Technical Report DCS-TR-641

# Exploration in Least-Squares Policy Iteration

**Lihong Li** and **Michael L. Littman** and **Christopher R. Mansley**

{lihong,mlittman,cmansley}@cs.rutgers.edu
RL$^3$ Laboratory
Department of Computer Science
Rutgers University
Piscataway, NJ 08854

First created: March, 2007
Last modified: October 9, 2008

**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.

**Keywords**: PAC-MDP, exploration, least-squares policy iteration, Markov decision processes, reinforcement learning.

# Exploration in Least-Squares Policy Iteration

Lihong Li and Michael L. Littman and Christopher R. Mansley

## 1 Introduction

One of the key problems in reinforcement learning (Sutton & Barto, 1998) 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 computationally intractable and it is often not obvious how to select a prior distribution for learning (Duff, 2002). We only consider non-Bayesian approaches in this paper. Thrun (1992) 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* (Kakade, 2003) in finite-state Markovian environments. 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.

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 (Brafman & Tennenholtz, 2002), into a state-of-the-art learning algorithm, least-squares policy iteration (LSPI) (Lagoudakis & Parr, 2003). 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 validity of our approach is then supported by 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 (Puterman, 1994), 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 $s$, and $\gamma \in [0, 1]$ is a discount factor.

A policy maps states to actions: $\pi : S \to A$. Given a policy $\pi$, we define the state-value function, $V^\pi(s)$, as the expected cumulative reward received by executing $\pi$ 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 $\pi$ thereafter. A reinforcement-learning agent attempts to *learn* an optimal policy $\pi^*$ whose value functions are denoted by $V^*(s)$ and $Q^*(s, a)$, respectively. It is known that $V^* = \max_\pi V^\pi$ and $Q^* = \max_\pi Q^\pi$. A greedy policy $\pi_Q$ with respect to a value function $Q$ is one that selects actions with maximum Q-values; namely, $\pi_Q(s) = \operatorname{argmax}_a Q(s, a)$. The greedy policy with respect to $Q^*$ is optimal.

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 (Puterman, 1994). 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.

## 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 and has succeeded in a number of challenging control problems such as riding a simulated bicycle (Lagoudakis & Parr, 2003). It adopts the approximate policy-iteration framework and uses a model-free version of least-squares temporal difference learning (LSTD) (Bradtke & Barto, 1996; Boyan, 2002) 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 $\phi$: $Q_\theta(s, a) = \theta^{\mathrm{T}} \cdot \phi(s, a)$.

LSPI is arguably the most competitive reinforcement-learning algorithm available in large environments. Being an approximate policy-iteration algorithm, LSPI is stable and theoretically sound. 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.

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. Lagoudakis and Parr (2003) 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. Their sample-collection approach, however, is not always applicable in practice. An online agent has to decide wisely how to collect samples autonomously, while random walks could be inefficient or even disastrous. In Section 4.1, we show how to augment the original LSPI with efficient online exploration.

### 3.2 Exploration in MDPs

The simplest and most popular exploration rule is probably *$\epsilon$-greedy*: the agent chooses the greedy action with respect to its value function with probability $1-\epsilon$, and a random action with probability $\epsilon$. An alternative approach, called *counter-based* exploration, has shown empirical advantages over $\epsilon$-greedy in some problems (Thrun, 1992). 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 $\epsilon$-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 (Kearns & Singh, 2002; Kakade, Kearns, & Langford, 2003), in which metric $E^3$ is probably the most relevant 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 small 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 (Brafman & Tennenholtz, 2002), which implements the *optimism-in-the-face-of-uncertainty* heuristic. 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*) $\hat{M}_K = \langle S, A, \hat{R}, \hat{P}, \gamma \rangle$ 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 State Spaces

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 (Chow & Tsitsiklis, 1991). 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 it generalizes values between states.

Exploration rules for finite MDPs can be extended to LSPI when it is applied online. For example, LSPI with $\epsilon$-greedy exploration picks a greedy action with probability $1-\epsilon$ and a random action with probability $\epsilon$; 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 predefined by the agent designer. 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 heuristic 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: $d : (S \times A) \times (S \times A) \to \mathbb{R}^+ \cup \{0\}$. 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 $(\epsilon_d, m)$-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 $\epsilon_d$-*close* to $(s, a)$. Furthermore, a state $s$ is called $(\epsilon_d, m)$-known if $(s, a)$ is $(\epsilon_d, m)$-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.

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, $\frac{R_{\max}}{1-\gamma}$, to states and state–action pairs that are not $(\epsilon_d, m)$-known. LSPI-RMAX is just LSPI, except that it 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. The first natural question is how to decide whether a state–action pair is $(\epsilon_d, m)$-known or not. A straightforward algorithm would be to search over all samples in $D$ and count the number of nearest neighbors within $\epsilon_d$ distance. However, when $D$ is large, even this $O(|D|)$-time algorithm is far too expensive. A better way is to employ smarter nearest-neighbor search techniques such as kd-trees (Friedman, Bentley,

**Algorithm 1** LSTD-RMAX

Input: $D$, $\epsilon_d$, $m$, $\pi$, $\phi$, $\gamma$, $k$ (number of features)
Output: weight vector $\theta$ for predicting $Q^\pi$
$A \leftarrow \mathbf{0}_{k\times k}$ (the $k \times k$ zero matrix)
$b \leftarrow \mathbf{0}_k$ (the $k \times 1$ zero column-vector)
**for all** $\langle s, a, r, s' \rangle \in D$ **do**
  **if** $(s, a)$ is $(\epsilon_d, m)$-known **then**
    **if** $s'$ is $(\epsilon_d, m)$-known **then**
      $A \leftarrow A + \phi(s, a) \cdot \big(\phi(s, a) - \gamma\phi(s', \pi(s'))\big)^{\mathrm{T}}$; $b \leftarrow b + \phi(s, a) \cdot r$
    **else**
      $A \leftarrow A + \phi(s, a) \cdot \phi(s, a)^{\mathrm{T}}$; $b \leftarrow b + \phi(s, a) \cdot \left(r + \frac{\gamma R_{\max}}{1-\gamma}\right)$
    **end if**
  **else**
    $A \leftarrow A + \phi(s, a) \cdot \phi(s, a)^{\mathrm{T}}$; $b \leftarrow b + \phi(s, a) \cdot \frac{R_{\max}}{1-\gamma}$
  **end if**
  **for all** $a' \in A \setminus \{a\}$ where $(s, a')$ is not $(\epsilon_d, m)$-known **do**
    $A \leftarrow A + \phi(s, a') \cdot \phi(s, a')^{\mathrm{T}}$; $b \leftarrow b + \phi(s, a') \cdot \frac{R_{\max}}{1-\gamma}$
  **end for**
**end for**
Return $\theta = A^{-1}b$

& Finkel, 1977) 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 complexity that is linear in the state dimension. 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 (Puterman, 1994)). Here, in contrast, discretization is used only to indicate whether a state–action pair is known. Thus, the class of value functions under consideration and the computational complexity mostly depend on number

[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.

**Algorithm 2** LSPI-RMAX

1: Input: $\epsilon_d$, $m$, $\phi$, $\gamma$
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**

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 (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 was between 50 and 100 in our experiments. Excluding samples has the effect of making sample distribution in $D$ more uniform, which is often preferred for LSPI in practice (Lagoudakis & Parr, 2003).

Third, since solutions found by LSTD are biased by the sample distribution in the input sample set $D$, steps 15 to 17 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. Empirically, it is desirable to avoid adding too many such artificial samples, which can be done easily in a number of ways.

Finally, LSPI is invoked in each step in Algorithm 2. Since LSPI is rather expensive and adding a single sample usually has ignorable effects on the value function, 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 $\epsilon_d = 0$. In fact, what LSTD-RMAX solves for is the value function of policy $\pi$ 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 $\hat{M}_K$.

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. Our preliminary analysis is specific to the bin-based implementation described in the previous section. Based on Assumption 1 below, we may show a bound on the sample complexity of exploration stated in Theorem 1, whose proof follows similar steps as the sample complexity

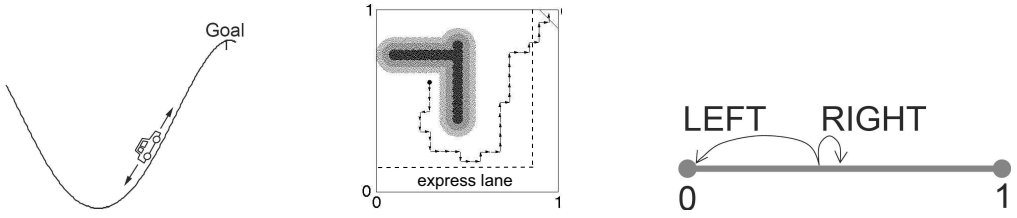


Figure 1: From left to right: MountainCar (borrowed from Sutton and Barto (1998)), ExpressWorld (adapted from Sutton (1996)), ContCombLock. The fourth domain, Bicycle, is not illustrated here.

proof for Rmax (Kakade, 2003).

**Assumption 1.** *Let $\epsilon, \delta \in (0,1)$ be given. Let $m = m(\epsilon, \delta) \in \mathbb{N}$ be some constant depending on $\epsilon$ and $\delta$. 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 $\epsilon$-optimal policy in the known-state MDP $M_K$ for any $K \subseteq \{1, 2, \cdots, C\}$, where at least $m(\epsilon, \delta)$ samples $(s, a)$ with $s \in S_k$ are drawn for each action $a \in A$ whenever $k \in K$.*

**Theorem 1.** *The sample complexity of exploration for LSPI-*Rmax *is*

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

*when it is implemented using bins $S_1, \ldots, S_C$ that are defined in Assumption 1.*

LSPI-Rmax is similar to metric $E^3$ in the sense that they both depend on some kind of smoothness assumption. The notation of $(\epsilon_d, m)$-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 (Boyan, 2002) 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.

## 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 orthogonal problem to exploration. MountainCar (Sutton & Barto, 1998) 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 (Sutton & Barto, 1998). We used CMAC features consisting of 3 layers of $4 \times 4$ 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 (Randløv & Alstrøm, 1998) (*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 (Lagoudakis & Parr, 2003); that is, there were 100 features and 5 actions.

EXPRESSWORLD is adapted from PUDDLEWORLD (Sutton, 1996). 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 $6 \times 6$ 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 (Koenig & Simmons, 1996). 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 $\Delta$ 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. We compared LSPI-RMAX against LSPI with $\epsilon$-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 discretization trick as LSPI-RMAX (*cf.*, Section 4.2), and maintained a counter for each bin.

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 $\epsilon$-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 $\epsilon$-greedy and counter-based exploration converged to suboptimal policies (visible in the graph as their reward slopes).

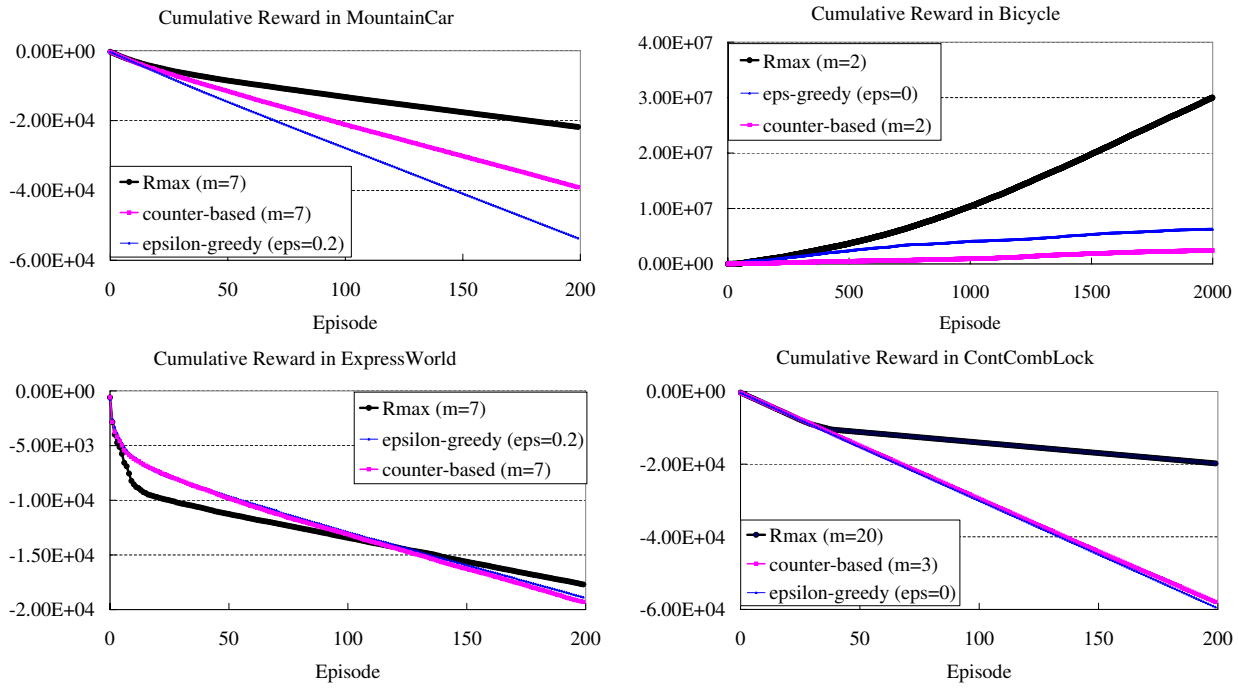


Figure 2: Learning curves for cumulative rewards, averaged over 30 runs.

The success of LSPI-RMAX comes from the fact that the agent actively explores the state space. This claim is supported by Figure 3(left), which plots the states visited by the three agents in EXPRESSWORLD for the first three episodes during a typical run. (We chose this domain for demonstration because it is

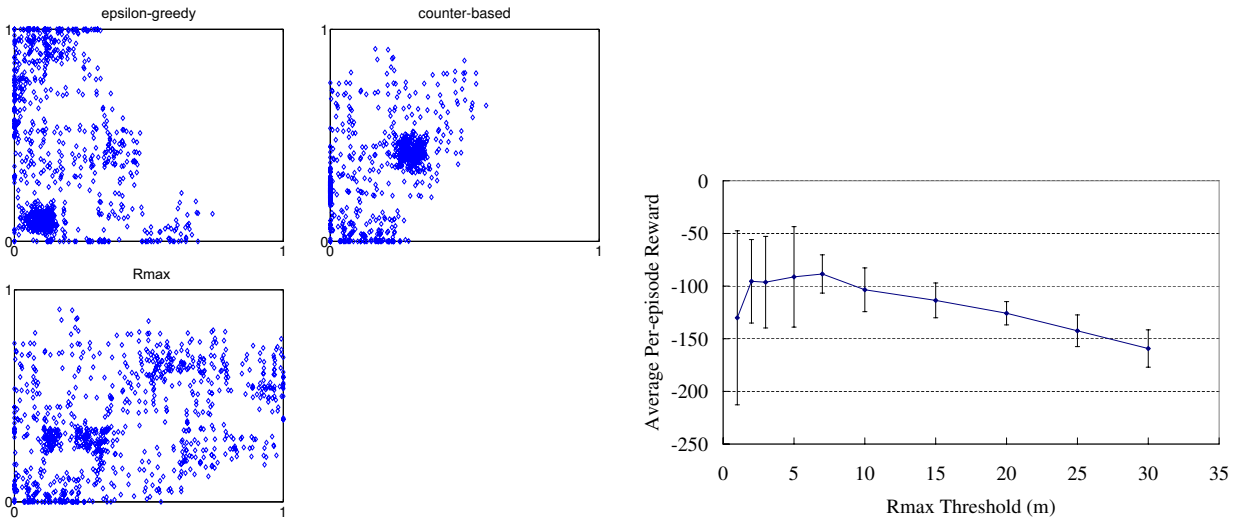


Figure 3: Left: state visitation of the first three episode of a typical run in ExpressWorld . Right: effect of exploration threshold $m$ on average per-episode reward in ExpressWorld .

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 trying to experience novel parts of the state space, while the other two agents experienced difficulty in getting far away from the start states.

Finally, we study the effect of parameter $m$ on cumulative rewards in LSPI-Rmax. Figure 3(right) 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 (Barto, Sutton, & Anderson, 1983) and Acrobot (Sutton & Barto, 1998) that are not included here due to space limitations. 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 (Engel, Mannor, & Meir, 2005), which readily provide confidence intervals that could be useful for exploration. A third problem is to investigate model-based approaches as opposed to the model-free one studied here.

Efficient exploration in continuous environments is a very interesting problem fundamental to real-life reinforcement learning. In this paper, we have taken 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 illustrated in four benchmark problems.

## References

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

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

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*, 33–57.

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

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

Duff, M. O. (2002). *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. Ph.D. thesis, University of Massachusetts, Amherst, MA.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of ICML*, pp. 201–208.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans on Mathematical Software*, *3*, 209–226.

Kakade, S. (2003). *On the Sample Complexity of Reinforcement Learning*. Ph.D. thesis, University College London, UK.

Kakade, S., Kearns, M. J., & Langford, J. (2003). Exploration in metric state spaces. In *Proceedings of ICML*, pp. 306–312.

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

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

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, *4*, 1107–1149.

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

Randløv, J., & Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of ICML*, pp. 463–471.

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

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Thrun, S. (1992). The role of exploration in learning control. In White, D. A., & Sofge, D. A. (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pp. 527–559.