarithmic factors). This implies that we also come within $\frac{1}{1-\gamma}$ of the lower bound for "number of suboptimal steps" sample complexity.

4 Overview

One of the ways that our approach differs from previous work on PAC-optimal exploration is that instead of assuming a monolithic learner and policy execution algorithm, we allow learning and policy execution to occur on separate, parallel processes. Each concurrent MDP is assigned a policy execution process, while a single learner processes samples generated from all policy execution processes and computes new policies.

A major challenge in the design of an efficient exploration algorithm with low TCE is that of approximation. Ideally we would like to have an infinite number of samples for every state-action-MDP triple. Unfortunately, maintaining computational and memory efficiency means that we have to limit the number of samples that we keep in memory and process at every step. In order to deal with the fact that the state-action-MDP space may have infinite cardinality, we make a smoothness assumption (which is significantly less restrictive than Lipschitz assumptions in previous work). Our smoothness assumption allows us to use samples from nearby state-action-MDP triples to infer bounds on the values of state-action-MDP triples for which we have insufficient (or no) samples. For computational and memory efficiency purposes, when a sufficient number of samples has been gathered in the vicinity of a particular state-action-MDP triple, future samples are discarded.

Finally, we need to be able to guarantee that the cost accumulated during exploration is low. To that end we use the principle of optimism in the face of uncertainty, and prove that our optimistic approximation scheme guarantees low TCE with high probability.

5 Definitions and Assumptions

Let $\mathcal{X}$ be the domain of x. Throughout this paper, $\forall x$ will serve as a shorthand for $\forall x \in \mathcal{X}$. For example we will use $\forall(s,a)$ as a shorthand for $\forall(s,a) \in (\mathcal{S}, \mathcal{A})$.

In the following $\boldsymbol{s, \bar{s}, \tilde{s}, s'}$ are used to denote various states, $\boldsymbol{a, \bar{a}, \tilde{a}, a'}$ are used to denote actions, and $\boldsymbol{M, \bar{M}, \tilde{M}, M'}$ are used to denote MDPs.

We will use $\boldsymbol{(s, a, M)}$ to denote state-action (s, a) in MDP M, $\boldsymbol{Q(s, a, M)}$ to denote the Q value function of (s, a) in MDP M, $\boldsymbol{BQ(s, a, M)}$ to denote the application of the Bellman operator for MDP M to $Q(s, a, M)$, and $\boldsymbol{(s, a, r, s', \mathcal{A}_{s'}, M)}$ to denote a $(s, a, r, s', \mathcal{A}_{s'})$ sample collected from MDP M.

We assume that all value functions Q live in a complete metric space. We also assume that all rewards are bounded, and without loss of generality that they lie in $\boldsymbol{[0, R_{\max}]}$.[2]

Definition 5.1. $\boldsymbol{Q_{\max}}$ *denotes an upper bound on the maximum expected, discounted reward from any state-action in the family of MDPs we are considering:*

$$0 \leq Q^{\pi}(s, a, M) \leq \boldsymbol{Q_{\max}} \leq \frac{R_{\max}}{1 - \gamma} \; \forall (s, a, M, \pi).$$

We also define $\boldsymbol{\hat{Q}_{\max}} = R_{\max} + \gamma Q_{\max}$.

Definition 5.2. $\boldsymbol{d(s, a, M, \bar{s}, \bar{a}, \bar{M})}$ *is defined to be the distance of state-action (s, a) in MDP M, to state-action $(\bar{s}, \bar{a})$ in MDP $\bar{M}$ in some well-defined distance metric. We also define the shorthand:*

$$\boldsymbol{d(s, a, M, \bar{s}, \bar{a}, \bar{M}, d_{known})} = \max\{0, d(s,a, M, \bar{s}, \bar{a}, \bar{M}) - d_{known}\},$$

where d_{known} is a user defined constant.

Examples of distance metrics one could use include weighted norms on the state-action-MDP space, norms on linear and non-linear transformations of the state-action-MDP space, as well as norms on features of the state-action-MDP space.

Definition 5.3. *The* **covering number** $\boldsymbol{\mathcal{N}_{\mathcal{SAM}}(d_{known})}$ *of a state-action-MDP space is the size of the largest minimal set*[3] *C of state-action pairs, such that for any (s, a) reachable from the starting state(s) of any MDP M in the set of MDPs we are exploring in, there exists $(\bar{s}, \bar{a}, \bar{M}) \in C$ such that:*

$$d(s, a, M, \bar{s}, \bar{a}, \bar{M}) \leq \boldsymbol{d_{known}}.$$

The covering number serves a dual purpose: It generalizes the concept of cardinality of a state-action-MDP space to the continuous setting, and it allows the user to trade off sample, space, and computational complexity with approximation precision in both continuous and discrete spaces. In the discrete setting $\mathcal{N}_{\mathcal{SAM}}(d_{known})$ is bounded above by the cardinality of the state-action-MDP space.

[2]It is easy to satisfy this assumption in all MDPs with bounded rewards by simply shifting the reward space.

[3]A minimal set having property X is a set such that if any one of its elements is removed it will cease to have property X. There can be multiple such sets, of different cardinalities. Among the minimal sets having property X, the largest minimal set has the greatest cardinality.

Definition 5.4. $\boldsymbol{|\mathcal{A}|}$ *is defined as the upper bound on the number of discrete actions the agent can take in any state. For simplicity of exposition we assume that $|\mathcal{A}|$ is finite. In section 9 we will show how any MDP with a continuous action space can be treated as if it had a discrete action space.*

Definition 5.5. *The* **sample set** *is defined as a mutable set of up to $k\mathcal{N}_{\mathcal{SAM}}(d_{known})$ $(s_i, a_i, r_i, s'_i, \mathcal{A}_{s'_i}, M_i)$ samples.*

The reason that the sample set is constrained to at most $k\mathcal{N}_{\mathcal{SAM}}(d_{known})$ samples is for space and computational efficiency.

Definition 5.6. *An* **approximation unit** *u is defined to be a 2-tuple comprised of the following:*

1. *A real value.*
2. *A mutable set of up to $\boldsymbol{k}$ pointers to samples in the sample set.*

An approximation unit is identified by a state-action-MDP triple as $\boldsymbol{u(s, a, M)}$.

As the name signifies, an approximation unit is the basic unit of approximation used by our algorithm. For simplicity, we will say "the samples in approximation unit $u(s, a, M)$" to refer to the samples pointed to by approximation unit $u(s, a, M)$.

Definition 5.7. *An* **approximation set** $\boldsymbol{U}$ *is defined as a mutable set of 0 to $\mathcal{N}_{\mathcal{SAM}}(d_{known})$ approximation units. No two approximation units in an approximation set can have the same identifying state-action-MDP triple.*

Definition 5.8. $\boldsymbol{K_a}$ *is defined to be the set $K_a = \{2^0, 2^1, 2^2, \ldots, 2^i\} \cup \{k\}$, where i is the largest positive integer such that $2^i < k$. Let $u(s, a, M)$ be an approximation unit pointing to k_0 samples $(s_i, a_i, r_i, s'_i, \mathcal{A}_{s'_i}, M_i)$ for $i \in [1, k_0]$. The function* $\boldsymbol{k_a(u(s, a, M))}$ *returns the largest value k_a in $\boldsymbol{K_a}$ such that $k_a \leq k_0$. If such a value does not exist,* $\boldsymbol{k_a(u(s, a, M))}$ *returns 0. We will call the first $k_a(u(s, a, M))$ samples of $u(s, a, M)$ the* **active samples** *of $u(s, a, M)$. Since no two approximation units in an approximation set can have the same identifying state-action-MDP triple, we will use* $\boldsymbol{k_a(s, a, M)}$ *to denote $k_a(u(s, a, M))$.*

In other words the active samples in $u(s, a, M)$ are the samples in $u(s, a, M)$ rounded down to the nearest power of 2, except when $u(s, a, M)$ contains k samples in which case all k samples are active. Some readers may wonder why K_a only contains k and powers of 2, rather than all integers in $[1, k]$. When setting K_a we have to balance two competing goals: 1) Ensuring that the Bellman error drops quickly as more samples are gathered, and 2) limiting the number of policy changes. While setting K_a to all integers in $[1, k]$ accomplishes the first goal, it does poorly on the second.

Definition 5.9. *Let* $\boldsymbol{\epsilon_b}$ *be a real value which we will call the* **exploration bonus**. *Let $u(s, a, M)$ be an approximation unit for which $k_a(s, a, M) > 0$. The function*

$\boldsymbol{F^{\pi}(Q, u(s,a,M))}$ *is defined as:*

$$\boldsymbol{F^{\pi}(Q, u(s,a,M))} = \frac{\epsilon_b}{\sqrt{k_a(s,a,M)}} + \frac{\sum_{i=1}^{k_a(s,a,M)} \left(r_i + \gamma Q(s'_i, \pi(s'_i), M_i) \right)}{k_a(s,a,M)},$$

where $(s_i, a_i, r_i, s'_i, \mathcal{A}_{s'_i}, M_i)$ *is the* i*-th sample pointed to by* $u(s,a,M)$*. We will use* $\boldsymbol{F(Q, u(s,a,M))}$ *to denote* $F^{\pi^Q}(Q, u(s,a,M))$.

F^{π} computes the average over the active samples of an approximation unit, and returns the average plus a bonus that depends on the number of samples in the approximation unit.

Definition 5.10. *If* U *contains at least one approximation unit* $u(s,a,M)$ *for which* $k_a(s,a,M) > 0$*, the function* $\boldsymbol{N(U, s, a, M)}$ *returns the identifying state-action-MDP triple of the approximation unit* $u(\bar{s}, \bar{a}, \bar{M}) \in U$ *with* $k_a(\bar{s}, \bar{a}, \bar{M}) > 0$ *such that*

$$d(s,a,M,\bar{s},\bar{a},\bar{M}, d_{known}) + \frac{\epsilon_b}{\sqrt{k_a(\bar{s},\bar{a},\bar{M})}}$$

is minimized. Ties are broken by an arbitrary deterministic function.

Instead of returning the identifying state-action-MDP triple of the nearest approximation unit with respect to $d(s,a,M,\bar{s},\bar{a},\bar{M},d_{known})$, $N(U,s,a,M)$ takes into account the number of active samples of every approximation unit.

Definition 5.11. *For state-action* (s,a) *in MDP* M*, the approximate optimistic Bellman operator* $\boldsymbol{\tilde{B}^{\pi}}$ *for policy* π *is defined as:*

$$\boldsymbol{\tilde{B}^{\pi}Q(s,a,M)} = \min\{Q_{\max}, F^{\pi}(Q, u(N(U,s,a,M))) + d(s,a,M,N(U,s,a,M),d_{known})\}.$$

We will use $\boldsymbol{\tilde{B}Q(s,a,M)}$ *to denote* $\tilde{B}^{\pi^Q}Q(s,a,M)$*. When* U *is the empty set,* $\tilde{B}^{\pi}Q(s,a,M) = Q_{\max}$.

The approximate optimistic Bellman operator is used in the inner loop of our exploration algorithm.

Definition 5.12. *The value function* $\boldsymbol{Q_U}$ *of an approximation set* U *is defined as:*

$$\boldsymbol{Q_U(s,a,M)} = \min\{Q_{\max}, u_v(N(U,s,a,M)) + d(s,a,M,N(U,s,a,M),d_{known})\},$$

where $u_v(N(U,s,a,M))$ *is the value stored in approximation unit* $u(N(U,s,a,M))$*. If* U *is the empty set* $Q_U(s,a,M) = Q_{\max}\ \forall(s,a,M)$.

Definition 5.13. $\boldsymbol{\epsilon_c} \geq 0$ *is the minimal non-negative constant satisfying:*

$$\forall(s,a,M,\bar{s},\bar{a},\bar{M},\pi,Q_{\tilde{U}}),$$
$$|B^{\pi}Q_{\tilde{U}}(s,a,M) - B^{\pi}Q_{\tilde{U}}(\bar{s},\bar{a},\bar{M})| \leq \boldsymbol{\epsilon_c} + d(s,a,M,\bar{s},\bar{a},\bar{M},d_{known}).$$

Note that while d_{known} is a user-defined parameter, algorithm execution does not require ϵ_c to be known.

Definition 5.13 describes one of the most important concepts this work is based on. It will allow us to show that as long as $B^{\pi}Q_{\tilde{U}}(s,a,M)$ is similar for nearby state-action-MDP triples, our algorithm will perform well. Contrary to previous work (Pazis and Parr 2013), we can guarantee this even when $B^{\pi}Q_{\tilde{U}}(s,a,M)$ is not Lipschitz continuous, or when our algorithm is learning from multiple MDPs rather than a single MDP.

6 Concurrent Exploration

When exploring in $\boldsymbol{k_p}$ concurrent MDPs, our algorithm consists of k_p (one per MDP) instantiations of the policy execution process (algorithm 1), and a single learner[4] (algorithm 2). Policy execution processes communicate with the learner by adding samples to a queue for inclusion in the sample set, and the learner communicates with the policy execution processes by publishing $\boldsymbol{\tilde{U}}$, a set of degenerate[5] approximation units that define the approximate value function $\boldsymbol{Q_{\tilde{U}}}$.

Algorithm 1 Policy execution (k_p instantiations, one for each concurrent MDP):

1: Initialize $\tilde{U}$ to the empty set.
2: **loop**
3: From state s in MDP M, using the latest available published $\tilde{U}$:
4: Perform action $a = \arg\sup_{\tilde{a}} Q_{\tilde{U}}(s,\tilde{a},M)$
5: Receive reward r
6: Transition to state s', and observe $\mathcal{A}_{s'}$.
7: Submit $(s,a,r,s',\mathcal{A}_{s'},M)$ to the inclusion candidate queue.
8: **end loop**

Algorithm 1 is a greedy policy execution algorithm. At every step, it uses the latest available published $\tilde{U}$, selects the greedy action, collects a sample, and submits it to the learner for processing.

Algorithm 2 is based on value iteration. Samples submitted by the policy execution processes are processed sequentially in lines 4 through 12. If a sample is more than d_{known} away from the nearest approximation unit in the approximation set, a new approximation unit is added. If the sample is within d_{known} of an approximation unit that has fewer than k samples, the sample is added to the sample set, and a pointer to the sample is added to all approximation units within d_{known} distance that have fewer than k samples. Lines 13 through 19 perform a single step of value iteration and publish the updated value function. They are repeated until the one step Bellman error of Q_U drops below ϵ_a.

[4] Our results readily generalize to the case where we have multiple learners, even in the case where the order in which samples are processed is different for each learner. For simplicity of exposition we will assume that only a single learner exists.

[5] Since only values of published approximation units are ever accessed, approximation units in $\tilde{U}$ do not contain sample pointers.

Algorithm 2 Learner

1: Initialize U to the empty set, publish a snapshot of U as $\tilde{U}$, and set update to false.
2: **loop**
3: Set U_{old} to U.
4: **while** the inclusion candidate queue is not empty **do**
5: Pop $(s, a, r, s', \mathcal{A}_{s'}, M)$ from the queue.
6: **if** $d(s, a, M, \bar{s}, \bar{a}, \bar{M}) > d_{known}\ \forall\ u(\bar{s}, \bar{a}, \bar{M}) \in U$ **then**
7: Add a new approximation unit $u(s, a, M)$ to U, initialize it with up to k pointers to any $(s_i, a_i, r_i, s'_i, \mathcal{A}_{s'_i}, M_i)$ samples in the sample set for which $d(s, a, M, s_i, a_i, M_i) \leq d_{known}$, and set update to true.
8: **end if**
9: **if** $d(s, a, M, \bar{s}, \bar{a}, \bar{M}) \leq d_{known}$ for some $u(\bar{s}, \bar{a}, \bar{M}) \in U$ containing fewer than k samples **then**
10: Add $(s, a, r, s', \mathcal{A}_{s'}, M)$ to the sample set, and a pointer to each $u(\bar{s}, \bar{a}, \bar{M}) \in U$ containing fewer than k samples for which $d(s, a, M, \bar{s}, \bar{a}, \bar{M}) \leq d_{known}$. If this increases the number of active samples for at least one approximation unit, set update to true.
11: **end if**
12: **end while**
13: **if** update is true **then**
14: Set Q_U to $\tilde{B}Q_{U_{old}}$.
15: Publish a snapshot of U as $\tilde{U}$.
16: **if** $-\boldsymbol{\epsilon_a} \leq Q_{U_{old}}(s, a, M) - Q_U(s, a, M) \leq \boldsymbol{\epsilon_a}\ \forall\ u(s, a, M) \in U$ **then**
17: Set update to false.
18: **end if**
19: **end if**
20: **end loop**

There are couple of implementation issues we should elaborate on:

- Setting Q_U in line 14 of the learner is performed by setting the values of each $u(s, a, M) \in U$ for which $k_a(s, a, M) > 0$ to $F(Q_{U_{old}}, u(s, a, M))$.
- Computing $F(Q_{U_{old}}, u(s, a, M))$ does not require recomputing $N(U, s', a', M)$. By maintaining $|\mathcal{A}|$ pointers to approximation units per sample (one for each $a' \in A_{s'}$), $F(Q_{U_{old}}, u(s, a, M))$ can be computed efficiently. To initialize the pointers for a particular sample, $N(U, s', a', M)$ needs to be called up to $|\mathcal{A}|$ times (once for each $a' \in A_{s'}$) when the sample is added to the sample set, and the pointers need to be updated every time the number of active samples for an approximation unit changes (at a much lower cost than recomputing $N(U, s', a', M)$).

7 Concurrency, Time, and Delay Models

We will keep our definitions of concurrency, timestep, and value function update delays general, so as to cover as many real-world situations as possible.

Definition 7.1. $\boldsymbol{k_p} \geq 1$ *denotes the maximum number of MDPs we are exploring in parallel.*

Definition 7.2. Timestep $\boldsymbol{t_i}$ *of MDP* M_i *is defined as the* t_i*-th action our policy takes in MDP* M_i*.*

Definition 7.3. Global time $\boldsymbol{t_g}$ *is a measure of time that is common between all MDPs.*

We do not assume that actions in different MDPs are synchronized, that they happen at a constant rate with respect to global time, or that all MDPs have the same start time. For example, it could be the case that in global time $[t_g(s), t_g(f)]$ the policy at MDP M_1 takes actions $2, 3$, the policy at MDP M_2 takes actions $7, 8, 9, 10$, while no actions have been taken yet in MDPs M_3 and M_4.

Assumption 7.4. *For ease of exposition we assume that samples are processed by the learner in the same order* (*with respect to global time*) *as the actions that generate them. This assumption can be easily removed, but it makes exposition significantly simpler.*

Definition 7.5. *Let action* (s, a) *be taken at MDP* M_i *at global time* $t_g(s)$ *generating sample* $(s, a, r, s', \mathcal{A}_{s'}, M_i)$*.* $\boldsymbol{D_{(s,a,r,s',\mathcal{A}_{s'}M_i),M_j}}$ *is the delay for sample* $(s, a, r, s', \mathcal{A}_{s'}, M_i)$ *with respect to MDP* M_j*. If processing* $(s, a, r, s', \mathcal{A}_{s'}, M_i)$ *does not cause the update flag of the algorithm to be set,* $D_{(s,a,r,s',\mathcal{A}_{s'},M_i),M_j} = 0$ *for all* M_j*. Otherwise* $D_{(s,a,r,s',\mathcal{A}_{s'},M_i),M_j}$ *is the number of actions taken in MDP* M_j *in* $[t_g(s), t_g(u_j))$*, where* $t_g(u_j)$ *is the time* M_j *receives an updated value function* Q_U *for which* $-\boldsymbol{\epsilon_a} \leq Q_{U_{old}}(s, a, M) - Q_U(s, a, M) \leq \boldsymbol{\epsilon_a}\ \forall\ u(s, a, M) \in U$ *on line 16 of algorithm 2 after* $(s, a, r, s', \mathcal{A}_{s'}, M_i)$ *has been processed. Every action taken in MDP* M_j *during* $[t_g(s), t_g(u_j))$ *is called a* **delay step** *for MDP* M_j*.*

8 Analysis

From Lemmas 8.1, 8.2, 8.3, and 8.4, and theorem 8.5 below, we have that the combination of Algorithms 1 and 2 is efficient PAC-MDP. Due to space constrains proofs are deferred to the appendix, available at the authors' websites.

Lemma 8.1. *The space complexity of algorithm 1 is:*

$$O\left(\mathcal{N}_{\mathcal{SAM}}(d_{known})\right)$$

per concurrent MDP.

Lemma 8.2. *The space complexity of algorithm 2 is:*

$$O\left(k|\mathcal{A}|\mathcal{N}_{\mathcal{SAM}}(d_{known})\right).$$

Lemma 8.3. *The per step computational complexity of algorithm 1 is bounded above by:*

$$O\left(|\mathcal{A}|\mathcal{N}_{\mathcal{SAM}}(d_{known})\right).$$

Since every step of algorithm 1 results in a sample being processed by algorithm 2, we will be bounding the per sample computational complexity of algorithm 2:

Lemma 8.4. *Let c be the maximum number of approximation units to which a single sample can be added[6]. The per sample computational complexity of algorithm 2 is bounded above by:*

$$O\left(ck|\mathcal{A}|\mathcal{N}_{\mathcal{SAM}}(d_{known}) + \frac{k|\mathcal{A}|\mathcal{N}_{\mathcal{SAM}}(d_{known})}{\ln\gamma}\ln\frac{\epsilon}{Q_{\max}}\right).$$

Theorem 8.5 below is the main theorem of this paper. It decomposes errors into the following four sources:

1. ϵ_s is the error caused by the fact that we are using only a finite set of samples (at most k) to estimate the mean, thus we only have an estimate.
2. ϵ_c, a function of d_{known}, is the error caused by the fact that since the space is continuous we cannot have samples for every state-action.
3. ϵ_a is the error caused by the fact that we are only finding an ϵ_a-approximation, rather than the true fixed point of $\tilde{B}$.
4. Finally, $\epsilon_e(t,j)$ is the error caused by the fact that at time t there may exist state-actions in MDP M_j that do not have an approximation unit with k samples within d_{known} distance, and value function updates may not be instantaneous.

Theorem 8.5. *Let $(s_{1,j}, s_{2,j}, s_{3,j}, \dots)$ for $j \in \{1, \dots, k_p\}$ be the random paths generated in MDPs $M_1, \dots, M_{k_p}$ respectively on some execution of algorithm 1, and $\tilde{\boldsymbol{\pi}}$ be the (non-stationary) policy followed by algorithm 1. Let $\boldsymbol{\epsilon_b} = \hat{Q}_{\max}\sqrt{\frac{\ln\frac{4\lceil 1+\log_2 k\rceil\mathcal{N}_{\mathcal{SAM}}(d_{known})^2}{\delta}}{2}}$, $\boldsymbol{k} \geq \frac{\hat{Q}_{\max}^2}{2\epsilon_s^2(1-\gamma)^2}\ln\left(\frac{4\lceil 1+\log_2 k\rceil\mathcal{N}_{\mathcal{SAM}}(d_{known})^2}{\delta}\right)$, ϵ_c be defined as in definition 5.13, ϵ_a be defined as in algorithm 2, k_p be defined as in definition 7.1, and $\boldsymbol{T_j}$ be the set of non-delay steps (see definition 7.5) for MDP M_j. If $\frac{2\left\lceil\frac{1}{1-\gamma}\ln\frac{Q_{\max}}{\epsilon_s}\right\rceil\ln\frac{2\lceil 1+\log_2 k\rceil}{\delta}}{(1+k_p)\mathcal{N}_{\mathcal{SAM}}(d_{known})} < 1$ and $\mathcal{N}_{\mathcal{SAM}}(d_{known}) \geq 2$, with probability at least $1-\delta$, for all t, j:*

$$V^{\tilde{\pi}}(s_{t,j}, M_j) \geq V^*(s_{t,j}, M_j) - \frac{4\epsilon_c + 2\epsilon_a}{1-\gamma} - 3\epsilon_s - \epsilon_e(t,j),$$

where[7]:

$$\sum_{j=1}^{k_p}\sum_{t=0}^{\infty}\epsilon_e(t,j) = \sum_{j=1}^{k_p}\sum_{t\in T_j}\epsilon_e(t,j) + \sum_{j=1}^{k_p}\sum_{t\notin T_j}\epsilon_e(t,j),$$

with:

$$\sum_{j=1}^{k_p}\sum_{t\in T_j}\epsilon_e(t,j) \approx \tilde{O}\left(\frac{\left(\frac{\hat{Q}_{\max}}{\epsilon_s(1-\gamma)} + k_p\right)\mathcal{N}_{\mathcal{SAM}}(d_{known})\hat{Q}_{\max}}{(1-\gamma)}\right)$$

and:

$$\sum_{t\notin T_j}\epsilon_e(t,j) \leq \left(Q_{\max}\sum_{k_a\in K_a}\sum_{i=1}^{\mathcal{N}_{\mathcal{SAM}}(d_{known})} D_{i,j,k_a}\right),$$

where D_{i,j,k_a} is the delay of the k_a-th sample for $k_a \in K_a$ in the i-th approximation unit, with respect to MDP M_j. Note that the probability of success $1-\delta$ holds for all timesteps in all MDPs simultaneously, and $\sum_{j=1}^{k_p}\sum_{t=0}^{\infty}\epsilon_e(t,j)$ is an undiscounted infinite sum. Unlike $\sum_{j=1}^{k_p}\sum_{t\in T_j}\epsilon_e(t,j)$ which is a sum over all MDPs, $\sum_{t\notin T_j}\epsilon_e(t,j)$ gives a bound on the total cost due to value function update delays for each MDP separately.

As we can see, increasing k_p by one, increases the TCE over all MDPs by only $\tilde{O}\left(\frac{\hat{Q}_{\max}\mathcal{N}_{\mathcal{SAM}}(d_{known})}{(1-\gamma)}\right)$, which is significantly less with respect to ϵ_s, $Q_{\max}$ and γ than the cost of exploring in an additional MDP separately (completely independent of $\frac{1}{\epsilon_s}$ and cheaper by a factor of $\frac{Q_{\max}}{1-\gamma}$).

"Number of suboptimal steps" sample complexity in the discrete, single MDP, instantaneous update case is not the focus of this work, yet using lemma 3.2 to convert theorem 8.5 to a "number of suboptimal steps" PAC bound yields an improvement over the best available bounds for discrete MDPs (Szita and Szepesvári 2010), by a factor of $\frac{Q_{\max}}{(1-\gamma)}$.

9 Discussion

Theorem 8.5 has a linear dependence on $\frac{1}{\epsilon_s}$ for the TCE metric, leading to bounds that are quadratic in $\frac{1}{\epsilon_s}$ with respect to the "number of suboptimal steps" metric. While both bounds are optimal with respect to what these metrics are measuring (see section 3), the fact that we are able to achieve linear dependence on $\frac{1}{\epsilon_s}$ for TCE is very informative. Since TCE is arguably much better aligned with how practical applications evaluate cost, it tells us that achieving low ϵ_s is much easier than the "number of suboptimal steps" metric would suggest.

Generalization

Definition 5.13 allows us to generalize across states, actions, and MDPs. One thing to notice is that there is no fundamental difference between generalizing across state-actions of the same MDP and state-actions of different MDPs. Generalizing across state-action-MDP triples is a natural extension of the idea of generalizing across state-actions rather than just across states.

When exploring in two identical MDPs, the distance between the same state-action in the two MDPs will be zero, while the covering number of the state-action-MDP space will be identical to the covering number of each MDP. Conversely, when two MDPs have nothing in common, the distance between state-actions across the two MDPs will be greater than or equal to $Q_{\max}$, and the covering number of the combined state-action-MDP space will be the sum of the covering numbers of the individual MDPs. The power of our generalization scheme is that in addition to these two extreme cases, it can handle everything in between. MDPs in a

[6] c will depend on the dimensionality of the state-action-MDP space. For domains that are big enough to be interesting it will be significantly smaller than other quantities of interest.

[7] $f(n) = \tilde{O}(g(n))$ is a shorthand for $f(n) = O(g(n)\log^c g(n))$ for some c.

state-action-MDP space can have sections in which they are identical, completely dissimilar, or exhibit varying degrees of similarity. Definition 5.13 allows us to take advantage of any degree of similarity to decrease the cost of exploration.

Contrary to previous work (Pazis and Parr 2013), definition 5.13 does not assume that $B^\pi Q_{\tilde{U}}(s, a, M)$ is Lipschitz continuous. The Lipschitz assumption is problematic for two reasons: Not only are Lipschitz continuity based algorithms required to know the Lipschitz constant in advance (which will typically not be the case in an exploration scenario), but even small discontinuities break their guarantees. By contrast, the effect that discontinuities and underestimations of the distance function have on definition 5.13 is an increase of ϵ_c, which causes our bounds to degrade gracefully.

Algorithms 1 and 2 are directly applicable to discrete MDPs. A distance function can be defined over a discrete space just as easily as it can be defined over a continuous one. Even if no distance information exists, by picking the distance function which returns $d(s, a, M, \bar{s}, \bar{a}, \bar{M}) = 0$ when $(s, a, M) = (\bar{s}, \bar{a}, \bar{M})$ and $d(s, a, M, \bar{s}, \bar{a}, \bar{M}) = \infty$ otherwise, the covering number becomes the cardinality of the state-action-MDP space $|S||A|k_p$ and our bounds still hold for discrete MDPs.

Continuous action spaces

In definition 5.4 we assumed that there exists a finite upper bound $|\mathcal{A}|$ on the number of actions the agent can take in any state. To some extent this is trivially true as long as we are dealing with electronics. Any digital to analogue converter will have finite precision and accept a discrete set of values as input. While this is sufficient to satisfy efficient PAC-MDP requirements, enumerating the available range is impractical for many practical applications.

Dealing with action selection in continuous and/or multi-dimensional action spaces is an open research problem that we cannot hope to fully cover in this section. Instead we will present a simple method to limit $|\mathcal{A}|$. Similarly to how we defined a covering number over the state-action-MDP space, we can define a covering set $\mathcal{N}_a(d_{known})$ for a particular action a. We will define $|\mathcal{A}|$ such that the cardinality of the covering set for any action in the state-action-MDP space will be less than or equal to $|\mathcal{A}|$. From definition 5.13, an approximate Bellman operator $\tilde{B}$ that evaluates only the actions in the covering set for each next state will only differ by at most ϵ_c from the approximate Bellman operator $\tilde{B}$ that evaluates all actions (even in the cases where $\tilde{B}$ has an infinite number of available actions). This means that even if we were to discretize our action space to $|\mathcal{A}|$ intelligently selected actions, we would only have to pay a penalty of at most $\frac{\epsilon_c}{1-\gamma}$ in our bounds.

Incorporating prior knowledge

Apart from a distance function (which is not even required to be informative for discrete state-action-MDP spaces), our analysis assumed that we are starting from a blank slate without any knowledge of the MDPs we are exploring in. It is very easy to incorporate prior knowledge from multiple sources, without adversely affecting our bounds.

Existing sample sets generated from the same state-action-MDP space (but not necessarily from the same MDPs we are going to explore) can be simply added to the inclusion candidate queue before (or even during) policy execution. Every sample added to the queue before policy execution begins, that was generated from the same MDPs we are going to be exploring in, has the potential to reduce exploration's sample requirements by one.

Rather than a fixed bound of $Q_{\max}$ for every state-action-MDP triple, prior knowledge of a better upper bound for some state-action-MDP triples can be incorporated directly into the Bellman operator. $((1-\gamma)\epsilon_s + \epsilon_a)$-accurate prior knowledge of the value of the identifying state-action-MDP triple of an approximation unit effectively reduces the covering number by one.

One of the ways in which we can derive a good distance function is by partial knowledge of the reward and transition models. Upper bounds on the reward received from various state-action-MDP triples and knowledge about their next-state distributions can be used to derive very accurate distance functions (or refine a coarser distance function). A distance function that accurately reflects the behavior of the Bellman operator can help keep the covering number small.

While beyond the scope of this work, there are many other approaches that can be combined with our algorithms to incorporate prior knowledge without adversely affecting our bounds (Mann 2012).

10 Related Work

The only other work we are aware of where learning and policy execution happen in parallel processes is the work of Hester et al. (Hester and Stone 2013; 2012; Hester, Quinlan, and Stone 2012). The authors correctly argue that while there exist algorithms that learn with few samples, and there exist algorithms that are sufficiently computationally efficient so as to be able to take actions in realtime, no previous work addresses both problems simultaneously. They then proceed to demonstrate experimentally that their algorithm is in fact able to learn very quickly while acting in realtime. While the experimental results of Hester et al. are very encouraging, their algorithm does not come with any performance guarantees, nor does it handle concurrent exploration in multiple MDPs (though extending it to the multiple MDP case appears straightforward).

A recent paper has demonstrated that it is possible to explore in multiple MDPs simultaneously (Guo and Brunskill 2015). Our work significantly improves on that result, by: 1) Extending concurrent exploration to continuous state-action spaces, 2) allowing for non-instantaneous value function updates, 3) reducing the sample complexity of exploration on all key quantities, and 4) allowing smooth generalization between concurrent MDPs, rather than clustering MDPs and treating each cluster as identical.

Lattimore and Hutter (2012) present an algorithm that, similarly to our own, integrates new samples every time the available number of samples for a particular state-action doubles. As a result, their algorithm achieves optimal dependence on the discount factor. Unfortunately their bounds

only hold for the restricted case of discrete MDPs where every state-action can transition to at most two next states.

Exploration in Metric State Spaces (Kakade, Kearns, and Langford 2003) is the first example in the PAC-MDP literature which tries to address exploration in continuous state spaces. While definitely a step in the right direction, the paper did not offer a concrete algorithm.

C-PACE (Pazis and Parr 2013), a fixed point based algorithm, was the first concrete PAC-optimal exploration algorithm for online, discounted, continuous state-action MDPs.

DGPQ (Grande, Walsh, and How 2014), is the first Q-learning inspired algorithm for PAC-optimal exploration in continuous spaces. Its development was motivated by the fact that many PAC-optimal algorithms (including C-PACE), perform a fixed point computation after every step, which makes them unsuitable for realtime applications. Experimental results with DGPQ are encouraging, as the authors demonstrate that DGPQ requires significantly less computation per step than C-PACE. While a step in the right direction, DGPQ does have some drawbacks of its own: 1) DGPQ assumes that it will always be able to perform an update between steps, which is not true for domains with delays. 2) Like all Q-learning inspired algorithms, DGPQ requires significantly more samples than fixed-point based methods. As this paper has demonstrated we can have the best of both worlds: The sample efficiency of fixed-point based methods can be combined with realtime execution if we allow for policy execution and learning to run on parallel processes.

11 Future Work

We designed and analyzed our algorithm in a setting where we only care about the total cost of exploration, not when this cost is incurred. While this setting is popular with the theoretical community, in practical applications *when* costs are incurred is far from irrelevant. Consider for example the case where PAC-optimal exploration is employed by a company designing a product. Exploration costs incurred during development are far less important than exploration costs incurred when the product has reached the end user. Designing and analyzing algorithms that have explicit exploration and exploitation phases based on the theory developed in this work is a promising next step.

One aspect of the real world which our exploration algorithm did not take into account is that in many real life systems catastrophic failures are unacceptable. Consequently methods for safe exploration are of great real world interest (Moldovan and Abbeel 2012).

A central assumption made by our algorithm in the context of continuous state-action spaces and multiple concurrent MDPs, is that there exists some distance function in which the effects of applying the Bellman operator are approximately smooth[8]. While it is reasonable to expect that such a distance function exists, many obvious distance functions that a user might try may not satisfy this requirement. Automatic discovery of suitable distance functions is an important next step.

[8]Our bounds allow for discontinuous functions, and small, local discontinuities will not significantly affect performance. Unfortunately, large discontinuities on a global scale can make ϵ_c unacceptably large.

Acknowledgments

We would like to thank Vincent Conitzer, George Konidaris, Mauro Maggioni, Peter Stone, and the anonymous reviewers for helpful comments and suggestions. This material is based upon work supported by the National Science Foundation under Grant No. IIS-1218931. Opinions, findings, conclusions or recommendations herein are those of the authors and not necessarily those of NSF.

References

Grande, R. C.; Walsh, T. J.; and How, J. P. 2014. Sample efficient reinforcement learning with Gaussian processes. In *International Conference on Machine Learning*, 1332–1340.

Guo, Z., and Brunskill, E. 2015. Concurrent PAC RL. In *AAAI Conference on Artificial Intelligence*, 2624–2630.

Hester, T., and Stone, P. 2012. Intrinsically motivated model learning for a developing curious agent. In *The Eleventh International Conference on Development and Learning*.

Hester, T., and Stone, P. 2013. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning* 90(3):385–429.

Hester, T.; Quinlan, M.; and Stone, P. 2012. RTMBA: A real-time model-based reinforcement learning architecture for robot control. In *IEEE International Conference on Robotics and Automation*, 85–90.

Jaksch, T.; Ortner, R.; and Auer, P. 2010. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research* 11:1563–1600.

Kakade, S.; Kearns, M. J.; and Langford, J. 2003. Exploration in metric state spaces. In *International Conference on Machine Learning*, 306–312.

Kakade, S. M. 2003. *On the sample complexity of reinforcement learning*. Ph.D. Dissertation, Gatsby Computational Neuroscience Unit, University College London.

Lattimore, T., and Hutter, M. 2012. PAC bounds for discounted MDPs. In *Proceedings of the 23th International Conference on Algorithmic Learning Theory*, volume 7568 of *Lecture Notes in Computer Science*, 320–334. Springer Berlin / Heidelberg.

Mann, T. A. 2012. *Scaling Up Reinforcement Learning Without Sacrificing Optimality by Constraining Exploration*. Ph.D. Dissertation, Texas A & M University.

Moldovan, T. M., and Abbeel, P. 2012. Safe exploration in Markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning*, 1711–1718.

Pazis, J., and Parr, R. 2013. PAC optimal exploration in continuous space Markov decision processes. In *AAAI Conference on Artificial Intelligence*, 774–781.

Strehl, A. L.; Li, L.; and Littman, M. L. 2009. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research* 10:2413–2444.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts.

Szita, I., and Szepesvári, C. 2010. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *International Conference on Machine Learning*, 1031–1038.