Online Exploration in Least-Squares Policy Iteration

Lihong Li
RL^3, Computer Science
Rutgers University
Piscataway, NJ 08854
lihong@cs.rutgers.edu

Michael L. Littman
RL^3, Computer Science
Rutgers University
Piscataway, NJ 08854
mlittman@cs.rutgers.edu

Christopher R. Mansley
RL^3, Computer Science
Rutgers University
Piscataway, NJ 08854
cmansley@cs.rutgers.edu

ABSTRACT

One of the key problems in reinforcement learning is balancing exploration and exploitation. Another is learning and acting in large or even continuous Markov decision processes (MDPs), where compact function approximation has to be used. In this paper, we provide a practical solution to exploring large MDPs by integrating a powerful exploration technique, Rmax, into a state-of-the-art learning algorithm, least-squares policy iteration (LSPI). This approach combines the strengths of both methods, and has shown its effectiveness and superiority over LSPI with two other popular exploration rules in several benchmark problems.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence-Learning; I.2.8 [**Computing Methodologies**]: Artificial IntelligenceProblem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

Least-Squares Policy Iteration (LSPI), Exploration, PAC-MDP, Markov Decision Processes, Reinforcement Learning

1. INTRODUCTION

One of the key problems in reinforcement learning [24] is the *exploration-exploitation tradeoff*, which strives to balance two competing types of behavior of an autonomous agent in an unknown environment: the agent can either make use of its current knowledge about the environment to maximize its cumulative reward (*i.e.*, to exploit), or sacrifice short-term rewards to gather information about the environment (*i.e.*, to explore) in the hope of increasing future long-term return.

Exploration can be framed as a *dual control* problem, and (in principle) can be solved *optimally* in a Bayesian manner. However, this approach is in general computationally intractable (*e.g.*, [10, 19]). Despite the recent advances in finding approximate solutions with less computation time, these techniques have not been shown to scale well to larger problems. Therefore, this paper only considers non-Bayesian approaches. Thrun [25] surveyed a number of popular exploration rules, but little can be said about their performance guarantees. In fact, some of them have provably poor performance in certain situations. Recently, there has been a growing interest in formally analyzing the *sample complexity of exploration* [13] in finite-state Markovian environments [8, 15]. This line of work has significantly advanced understanding of the exploration-exploitation dilemma, but has not been merged with approaches for function approximation needed for scaling up except in limited cases [22].

Cite as: Online Exploration in Least-Squares Policy Iteration, Lihong Li, Michael L. Littman and Christopher R. Mansley, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

In contrast, this paper is concerned with intelligent exploration in large or even continuous environments where compact function approximation has to be used. In particular, we propose a practical solution by integrating a powerful exploration technique, Rmax [8], into a state-of-the-art learning algorithm, least-squares policy iteration (LSPI) [17]. Our approach enjoys the strengths of both methods: on the one hand, it borrows ideas from Rmax to actively explore the state space; on the other hand, LSPI acts as an efficient learner and planner. Based on finite training samples, these two elements together produce a policy that either reduces uncertainty about the environment, or exploits the current knowledge to maximize utility of the agent.

The paper is organized as follows. Section 2 introduces notation for Markov decision processes and reinforcement learning. Section 3 reviews LSPI and a few representative exploration strategies in the literature. Section 4 studies exploration in large state spaces, and proposes the LSPI-Rmax algorithm. We briefly discuss a few implementation issues and also relate this algorithm to several existing works. The effectiveness of our approach is then demonstrated in a series of experiments on four benchmark RL problems in Section 5. Finally, we conclude the paper in Section 6.

2. PRELIMINARIES

We consider environments modeled as Markov decision processes [20], or MDPs for short. An MDP M can be described as a five-tuple $\langle S, A, T, R, \gamma \rangle$, where S is a set of states, A is a finite set of actions, T is the transition function with $T(s, a, s')$ denoting the probability of reaching s' from s by taking action a, R is a bounded reward function with $R(s, a) \in [R_{\min}, R_{\max}]$ denoting the expected immediate reward gained by taking action a in state s for some constants $R_{\min}$ and $R_{\max}$, and $\gamma \in [0, 1]$ is a discount factor.

A deterministic policy π maps states to actions; that is, $\pi : S \rightarrow A$. Given a policy π, we define the state-value function, $V^{\pi}(s)$, as the expected cumulative reward received by executing π starting from state s. Similarly, the state–action value function, $Q^{\pi}(s,a)$, is the expected cumulative reward received by taking action a in state s and following π thereafter. It is known that [20] the optimal value functions exist: $V^* = \max_\pi V^\pi$ and $Q^* = \max_\pi Q^\pi$; furthermore, the greedy policy with respect to Q^* is optimal, where a greedy policy π_Q with respect to a value function Q is one that selects actions with maximum Q-values; namely, $\pi_Q(s) = \operatorname{argmax}_a Q(s,a)$. Because of the bounded reward function and the discount factor, the value functions must be bounded as well: $R_{\min}/(1-\gamma) \leq Q^\pi(s,a) \leq R_{\max}/(1-\gamma)$.

Given the complete model of a finite MDP (*i.e.*, the five tuple), standard algorithms exist for finding the optimal value function and the optimal policy, including linear programming, value iteration, and policy iteration [20]. However, if the transition and/or reward functions are unknown, the agent has to learn the optimal value function or policy by interacting with the environment. This problem is also called reinforcement learning [24].

3. PREVIOUS WORK

In this section, we discuss relevant literature, especially the building blocks of our work, including LSPI and a number of representative exploration strategies.

3.1 LSPI

LSPI is well-known for its excellent performance in data efficiency and has succeeded in a number of challenging control problems such as riding a simulated bicycle [17]. It adopts the approximate policy-iteration framework [4] and uses a model-free version of least-squares temporal difference learning (LSTD) [5, 7] as a subroutine for policy evaluation. Like LSTD, LSPI is highly efficient in utilizing training samples and operates directly in a linear function space defined by a given set of features ϕ: $Q_\theta(s,a) = \theta^{\mathrm{T}} \cdot \phi(s,a)$, where the feature function ϕ maps state–actions to a feature vector of k components, and θ is the coefficient vector.

LSPI is arguably the most competitive reinforcement-learning algorithm available in large environments. Being an approximate policy-iteration algorithm, LSPI is theoretically sound [4]. It completely avoids learning rates and does not suffer from the problem of divergence in value functions, which is a risk many algorithms with function approximation have to face [2, 6].

In its original form, LSPI is an *off-line* algorithm, in the sense that it requires a fixed set of training samples as input and returns a policy as output [17]. Lagoudakis and Parr suggested using truncated random walks from random start states to collect training samples and to feed them to LSPI, leaving the online exploration problem open [17]. Their sample-collection approach, however, is not always applicable in practice, especially when the agent cannot be *reset.* Meanwhile, an online agent has to decide wisely how to collect samples autonomously, while random walks could be inefficient or even disastrous.

In the next subsection, we introduced a few simple-to-implement heuristics for online exploration, which can be combined with LSPI in a straightforward way. In Section 4.1, we show how to augment the original LSPI with more efficient exploration strategy that is motivated by recent advances in sample-efficient reinforcement learning.

3.2 Exploration in MDPs

The simplest and most popular exploration rule is probably ϵ-*greedy*: the agent chooses the greedy action with respect to its value function with probability $1-\epsilon$, and a random action with probability ϵ. An alternative approach, called *counter-based* exploration, has shown empirical advantages over ϵ-greedy in some problems [25]. It requires a threshold m, and a counter c is maintained for each (s,a) pair that remembers how many times action a has been taken in state s. When the agent is in state s, it randomly picks action a such that $c(s,a) < m$; if no such action exists, a greedy action is chosen. Both ϵ-greedy and counter-based methods can be shown to be inefficient in some problems.

The E^3 family of exploration algorithms explicitly alternates exploration and exploitation phases [14, 15], in which metric E^3 is probably the most relevant to the this paper within this family. It makes two assumptions: the *local modeling assumption* requires that an accurate local model be available (with high probability) given enough samples in that local region; the approximate planning assumption requires that an approximate planner be able to return a near-optimal policy for a state, given access to a generative model. It is proved that, with high probability, after a polynomial number of steps, these E^3 algorithms will terminate with a near-optimal policy for the currently occupied state.

Another approach with similar formal guarantees is Rmax [8], which implements the *optimism-in-the-face-of-uncertainty* principle. Like the counter-based method, Rmax has a threshold m and maintains a counter for each (s,a) pair for the same purpose. Unlike the counter-based method, Rmax constructs an internal model (known as the *empirical known-state MDP*) $\hat{M}_K = \langle S, A, \hat{T}, \hat{R}, \gamma \rangle$ as follows. If $c(s,a) \geq m$, then (s,a) is called *known*, and Rmax uses relevant samples to build maximum-likelihood estimates $\hat{R}_K(s,a)$ and $\hat{T}_K(s,a,\cdot)$. Otherwise, (s,a) is *unknown*, and Rmax considers it maximally rewarding forever to take a in s; precisely, it sets $\hat{R}(s,a) = R_{\max}$ and $\hat{T}(s,a,s) = 1$. A *known-state MDP* M_K is similar to $\hat{M}_K$ except that its reward and transition functions for known state–actions are identical to those of M. With this trick, Rmax provides a simple yet powerful solution to balance exploration and exploitation.

4. EXPLORATION IN LARGE MDPS

We now study the problem of exploration in MDPs with large state spaces. By the technique of discretization, a large state space can be partitioned into a much smaller set of aggregated states. Under certain assumptions, solving the smaller MDP gives a near-optimal solution to the original, large MDP [9]. Thus, we may first discretize the state space and then apply techniques for finite MDPs to obtain a near-optimal policy. Unfortunately, discretization suffers from the *curse of dimensionality.* Instead, we will use LSPI with linear function approximation, which is feasible in large problems because of generation.

Exploration rules for finite MDPs can be extended to LSPI when it is applied online. For example, LSPI with ϵ-greedy exploration picks a greedy action with probability $1-\epsilon$ and a random action with probability ϵ; counter-based exploration can be generalized in a pretty straightforward way: instead of counting the number of times action a has been tried in the current state s, we may count the number of times a has been tried in states close to s, where "closeness" is measured by a pre-defined distance metric and is problem-specific. In the following section, we describe a more complicated, online variant of LSPI that is motivated by Rmax. The basic idea is to generalize the optimism-in-the-face-of-uncertainty principle of Rmax to LSPI in continuous state spaces.

4.1 LSPI-Rmax

As a first step toward combining LSPI with Rmax-style exploration, we modified the concept of known state–action pairs to continuous state spaces. Our approach requires a distance function that computes the dissimilarity (or distance) between two state–action pairs: $d : (S \times A) \times (S \times A) \to \mathbb{R}_+$. Let $\epsilon_d \geq 0$ and $m > 0$ be two predefined constants. Given a set of sample transitions $D = \{\langle s_i, a_i, r_i, s'_i \rangle\}$, a pair (s, a) is called (ϵ_d, m)-known if $|\{(s_i, a_i) \mid d(s, a, s_i, a_i) \leq \epsilon_d\}| \geq m$. In words, the algorithm has seen at least m samples in D that are ϵ_d-*close* to (s, a). Furthermore, a state s is called (ϵ_d, m)-known if (s, a) is (ϵ_d, m)-known for every action a.

With this notion, we are now ready to present LSTD-Rmax (Algorithm 1), a variant of LSTD that assigns maximum value to unknown states and unknown state–action pairs. If both (s, a) and s' are (ϵ_d, m)-known, LSTD-Rmax operates exactly as LSTD (steps 7—9). If (s, a) is (ϵ_d, m)-known but s' is not, then we pretend s' is a goal state whose value is the largest possible, that is, $R_{\max}/(1-\gamma)$. Therefore, the return of (s, a) is the sum of the immediate reward, r, and the pretended *discounted* value of s', $\gamma R_{\max}/(1-\gamma)$; this corresponds to steps 10—12 that encourage the agent to further explore the neighborhood of s'. Similarly, when (s, a) is not (ϵ_d, m)-known, the return of this state–action pair is treated as $R_{\max}/(1-\gamma)$, which results in steps 15—16. Finally, since solutions found by LSTD-Rmax are biased by the sample distribution in the input sample set D, steps 18 to 21 add artificial samples to (s, a') pairs so that untried actions a' in s are preferred when a nearby state is visited in the future. These state–action pairs also have the largest possible value, $R_{\max}/(1-\gamma)$.

With LSTD-Rmax as a building block, we can complete the full online learning algorithm, LSPI-Rmax, whose generic form is given in Algorithm 2. In this algorithm, the agent always chooses greedy actions with respect to its current value-function estimate. When new training samples are added to the sample set D, it runs LSPI to update its value function, in which it uses LSTD-Rmax instead of LSTD to evaluate value functions.

To summarize: LSTD-Rmax is just like LSTD, except that it assigns maximum value, $R_{\max}/(1-\gamma)$, to states and state–action pairs that are not (ϵ_d, m)-known. LSPI-Rmax is just an online version of LSPI that uses LSTD-Rmax instead of LSTD to evaluate policies.

4.2 Implementation Issues

While we have given the generic form of LSPI-Rmax, a number of implementation issues remain open. We describe our solutions to these problems, but note that other solutions do exist and might work better in certain situations.

The first natural question is how to decide whether a state–action pair is (ϵ_d, m)-known or not. A straightforward algorithm would be to search over all samples in D and count the number of nearest neighbors within ϵ_d distance. However, when D is large, even this $O(|D|)$-time algorithm is far

Algorithm 1 LSTD-Rmax

```
 1: Input:
      • $D$: set of transitions
      • $\pi$: policy to be evaluated
      • $\phi$: feature function ($k$ is the number of features)
      • $\gamma$: discount factor
      • $\epsilon_d, m$: thresholds for $(\epsilon_d, m)$-known state–actions
 2: Output: weight vector $\theta$ for predicting $Q^\pi$
 3: $A \leftarrow \mathbf{0}_{k\times k}$ (the $k \times k$ zero matrix)
 4: $b \leftarrow \mathbf{0}_k$ (the $k \times 1$ zero column-vector)
 5: for all $\langle s, a, r, s' \rangle \in D$ do
 6:   if $(s, a)$ is $(\epsilon_d, m)$-known then
 7:     if $s'$ is $(\epsilon_d, m)$-known then
 8:       $A \leftarrow A + \phi(s, a) \cdot \big(\phi(s, a) - \gamma\phi(s', \pi(s'))\big)^{\mathrm{T}}$
 9:       $b \leftarrow b + \phi(s, a) \cdot r$
10:     else
11:       $A \leftarrow A + \phi(s, a) \cdot \phi(s, a)^{\mathrm{T}}$
12:       $b \leftarrow b + \phi(s, a) \cdot \Big(r + \frac{\gamma R_{\max}}{1-\gamma}\Big)$
13:     end if
14:   else
15:     $A \leftarrow A + \phi(s, a) \cdot \phi(s, a)^{\mathrm{T}}$
16:     $b \leftarrow b + \phi(s, a) \cdot \frac{R_{\max}}{1-\gamma}$
17:   end if
18:   for all $a' \in A \setminus \{a\}$ where $(s, a')$ is not $(\epsilon_d, m)$-known
      do
19:     $A \leftarrow A + \phi(s, a') \cdot \phi(s, a')^{\mathrm{T}}$
20:     $b \leftarrow b + \phi(s, a') \cdot \frac{R_{\max}}{1-\gamma}$
21:   end for
22: end for
23: Return $\theta = A^{-1}b$
```

Algorithm 2 LSPI-Rmax

```
1: Input:
     • $\phi$: feature function ($k$ is the number of features)
     • $\gamma$: discount factor
     • $\epsilon_d, m$: thresholds for $(\epsilon_d, m)$-known state–actions
2: Initialize $\theta$, $s_1$, and set $D \leftarrow \emptyset$
3: for $t = 1, 2, 3, \cdots$ do
4:   Choose the greedy action $a_t$ in $s_t$ with respect to $Q_\theta$,
     observe reward $r_t$ and $s_{t+1}$
5:   $D \leftarrow D \cup \{\langle s_t, a_t, r_t, s_{t+1} \rangle\}$
6:   Update $\theta$ by running LSPI with LSTD-Rmax
7: end for
```

too expensive. A better way is to employ smarter nearest-neighbor search techniques such as kd-trees [12] whose time complexity is sub-linear in $|D|$, but still is prohibitively expensive in high dimension state spaces. Therefore, it might be worthwhile to sacrifice exact counting for faster computation. In our implementation, we coarsely discretize $S \times A$ into bins, and the number of nearest neighbors is approximated by counting how many samples in D fall in the same bin. Thus, all samples in a bin are simultaneously known or unknown under this approximation.[1] By maintaining a counter for each bin, we are able to achieve a much lower

[1]This implementation also has a desired side-effect of smoothing the optimal value function of the known-state MDP, which makes it easier to learn the optimal value function and policy for the known-state MDP.

time complexity that is linear in the state dimension. Therefore, this binning implementation can be viewed as a computationally efficient approximation to the exact nearest-neighbor search procedure such as the kd-tree algorithms. We note that better choices for identifying known state–actions are possible in the presence of domain knowledge, and the number of bins may grow sub-exponentially.

It is important to note that the discretization procedure we use here should not be confused with discretization for solving large MDPs, as mentioned at the beginning of Section 4. Discretization there directly decides the complexity of value functions being considered, and solving a discretized model often requires repeated access to the whole model (*e.g.*, in the case of dynamic programming [20]). Here, in contrast, discretization is used only to indicate whether enough samples have been observed in a local region, that is, whether a state–action pair is known. Thus, the class of value functions under consideration and the computational complexity mostly depend on number of features, rather than number of bins used by LSPI-RMAX. These differences make our approach less prone to the curse of dimensionality than discretization methods.

A second problem is that D keeps growing without limit as time goes on. However, if the MDP dynamics are smooth[2] (which is often the case in practice), it is unnecessary to keep all samples in D if many of them are close to each other. Thus, we have chosen to avoid adding a sample to D if the corresponding bin size reaches a predefined capacity, which varied between 50 and 100 in our experiments in Section 5. Excluding samples has the effect of making sample distribution in D more uniform, which is often preferred for the policy-improvement step of LSPI in practice [17].

Third, steps 18 to 21 ensure that untried actions a' in s are preferred when a nearby state is visited in the future. Empirically, it is desirable to avoid adding too many such artificial samples, which may overwhelm the real samples. This can be done in a number of ways. In our implementation, an artificial sample is used only when the bin corresponding to (s, a') is empty. This choice has the desired property that if the local region of (s, a') is underrepresented, the artificial sample has the effect of increasing the value of a' in s so as to encourage exploration; on the other hand, if a' has been tried in nearby states of s', the algorithm will use those real samples and does not use artificial ones.

Finally, LSPI is invoked in each step in Algorithm 2. Since LSPI is rather expensive (the complexity of each iteration is on the order of $O(k^3)$) and adding a sample to the training set D usually has ignorable effects on the value function it finds, our implementation calls LSPI (step 6 in Algorithm 2) only after a certain number of samples are added to D.

4.3 Discussion

Because LSPI-RMAX's root is in RMAX, it is natural to consider the relation between them. Given a finite MDP, RMAX estimates $Q(s, a)$ for each (s, a). If we view this tabular representation as a linear function approximation such that there is an independent feature per state–action pair, then LSPI-RMAX precisely duplicates RMAX when $\epsilon_d = 0$. (Note that RMAX has a similar parameter m.) In fact, what LSTD-RMAX solves for is the value function of policy π on the empirical known-state MDP $\hat{M}_K$ maintained by RMAX. Consequently, step 6 of Algorithm 2 becomes exact policy iteration and returns the optimal value function of $\hat{M}_K$.

[2]Nearby sates have similar transition and reward functions if the MDP's dynamics are smooth.

This connection to RMAX serves as a sanity check of the plausibility of our approach. We can make this similarity formal. A thorough analysis of LSPI-RMAX is difficult, since a nontrivial performance guarantee for LSPI itself is open except in limited situations [1]. Our preliminary analysis is specific to the bin-based implementation described in the previous section. Similar to known states, we call a bin to be *known* if every action has been tried at least m times in states in this bin. Suppose the state space is partitioned into C disjoint bins. Let $K \subseteq \{1, 2, \cdots, C\}$ be the subset of *known* bins, then the known-bin MDP can be defined as $M_K = \langle S, A, T_K, R_K, \gamma \rangle$ where

$$R_K(s,a) = \begin{cases} R(s,a) & \text{if } s \text{ is in a known bin} \\ R_{\max} & \text{otherwise,} \end{cases}$$

$$T_K(s,a,s') = \begin{cases} T(s,a,s') & \text{if } s \text{ is in a known bin} \\ \mathbb{I}(s = s') & \text{otherwise,} \end{cases}$$

where $\mathbb{I}$ is the set-indicator function.

Assumption 1 below asserts that the LSPI algorithm, when using the feature function ϕ, is able to return a near-optimal policy in known-bin MDP.

ASSUMPTION 1. *Let $\epsilon, \delta \in (0,1)$ be given. Let $m = m(\epsilon, \delta)$ be some positive constant depending on ϵ and δ. We call C a* cover number *of the MDP if the state space S can be partitioned into C disjoint bins: $S = S_1 \cup S_2 \cup \cdots \cup S_C$, such that, with probability at least $1-\delta$, LSPI with the given set of features returns an ϵ-optimal policy in the known-state MDP M_K for any $K \subseteq \{1, 2, \ldots, C\}$.*

This assumption allows one to show a bound on the sample complexity of exploration stated in Proposition 1, whose proof follows similar steps as the sample complexity proof for RMAX [13]. Although it is hard to verify Assumption 1 in practice, the theorem serves as a best-case sanity check that the algorithm does employ the data-efficiency of RMAX. An important open question is to generalize the analysis to broader classes of linear function approximation.

PROPOSITION 1. *When it is implemented using bins $S_1, \ldots, S_C$ that are defined in Assumption 1, LSPI-*RMAX*'s sample complexity of exploration is*

$$O\left(\frac{CAm\left(\epsilon, \frac{\delta}{CA}\right)}{\epsilon^3 (1-\gamma)^6} \log \frac{CA}{\delta} \right).$$

LSPI-RMAX is similar to metric E^3 in the sense that they both depend on some kind of smoothness assumption. The notation of (ϵ_d, m)-known state–actions plays a similar role to the local modeling assumption. Metric E^3 estimates the model explicitly based on samples and then employs a hypothetical planner to get the desired policy. In contrast, LSTD-RMAX builds a compressed empirical model implicitly [5] and approximate policy iteration is then used to solve the planning problem. The near-optimality part in Assumption 1 is roughly the analog of the approximate planner assumption made in metric E^3. Since LSPI does not make a clean distinction between learning and planning, it is not clear how to use the same assumptions that applied to metric E^3 to analyze LSPI-RMAX.

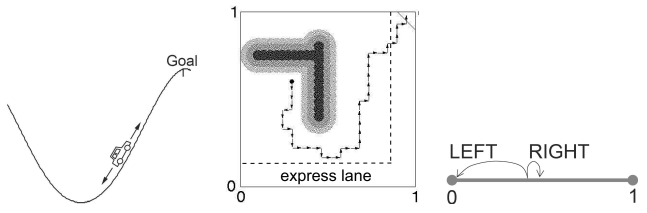

Figure 1: From left to right: MountainCar (borrowed from Sutton and Barto [24]), ExpressWorld (adapted from Sutton [23]), ContCombLock. The fourth domain, Bicycle, is not illustrated here.

5. EMPIRICAL STUDIES

This section reports experimental results on four continuous, episodic domains ordered by increasing difficulty of exploration. We did not try to optimize features, which is an important problem. Further investigation is needed to study the effect of features on exploration.

5.1 Setup

MountainCar [24] is a problem in which the agent tries to drive a car to a hilltop (see Figure 1(a)). To do this, the agent has to reverse the car to move away from the goal and then apply full throttle until it reaches the hilltop. This problem has two continuous state variables and three actions. Every step gives rise to a reward of -1, and the state transition is governed by a system of nonlinear equations [24]. We used CMAC features consisting of 3 layers of 4×4 tilings, which are then repeated for each of the three actions, leading to a total of $3 \times 3 \times 4 \times 4 = 144$ features. The same approach of repeating features for every action was adopted in the other problems.

Bicycle is the problem of balancing a bicycle. We used the deterministic version of bicycle [21] (*i.e.*, the noise in the displacement action is always zero) to illustrate how LSPI-Rmax works in a challenging, high dimensional problem, although the problem is somewhat easier than the original, stochastic one. There are six continuous state variables and two continuous action variables. Each balancing step leads to a reward of $+1$, and an episode terminates when the bicycle falls. We used the same set of hand-coded features and the same discretization in the continuous action space as in the original LSPI paper [17]; that is, there were 100 features and 5 actions.

ExpressWorld is adapted from PuddleWorld [23]. In PuddleWorld, the state space is a two-dimensional continuous grid world in which the agent can move along four directions (N, E, S, and W) to reach the goal region in the north-east corner while trying to avoid two puddles (see Figure 1(b)). Each step yields a -1 reward plus a penalty for entering the puddle region. To make exploration more important in this task, we add an "express lane" 0.15 units wide—if the agent moves within this lane, every immediate reward is -0.5 instead of -1. Start states are drawn randomly from the left half plane so that the agent has to learn how to avoid puddles, as well as to explore actively to discover the goal as well as the express lane. We have found empirically that this is a quite challenging exploration task. Partly because of the reward function that has sharp changes in the puddle region, it is not easy to find good features for this problem. Therefore, we simply used a 6×6 discretization of the state space and treated it as a CMAC feature with one layer of gridding, which resulted in a total of $4 \times 6 \times 6 = 144$ features.

The last problem, ContCombLock, is a continuous version of combination lock, which was designed to require a smart exploration strategy [16]. The state space is a segment $[0, 1]$ with a fixed start state 0 (Figure 1(c)). There are two actions: left always takes the agent back to the start state 0; the other action right takes the agent from state x to a new state $y = x + 0.02 + \Delta$, where Δ is generated via Gaussian noise with mean 0 and standard deviation 0.005. If the agent reaches a state $x > 0.98$, the episode terminates. Every step results in a -1 reward. Therefore, the optimal policy is to always choose right, and on average each episode takes about 50 steps to finish. We used CMAC features that have 3 layers of grids, each dividing the state space into 6 pieces. Hence, $2 \times 3 \times 6 = 36$ features were used.

For Bicycle, the agent was allowed to run 2000 episodes, each of which was at most 72000 steps long; in the other problems, the agent had to run up to 200 episodes, each of which was at most 300 steps long. The discount factor was set to 0.99 for all four problems.

In the experiments, the capacity of a bin was set to 100 except in Bicycle where it was 50 (since this problem's state space is larger). We invoked LSPI in LSPI-Rmax every time 100 new samples were added to D, except in ContCombLock where LSPI was invoked immediately after a new sample was added.

We compared LSPI-Rmax against LSPI with ϵ-greedy and counter-based exploration. For each of them, we have tried different exploration parameters and report the best result. Since all problems are continuous, we had to modify the counter-based method to use the same trick as LSPI-Rmax (*cf.*, Section 4.2), and maintained a counter for each bin.

5.2 Results

Figure 2 shows the learning curves for cumulative reward of LSPI with different exploration rules in all four problems, where the y-axis is the cumulative rewards, averaged over 30 runs. The slope of a curve corresponds to per-episode reward. In MountainCar and Bicycle, all three exploration rules had similar cumulative rewards for the first dozen episodes, but LSPI-Rmax quickly showed its advantage over the other two and converged to a much better policy. It is also observed that ϵ-greedy and counter-based exploration rules can be better than the other, depending on specific problems. In ContCombLock, LSPI-Rmax was the only one that succeeded.

The learning curve for ExpressWorld probably best illustrates the way LSPI-Rmax works. At the beginning of learning, LSPI-Rmax actually receives much less cumulative rewards since it attempts to visit different parts of the state space and thus receives a lot of penalty for entering puddles. However, after about 10 episodes, it has obtained a better policy, which finally compensates the cost of exploration at the beginning. In contrast, LSPI with ϵ-greedy and counter-based exploration converged to suboptimal policies (visible in the graph as their reward slopes).

The success of LSPI-Rmax comes from the fact that the agent actively explores the state space. This claim is supported by Figure 3, which plots the states visited by the three agents in ExpressWorld for the first three episodes

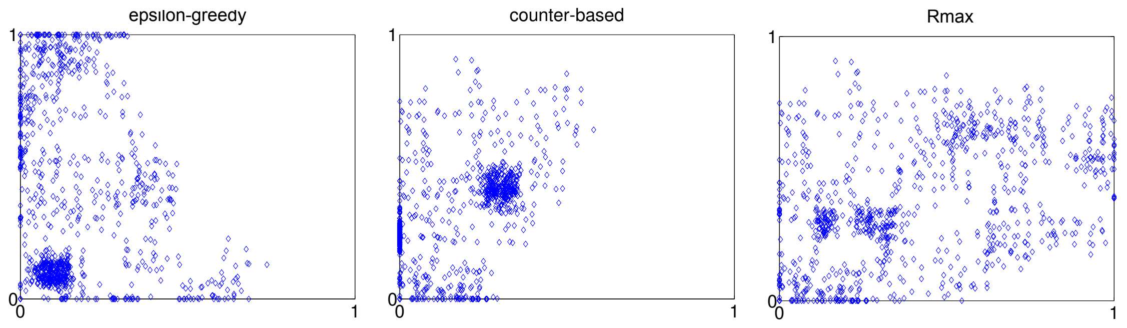

Figure 3: State visitation of the first three episodes of a typical run in ExpressWorld.

during a typical run. (We chose this domain for demonstration because it is easier to visualize its 2D state space). Observe that all three agents failed to reach the goal for the first three episodes. However, the LSPI-Rmax agent apparently behaves more intelligently by experiencing novel parts of the state space, while the other two agents had difficulty getting far away from the start states on the left half plane.

Finally, we study the effect of parameter m on cumulative rewards in LSPI-Rmax. Figure 4 plots the average per-episode reward, as well as standard deviation, of LSPI-Rmax during the entire 30 runs in ExpressWorld with different threshold m values. The results demonstrate how m controls the exploration-exploitation tradeoff. When m is small, the agent tries to exploit sooner, but risks at ending up with less effective policies, which explains the large variance in the average per-episode reward. When m gets larger, the agent becomes more conservative and tends to explore more before exploiting. Consequently, learning is more robust and the variance is small. However, being conservative comes with costs as the algorithm delays exploitation while exploring. In between, medium m values work best. Similar patterns are observed in other problems.

6. CONCLUSIONS

This paper raises a number of interesting problems for future research. LSPI-Rmax has worked well on several other benchmark problems, including CartPole [3] and Acrobot [24] that are not included here. In the future, we would like to test it on more realistic control problems. A very challenging problem is to extend the formal analysis for LSPI-Rmax without making the somewhat hard-to-verify Assumption 1. Ideally, we would like to have sample complexity bounds that depend on the number of features rather than the number of state variables. A possible direction is to use Gaussian processes as the function approximator [11], which readily provide confidence intervals that could be useful for exploration. A third problem is to investigate model-based approaches (*e.g.*, [18]) as opposed to the model-free one studied here in large problems. Finally, it is interesting to extend the algorithm to handle continuous actions.

Efficient exploration in continuous environments is a fundamental problem in real-life reinforcement learning. This paper takes a step towards this goal by proposing a plausible, online LSPI variant that borrows ideas from efficient exploration techniques for finite MDPs. A few principled and practical issues are discussed, and insights are drawn by relating it to existing works. The algorithm's effectiveness is demonstrated in four benchmark problems.

7. REFERENCES

[1] A. Antos, C. Szepesvári, and R. Munos. Learning near-optimal policies with Bellman-residual minimizing based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

[2] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 30–37, 1995.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Trans on Systems, Man, and Cybernetics*, 13:835–846, 1983.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, September 1996.

[5] J. A. Boyan. Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

[6] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pages 369–376, 1995.

[7] S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

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

[9] C.-S. Chow and J. N. Tsitsiklis. An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Trans on Automatic Control*, 36(8):898–814, 1991.

[10] M. O. Duff. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts, Amherst, MA, 2002.

[11] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML-05)*, pages 201–208, 2005.

[12] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans on Mathematical Software*, 3:209–226, 1977.

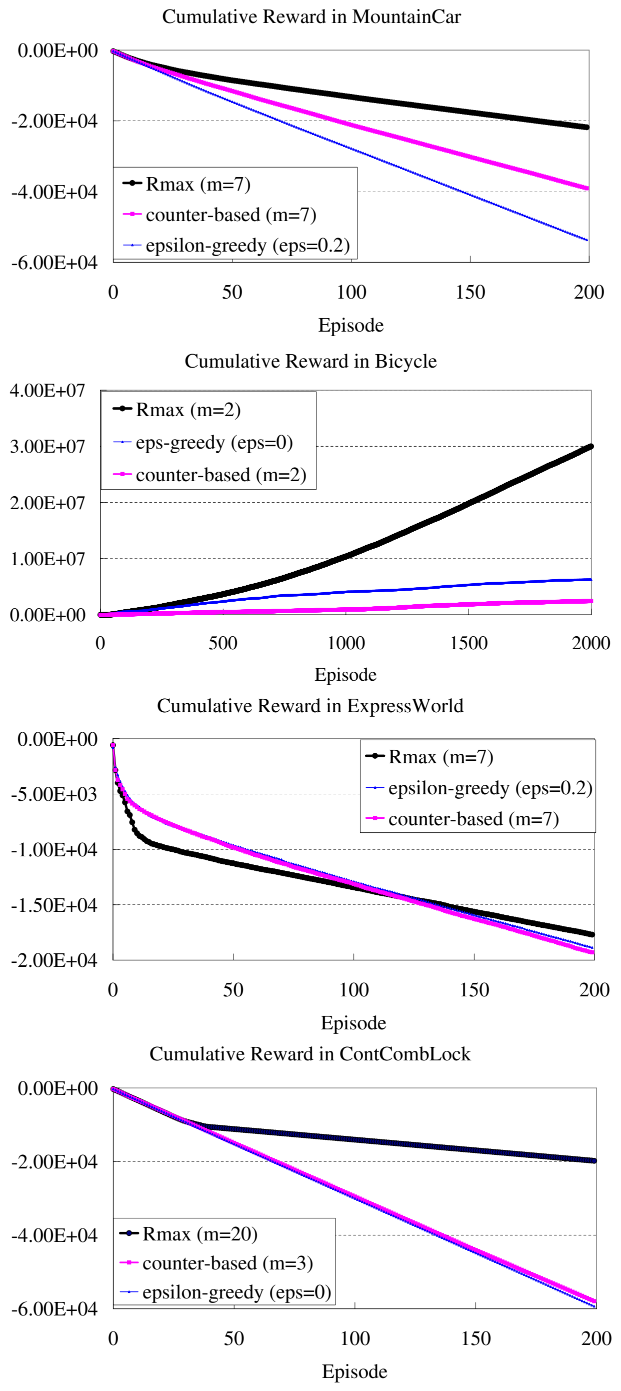

Figure 2: Learning curves for cumulative rewards with three exploration strategies. All results are averaged over 30 independent runs.

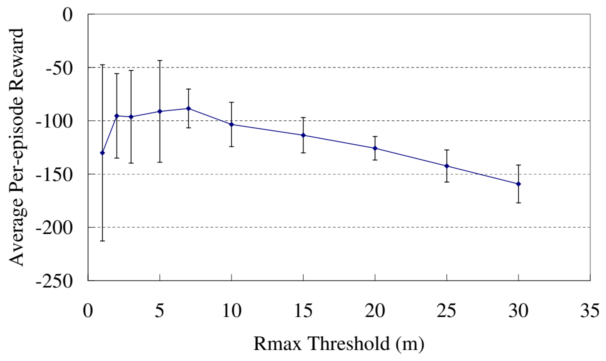

Figure 4: Effect of exploration threshold m on average per-episode reward in ExpressWorld.

[13] S. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, UK, 2003.

[14] S. Kakade, M. J. Kearns, and J. Langford. Exploration in metric state spaces. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 306–312, 2003.

[15] M. J. Kearns and S. P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.

[16] S. Koenig and R. G. Simmons. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22:227–250, 1996.

[17] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[18] A. Nouri and M. L. Littman. Multi-resolution exploration in continuous spaces. In *Advances in Neural Information Processing Systems 21 (NIPS-08)*, 2009.

[19] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, pages 697–704, 2006.

[20] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, New York, 1994.

[21] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*, pages 463–471, 1998.

[22] A. L. Strehl and M. L. Littman. Online linear regression and its application to model-based reinforcement learning. In *Advances in Neural Information Processing Systems 20 (NIPS-07)*, pages 1417–1424, 2008.

[23] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044, 1996.

[24] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, March 1998.

[25] S. Thrun. The role of exploration in learning control. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pages 527–559. 1992.