Action Refinement in Reinforcement Learning by Probability Smoothing

Thomas G. Dietterich TGD@CS.ORST.EDU
Department of Computer Science, 102 Dearborn Hall, Oregon State University, Corvallis, OR, 97331

Dídac Busquets DIDAC@IIIA.CSIC.ES
Ramon López de Màntaras MANTARAS@IIIA.CSIC.ES
Carles Sierra SIERRA@IIIA.CSIC.ES
Artificial Intelligence Research Institute (IIIA), Spanish Council for Scientific Research (CSIC), Campus UAB, 08193 Bellaterra, Barcelona, Spain

Abstract

In many reinforcement learning applications, the set of possible actions can be partitioned by the programmer into subsets of similar actions. This paper presents a technique for exploiting this form of prior information to speed up model-based reinforcement learning. We call it an action refinement method, because it treats each subset of similar actions as a single "abstract" action early in the learning process and then later "refines" the abstract action into individual actions as more experience is gathered. Our method estimates the transition probabilities $P(s'|s, a)$ for an action a by combining the results of executions of action a with executions of other actions in the same subset of similar actions. This is a form of "smoothing" of the probability estimates that trades increased bias for reduced variance. The paper derives a formula for optimal smoothing which shows that the degree of smoothing should decrease as the amount of data increases. Experiments show that probability smoothing is better than two simpler action refinement methods on a synthetic maze problem. Action refinement is most useful in problems, such as robotics, where training experiences are expensive.

1. Introduction

In model-based reinforcement learning, experience gained during exploration is employed to learn models of the state-action transition function and the reward function. From these learned models, the optimal policy can be computed by incremental or batch dynamic programming algorithms such as prioritized sweeping (Moore & Atkeson, 1993) and value iteration (Bertsekas, 1995). Model-based methods are appropriate when the state and action spaces are relatively small and finite and when the exploration is being performed on a physical system (as opposed to a simulator). In such cases, each exploratory action is expensive, and we seek reinforcement learning methods that can find near-optimal policies with very little exploratory training data. In contrast, if the reinforcement learner is interacting with an inexpensive simulator, then model-free methods such as SARSA(λ) and Q learning may be preferred. Although they generally require many more training experiences to learn a good policy (Moore & Atkeson, 1993), they require less storage space, because they do not need to represent the model.

In all forms of learning, the primary way to reduce the need for training data is to incorporate some kind of prior knowledge. Previous work on abstraction in reinforcement learning has studied prior knowledge in the form of subgoals, subroutines, and state abstraction (Parr & Russell, 1998; Sutton et al., 1999; Dietterich, 2000). These allow the user to express prior knowledge about the structure of the policy and about which state variables are relevant for different subgoals or subroutines. By applying such prior knowledge, the reinforcement learning algorithm can generalize across states—that is, it can identify cases in which the optimal policy in one state s_1 is identical to the optimal policy in another state s_2.

In this paper, we explore a weaker form of prior knowledge that does not generalize across states, but instead generalizes across actions. We study the case in which the user knows that two actions a_1 and a_2 have similar effects in the world, independent of the states in which they are executed. This allows the learning algorithm to conclude that the transition probabilities and rewards for a_1 and a_2 are similar.

There are many situations in which a reinforcement learning agent may have actions with similar effects and rewards. For example, consider a mobile robot whose set of available actions includes two forward motion actions. One action moves the robot until the wheel shaft encoders indicate that the robot has moved one meter. Another action moves the robot for 5 seconds. Much of the time these two actions will have similar effects, but there are situations (e.g., when the robot is stuck and the wheels do not turn) in which the one-meter action will loop forever, so the 5-second action would be a better choice. Another example might arise in routing packets on the internet using Q-routing (Boyan & Littman, 1994), where each action corresponds to routing a packet to a neighboring switch. Some of the neighboring switches might be very close to each other, and therefore, routing a packet to any one of them would have similar effects.

Even in cases where the final reinforcement learning agent does not have several similar actions, it may be useful to consider many similar actions when designing the agent. To illustrate this point, consider the mobile robot navigation problem that we have been studying (Busquets et al., 2002). Our robot has low level primitive actions (operating motors to turn wheels and cameras, invoking various image processing routines), but learning at the level of such actions would be very inefficient. Instead, as many others have done (e.g., (Asada et al., 1996)), we have designed a state space and an action space that focus the reinforcement learning on the aspects of the navigation task that we are unable to directly program by hand.

The robot navigates by finding visual landmarks. The robot's camera has a viewing angle of 60 degrees. We partitioned the space around the robot into six 60-degree sectors such that if the robot points the camera successively in the center of each sector, it will obtain a complete view of the surrounding environment. Then we designed a high-level action called "Move While Looking for Landmarks (MLL)" in which the robot moves forward while aiming its camera in one of the six sectors to search for new visual landmarks. We defined this action such that the robot always looks in the sector having the fewest known landmarks. But we can imagine many variants of this action that might be useful in some states. For example, we could have high-level actions in which the camera always looks in the direction of forward motion or always looks in the sector having the most distant landmarks or always looks in the sector containing the goal landmark. We could also define six MLL actions, one for each sector and let the robot decide which sector to examine with the camera.

As designers, we do not know which of these actions (or action sets) would be most useful. One way to find out would be to include all of these actions in the MDP and let the reinforcement learning system determine which actions are useful. But this would vastly increase the amount of exploration required to learn a good policy. Another possibility would be to train the robot several times, each time with a different set of actions. But that would require even more training experiences.

One thing we know about these different variants of "Move While Looking for Landmarks" is that they all have similar behavior. This suggests that we could somehow treat them as an equivalence class early in the learning process and then later allow the learning algorithm to discriminate among them.

We will employ the term *action refinement* to describe any process by which a reinforcement learning algorithm initially treats a set of similar actions as a single abstract action and later refines that abstract action into individual actions. We will assume that the programmer has partitioned the available actions into action sets $A_1, A_2, \ldots, A_L$. Each action set A_l is a subset of the set A of available actions, and by defining A_l, the programmer is specifying the prior knowledge that any two actions $a, a' \in A_l$ have similar effects and receive similar rewards. We will assume that each action a belongs to exactly one of the action sets.

In this paper we investigate a method of action refinement based on smoothing probability estimates. The paper begins with a mathematical derivation of the probability smoothing method. This derivation tells us the form of the optimal smoothing procedure, but it makes some unrealistic assumptions. We replace those assumptions with some more conservative ones that we verify experimentally give reasonable behavior. The paper then introduces two competing action refinement methods and compares them experimentally on a stochastic maze problem. The results show that the probability smoothing method is the best method. To finish the paper, we investigate the sensitivity of the method to the size and correctness of the action sets A_l and to the correctness of the conservative assumptions introduced in the derivation of the method. The paper shows that probability smoothing is a very efficient and easy-to-implement action refinement method with excellent performance.

2. The Probability Smoothing Method

2.1 Definitions

We consider the standard case where a reinforcement learning agent is interacting with an unknown but fully-observable Markovian environment. The Markovian environment contains a finite set of states S and affords a finite set of actions A. At each time t, the agent observes the current state s_t of the environment, chooses an action $a_t \in A$, and executes the action. The environment makes a transition to state s_{t+1} and returns an immediate reward r_t. The programmer has grouped the actions A into L disjoint action sets $A_1, \ldots, A_L$ to indicate subsets of actions that are "similar".

The agent explores the environment by choosing actions according some exploration policy and recording the resulting $\langle s_t, a_t, r_t, s_{t+1}\rangle$ experience tuples. At certain chosen times t, the agent computes a model M_t of the environment by computing statistics from the stored tuples. Let $N_t(s,a)$ be the number of times action a has been executed in state s and $N_t(s,a,s')$ be the number of times this resulted in a transition to state s'. Let $W_t(s,a,s')$ be the total of the rewards in the corresponding tuples $\langle s,a,r,s'\rangle$ (i.e., the total of the rewards r received when a caused a transition from s to s').

The maximum likelihood model M_t consists of P_t and R_t defined as follows:

$$P_t(s'|s,a) = \frac{N_t(s,a,s')}{N_t(s,a)} \quad R_t(s,a,s') = \frac{W_t(s,a,s')}{N_t(s,a,s')}.$$

Let us define the probability smoothing model, $\mathcal{M}_t$, to consist of $\mathcal{P}_t$ and $\mathcal{R}_t$ computed as follows:

$$\mathcal{P}_t(s'|s,a) = \frac{\sum_{a'\in A_l} \lambda_{a'} N_t(s,a',s')}{\sum_{a'\in A_l} \lambda_{a'} N_t(s,a')} \quad (1)$$

$$\mathcal{R}_t(s,a,s') = \frac{\sum_{a'\in A_l} \lambda_{a'} W_t(s,a',s')}{\sum_{a'\in A_l} \lambda_{a'} N_t(s,a',s')}. \quad (2)$$

In these formulas, A_l represents the action set containing action a. To estimate the transition and reward functions for a, we combine the counts and rewards of all of the actions a' in A_l, each weighted by a parameter $\lambda_{a'}$. For action a, we will fix λ_a to have the constant value 1, but for the other actions, we must determine the optimal values of the smoothing parameters $\lambda_{a'}$.

2.2 Derivation of Optimal Smoothing Parameters

The approach we will pursue to determine the optimal smoothing parameters is the following. First, we will select a measure of the error between the true probability $P(s'|s,a)$ and the predicted probability $\mathcal{P}(s'|s,a)$. Then we will compute the expected value of this error with respect to fixed-sized samples of action a and its fellow actions in A_l. This expected value will be a function of the λ values, so we can determine the optimal values of the λ's by differentiating with respect to each λ, setting to zero, and solving the resulting system of equations.

The error measure that we will employ is the squared difference between $P(s'|s,a)$ and $\mathcal{P}(s'|s,a)$:

$$J(s,a) = \sum_{s'} [P(s'|s,a) - \mathcal{P}(s'|s,a)]^2.$$

We will begin with the simplest possible case and then elaborate it later. Suppose A_l comprises only two actions: a_1 and a_2. Let us focus on predicting $P(s'|s,a_1)$ for specific states s and s'. Let $p_1 = P(s'|s,a_1)$ and $p_2 = P(s'|s,a_2)$. We can view these as two coins with probability of heads p_1 and p_2, respectively. Let $H_1 = N(s,a_1,s')$ be the number of heads observed for the first coin after $N_1 = N(s,a_1)$ coin tosses, and define H_2 and N_2 analogously. In this simple case,

$$\mathcal{P}(s'|s,a_1) = \hat{p}_1 = \frac{H_1 + \lambda H_2}{N_1 + \lambda N_2}.$$

Let D be the joint probability distribution of H_1 and H_2 given samples of fixed size N_1 and N_2. Then our goal is to choose λ to minimize the expected total prediction error:

$$\begin{aligned}
TPE(\lambda) &= E_D[J(s,a)] \\
&= E_D[(p_1-\hat{p}_1)^2 + (1-p_1-(1-\hat{p}_1))^2] \\
&= E_D[2(p_1-\hat{p}_1)^2] \\
&= 2E_D[(p_1-\hat{p}_1)^2] \\
&= 2E_D[p_1^2 - 2p_1\hat{p}_1 + \hat{p}_1^2] \\
&= 2\left(p_1^2 - 2p_1 E_D[\hat{p}_1] + E_D[\hat{p}_1^2]\right).
\end{aligned}$$

In the last line, we have employed the fact that p_1 is a constant and expectation is a linear operator. Now we must determine the first and second moments of $\hat{p}_1$, which are $E_D[\hat{p}_1]$ and $E_D[\hat{p}_1^2]$.

We can compute the first moment by substituting the definition of $\hat{p}_1$ and using the fact that H_1 and H_2 are binomial random variables. Recall that the first moment of a binomial random variable with sample size N and parameter p is pN. Hence,

$$\begin{aligned} E_D[\hat{p}_1] &= E_D\left[\frac{H_1+\lambda H_2}{N_1+\lambda N_2}\right] \\ &= \frac{E_D[H_1]+\lambda E_D[H_2]}{N_1+\lambda N_2} \\ &= \frac{p_1 N_1+\lambda p_2 N_2}{N_1+\lambda N_2}. \end{aligned}$$

We follow the same procedure to compute the second moment:

$$\begin{aligned} E_D[\hat{p}_1^2] &= E_D\left[\left(\frac{H_1+\lambda H_2}{N_1+\lambda N_2}\right)^2\right] \\ &= E_D\left[\frac{H_1^2+2\lambda H_1 H_2+\lambda^2 H_2^2}{(N_1+\lambda N_2)^2}\right] \\ &= \frac{E_D[H_1^2]+2\lambda E_D[H_1 H_2]+\lambda^2 E_D[H_2^2]}{(N_1+\lambda N_2)^2} \end{aligned}$$

To simplify this further, we exploit the fact that $E_D[H_1 H_2] = E_D[H_1]E_D[H_2]$, because H_1 and H_2 are independent random variables. We also use the second moment of a binomial random variable, which is $pN(1-p+pN)$. Combining these, we have

$$E_D[\hat{p}_1^2] = \frac{\begin{array}{r} p_1 N_1(1-p_1+p_1 N_1)+2\lambda p_1 N_1 p_2 N_2 + \\ \lambda^2 p_2 N_2(1-p_2+p_2 N_2) \end{array}}{(N_1+\lambda N_2)^2}$$

Now we substitute the expressions for the first and second moments back into the expression for $TPE(\lambda)$ and simplify to obtain

$$TPE(\lambda) = 2\frac{N_1 p_1(1-p_1)+\lambda^2 N_2^2\epsilon^2+\lambda^2 N_2 p_2(1-p_2)}{(N_1+\lambda N_2)^2},$$

where we have introduced $\epsilon = |p_1 - p_2|$ to represent the difference between the probabilities of the two coins.

Differentiating this and setting it to zero gives

$$\lambda = \frac{p_1(1-p_1)}{N_2\epsilon^2+p_2(1-p_2)}.$$

We can also write this as

$$\lambda = \frac{V_1}{N_2\epsilon^2+V_2}, \tag{3}$$

where $V_1 = p_1(1-p_1)$ is the variance of the first coin, and $V_2 = p_2(1-p_2)$ is the variance of the second coin.

This formula has the following properties. If the variance of coin 1 is high, then λ increases, because there is a greater need to reduce this high variance by smoothing with the second coin. If the variance of coin 2 is high, λ decreases, because the second coin is less useful for smoothing. As the coins become less similar (i.e., as ϵ increases), λ decreases, because smoothing with the second coin will increase the bias. Finally, as N_2 gets larger, λ decreases, again to avoid increasing the bias. Note that N_1 does not appear in the formula. If N_1 grows large while N_2 remains constant, then the counts H_1 from the first coin will dominate the estimate $\hat{p}_1$, and no change in λ is needed. We can state this as a theorem:

Theorem 1 *$\hat{p}_1$ is a consistent estimator of p_1 even if N_2 also grows without bound. In other words,*

$$\lim_{N_1\to\infty}\hat{p}_1 = p_1 \quad \textit{and} \quad \lim_{N_1\to\infty}\lim_{N_2\to\infty}\hat{p}_1 = p_1$$

Proof: If N_2 is constant, then

$$\begin{aligned} \lim_{N_1\to\infty}\hat{p}_1 &= \lim_{N_1\to\infty}\frac{H_1/N_1+\lambda H_2/N_1}{1+\lambda N_2/N_1} \\ &= \frac{p_1+0}{1+0} = p_1 \end{aligned}$$

If N_2 is allowed to grow, then either $\epsilon = 0$, in which case $\lambda = 1$ and $\hat{p}_1$ is always unbiased or else $\epsilon > 0$, and $\lim_{N_2\to\infty}\lambda = 0$, which makes $\hat{p}_1$ unbiased in the limit. **Q.E.D.**

A consequence of this theorem is that in the limit, model-based reinforcement learning with probability smoothing will converge to the optimal policy.

Figure 1 plots λ for the case where $p_1 = 0.5$ and $N_1 = N_2$. Note that even when $\epsilon = 0.3$ and $N_1 = N_2 = 20$, it is still worthwhile to use a non-zero λ. Also note that if $\epsilon = 0.1$, and we have 100 samples of each coin, the optimal value of λ is still around 0.2, which is surprisingly large.

There are two ways that we must extend the analysis. First, suppose A_l contains more than two different actions. For the case where we have a third coin with H_3 heads observed in N_3 tosses (and with probability p_3 of heads), $\hat{p}_1$ becomes

$$\hat{p}_1 = \frac{H_1+\lambda_2 H_2+\lambda_3 H_3}{N_1+\lambda_2 N_2+\lambda_3 N_3}.$$

Let $\epsilon_2 = p_2 - p_1$ and $\epsilon_3 = p_3 - p_1$, then the optimal values for λ_2 and λ_3 are

$$\begin{aligned} \lambda_2 &= \frac{\epsilon_3^2 N_3 V_1 - \epsilon_2\epsilon_3 N_3 V_1 + V_1 V_3}{\epsilon_3^2 N_3 V_2 + \epsilon_2^2 N_2 V_3 + V_2 V_3} \\ \lambda_3 &= \frac{\epsilon_2^2 N_2 V_1 - \epsilon_2\epsilon_3 N_2 V_1 + V_1 V_2}{\epsilon_3^2 N_3 V_2 + \epsilon_2^2 N_2 V_3 + V_2 V_3}. \end{aligned}$$

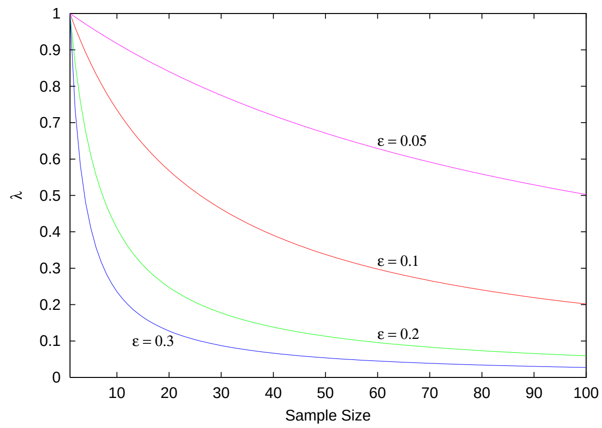

Figure 1. Optimal values of λ for sample size $N_1 = N_2$ and various values of ϵ.

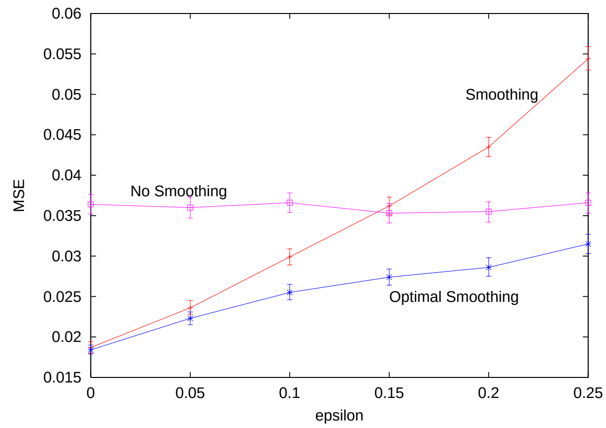

Figure 2. Comparison of Mean Squared Error of Adaptive Smoothing with assumed values $p_1 = 0.1$, $\epsilon = 0.05$, and Laplace smoothing of 0.50 when the true value of $p_1 = 0.1$ and for various values of the true ϵ.

This expression is complicated, because if ϵ_2 and ϵ_3 are of opposite sign, then their biases counteract each other, so the value of λ can be increased. On the other hand, if $\epsilon_2 = \epsilon_3 = \epsilon$, then $p_2 = p_3$, and $\lambda_2 = \lambda_3 = V_1/(\epsilon^2(N_2 + N_3) + V_2)$. Hence, the λ values decrease significantly.

The second way to extend the analysis is to consider actions with $B > 2$ possible outcomes. In this case, we view $P(\cdot|s, a_1)$ as a multinomial distribution with parameters $p_1^{(1)}, p_1^{(2)}, \ldots, p_1^{(B)}$. Let the transition probabilities of action a_2 be defined analogously as $\{p_2^{(i)}\}_{i=1}^B$, and define $\epsilon^{(i)} = p_2^{(i)} - p_1^{(i)}$. With these definitions, the optimal value for λ has the same form as Eqn. 3, but with the following substitutions:

$$\begin{aligned} V_1 &:= \left[\sum_{i=1}^{B-1} p_1^{(i)}\left(1 - p_1^{(i)}\right)\right] - \left[\sum_{i<j}^{B-1} p_1^{(i)} p_1^{(j)}\right] \\ V_2 &:= \left[\sum_{i=1}^{B-1} p_2^{(i)}\left(1 - p_2^{(i)}\right)\right] - \left[\sum_{i<j}^{B-1} p_2^{(i)} p_2^{(j)}\right] \\ \epsilon^2 &:= \left[\sum_{i=1}^{B-1} \left(\epsilon^{(i)}\right)^2\right] + \left[\sum_{i<j}^{B-1} \epsilon^{(i)} \epsilon^{(j)}\right] \end{aligned}$$

2.3 Determining the Level of Smoothing in Practice

During reinforcement learning, we will not know the true values of p_1, p_2, or ϵ, so we cannot use Eqn. 3 directly. A naive approach to choosing λ would be to plug the maximum likelihood estimates of p_1, p_2, and ϵ into Eqn. 3. However, these maximum likelihood estimates have such high variance that the computed values of λ tend to vary wildly, and the resulting estimates of $\hat{p}_1$ are poor.

Instead, we adopt an approach we call "default smoothing" in which we assume default values for p_1, p_2, and ϵ in the λ formula, and plug in the value of N_2 from the real data. To choose these default values, we plotted TPE curves for various assumed and true values of the probabilities. When the true value of p_1 is near 0.5, *any* non-zero value of λ gives excellent results, because coin 1 has maximum variance. But as the true value of p_1 becomes small, the variance of coin 1 becomes quite small, and it is harder to choose good default values to plug into the formula. After some trial and error, we selected default values of $p_1 = 0.1$, $p_2 = 0.15$, and (hence) $\epsilon = 0.05$. Although these values will be conservative (i.e., they will never produce $\lambda = 1$), they work well when $\epsilon < 0.15$ for all values of p_1.

In addition, we found that incorporating mild Laplace smoothing into the estimator helped slightly. Specifically, we employ

$$\mathcal{P}_t(s'|s, a) = \frac{\frac{1}{2|S'|} + \sum_{a' \in A_l} \lambda_{a'} N_t(s, a', s')}{\frac{1}{2} + \sum_{a' \in A_l} \lambda_{a'} N_t(s, a')}, \tag{4}$$

where S' is the set of all states that have resulted from executing any of the actions $a' \in A_l$.

Figure 2 shows the behavior of default smoothing based on simulated data. It plots the mean squared error (which is $TPE(\lambda) - V_1$) against the true value of ϵ when the true value of $p_1 = 0.1$ and $N_1 = N_2 = 5$ examples. It compares no smoothing (in which the maximum likelihood estimate is employed), with optimal smoothing (in which the true values of p_1 and

Table 1. Transition probability distribution for the four action modifiers.

	Probability of moving		
Modifier:	Straight	Left	Right
Left	0.7	0.2	0.1
Right	0.7	0.1	0.2
Left2	0.6	0.3	0.1
Right2	0.6	0.1	0.3

ϵ are used to compute λ) and default smoothing. We see that for $\epsilon < 0.14$, default smoothing gives better performance than no smoothing. This represents the worst case that we simulated. Plots for larger true values of p_1 show default smoothing performing nearly as well as optimal smoothing.

In the multinomial case (i.e., where the actions have more than two possible resulting states), we employed the binary formula with the same default values. That is, we fixed $V_1 = 0.09$, $V_2 = 0.1275$, and $\epsilon^2 = 0.0025$. In the case where A_l contains more than 2 actions, we considered the worst case where $\epsilon_i = 0.05$ for all of the other actions a_i, $i > 1$. In other words, we replaced N_2 in Eqn. 3 with $\sum_{i>1} N_i$.

In all cases, we employed the same λ values to estimate $\mathcal{R}(s, a, s')$ as we used for estimating $\mathcal{P}(s'|s, a)$.

3. Experimental Study of Action Refinement

To evaluate the effectiveness of probability smoothing for action refinement, we designed a toy maze problem (see Figure 3). There are 81 non-terminal states and 16 actions grouped into four action sets: {NorthLeft, NorthLeft2, NorthRight, NorthRight2}, {SouthLeft, SouthLeft2, SouthRight, SouthRight2}, {EastLeft, EastLeft2, EastRight, EastRight2}, {WestLeft, WestLeft2, WestRight, WestRight2}. These 16 actions are constructed from the cross-product of the four compass directions {North, South, East, West} with four modifiers {Left, Left2, Right, Right2}. Each modifier corresponds to a probability distribution that gives the probability of moving straight (in the selected compass direction), left, and right. Table 1 shows these four probability distributions. According to this table, for example, the probabability of moving north (straight) when executing a NorthLeft action is 0.7; the probability of moving west (left) is 0.2; and the probability of moving east (right) is 0.1.

A reward of -0.04 is given for each action (including cases where the agent bumps into a wall). There are two terminal states, one of which gives a reward $+1$

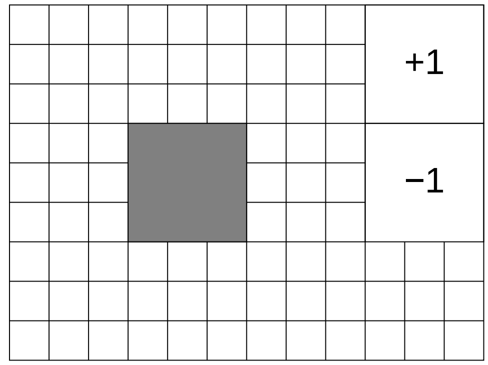

Figure 3. Maze world employed in the experiments. The squares marked $+1$ and -1 are terminal states with the indicated terminal rewards.

(for a net reward of 0.96) and the other of which gives a reward of -1 (for a net reward of -1.04). In each trial, the agent is started in a random non-terminal state, and the trial terminates when the agent enters one of the two terminal states. This problem is based on the 4x3 world in Russell & Norvig's (1995) textbook.

To measure the performance of a policy π on this problem, we compute the value function V^π and then sum the values of all 81 non-terminal states. By this criterion, the optimal policy has a total value of 43.37.

We employed the following model-based learning procedure. At each time t, the agent chooses an action at random and executes it. It collects the resulting tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ and saves these tuples. Every 100 steps, it computes estimates $\mathcal{P}_t$ and $\mathcal{R}_t$ and applies value iteration to compute the optimal policy of the corresponding estimated MDP. The total value of this policy (applied to the true MDP) is computed and output. This process was repeated for 100,000 action steps. The entire learning procedure was repeated 100 times (with different random number seeds), and the results analyzed to compute 95% confidence intervals on the total value of the policy every 100 steps. (Replicating only 30 times gives the same results but with slightly wider confidence intervals.)

We compared probability smoothing with three alternative methods. First, we performed the experiment with no smoothing at all by setting $\lambda = 0$.

Second, we performed the experiment with $\lambda = 1$. This treats all actions in each action set as being completely equivalent, and the resulting probability estimates $P(\cdot|s, a)$ will be identical.

Finally, we performed the experiment with only four actions (NorthRight, SouthRight, EastRight, and

WestRight), one from each action set. The rationale for this fourth method is that because the actions within each action set are very similar, perhaps it would suffice to just choose one of them.

Figure 4 compares no smoothing with probability smoothing. The horizontal axis shows the number of exploratory actions that have been performed per state-action pair. The graph shows that probability smoothing gives significantly better performance than the non-smoothing methods. Consider, for example, the amount of training required for the performance of each policy to reach a total value of 35. Without smoothing, each action must be executed 51.2 times in every state (on the average). With probability smoothing, each action must only be executed 10.2 times—a speedup of a factor of 5. This shows that we can train 16 actions with probability smoothing using less experience than we would need to train only 4 actions without probability smoothing. From a design perspective, probability smoothing allows us evaluate four versions of each action at no additional cost. (We repeated this experiment (data not shown) with different values for V_1, V_2, and ϵ in Eqn. 3. The results are nearly identical even when $V_1 = 0.25$, $V_2 = 0.8$, and $\epsilon = 0.3$.)

Figure 5 compares probability smoothing with fixed smoothing ($\lambda = 1$) and with the four-action method. There are no statistically significant differences until approximately 9.3 exploration steps, at which point both probability smoothing and the four-action method do significantly better than $\lambda = 1$ smoothing. At about 23 exploration steps, probability smoothing begins to significantly out-perform the four-action method. At 77 steps, probability smoothing has attained a total reward of 41.5, the four-action method a total reward of 38.9, and the fixed smoothing method a total reward of 34.8. This shows that $\lambda = 1$ smoothing and the four-action method both give speedups similar to probability smoothing during the early phases of training. They fail later in the process for different reasons. The $\lambda = 1$ smoothing fails because it is biased. The four-action method fails because the four actions are not sufficient to produce the optimal policy.

These results show that probability smoothing performs better than any of the three alternatives. It is vastly superior to no smoothing, and in the limit of large training sets, it performs better than either fixed smoothing (which will introduce large biases into the probability estimates) or the four-action method (which will not be able to find the optimal policy).

We were curious about the sensitivity of probability smoothing to the correctness and size of the action sets. We conducted two experiments. First, we var-

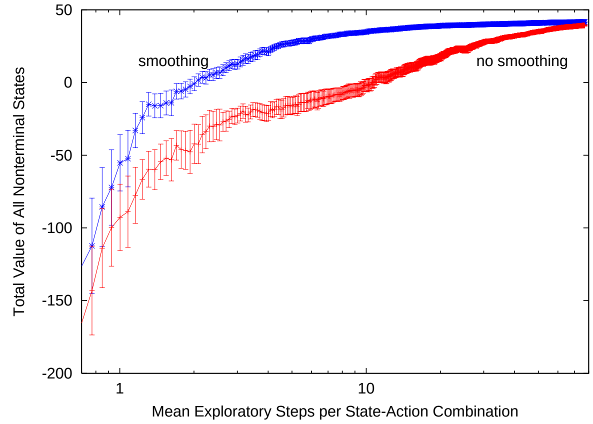

Figure 4. Comparison of probability smoothing with no smoothing ($\lambda = 0$). Note log scale on horizontal axis.

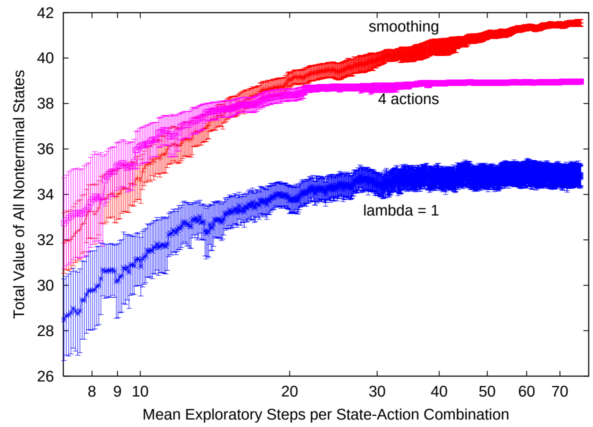

Figure 5. Comparison of probability smoothing with fixed smoothing ($\lambda = 1$) and the four-action method

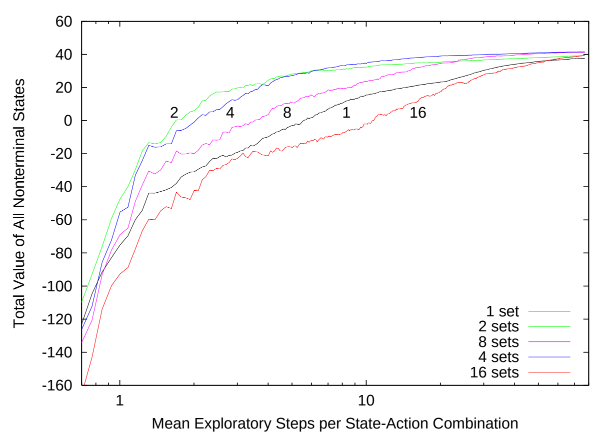

Figure 6. Comparison of probability smoothing with different numbers of equivalence classes

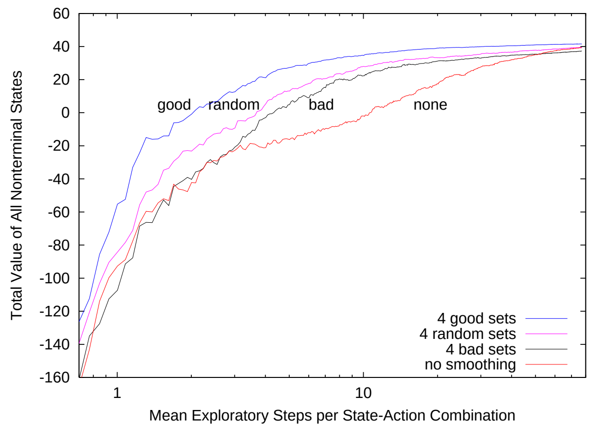

Figure 7. Behavior of probability smoothing with action sets of varying quality

ied the number of action sets from 1, 2, 4, 8, and 16. The actions were distributed equally across the action sets, and similar actions were grouped together to the extent possible. Figure 6 shows that by varying the number and size of the action sets, we can produce different bias/variance tradeoff points. 16 separate action sets gives high variance (because it defeats probability smoothing). A single action set gives high bias, because probability smoothing combines the probability transition models of all of the actions. Four sets of 4 actions and two sets of 8 actions gave the best performance during the early part of the curve.

In the second experiment, we varied the degree to which the actions within an action set were similar. Figure 7 shows that at small sample sizes, a poor grouping of actions can be worse than no smoothing at all. At intermediate sample sizes, even random groupings give better performance than no smoothing, because they are able to trade some increased bias for a reduction in variance. At large sample sizes, however, the bias in the random and bad groupings leads to worse performance than either no smoothing or well-chosen action sets. This is because, not surprisingly, the default smoothing parameters are setting λ too large for the poorly-chosen action sets.

4. Concluding Remarks

This paper has introduced a simple method, probability smoothing, for achieving action refinement in model-based reinforcement learning. Action refinement can be employed to speed up RL applications in which there are many actions but where the actions can be partitioned into sets of similar actions. In addition, with probability smoothing, the problem of designing a set of good actions in a reinforcement learning application is significantly eased, because all of the different actions can be included in the reinforcement learning problem with little or no increase in the training cost.

The parameter settings for probability smoothing can be determined from theoretical computations—no tuning or calibration is required on the actual MDP (unlike $TD(\lambda)$ or Q learning). In addition, probability smoothing can be easily implemented within the code for performing Bellman backups in value iteration, policy iteration, and prioritized sweeping. We are currently applying it to help design actions in our robot navigation project.

Acknowledgements

The authors acknowledge the support of the US NSF under grant IIS-0083292, the Commission for Cultural, Educational, and Scientific Exchange between Spain and the US, and the Spanish Plan Nacional Project DPI 2000-1352-C02-02. Dídac Busquets holds the CIRIT doctoral scholarship 2000FI-00191.

References

Asada, M., Noda, S., Tawaratsumida, S., & Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, *23*, 279–303.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Belmont, MA: Athena Scientific.

Boyan, J. A., & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. *NIPS-94* (pp. 671–678). Morgan Kaufmann, San Francisco.

Busquets, D., Lopez de Mantaras, R., Sierra, C., & Dietterich, T. G. (2002). Reinforcement learning for landmark-based robot navigation. *AAMAS-02*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

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

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *NIPS-98* (pp. 1043–1049). Cambridge, MA: MIT Press.

Russell, S. J., & Norvig, P. (1995). *Artificial intelligence. A modern approach*. Englewood Cliffs, NJ: Prentice-Hall.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.