Efficient Exploration for Reinforcement Learning

Eric Wiewiora

May 29, 2004

Abstract

Reinforcement learning is often regarded as one of the hardest problems in machine learning. Algorithms for solving these problems often require copious resources in comparison to other problems, and will often fail for no obvious reason. This report surveys a set of algorithms for various reinforcement learning problems that are known to terminate with good solution after a number of interactions with the domain that is polynomial in it's parameters. These algorithms are said to be solving the exploration problem. We analyze these algorithms in the probably approximately correct (PAC) framework, after a brief introduction to this powerful method. We see that the runtime of efficient exploration algorithms appears to depend on the method with which the learner samples from the domain, and offer an explanation for why is the case. Finally, a brief review of recent work and related areas is provided.

1 Introduction

Reinforcement learning (RL) regards an agent that learns to make good sequences of decisions. Unlike classification, decisions made in the reinforcement learning context influence the future decisions that will be faced. The agent attempts to optimize a reward signal that is received after every decision. Because current and future decisions are not independent, the agent must consider the future implications of as well as the immediate consequences of each decision. See [Bertsekas and Tsitsiklis, 1996] or [Sutton and Barto, 1998] for an in-depth introduction to this learning problem.

In this survey, we will examine theoretically justified algorithms for finding ways to make good decisions. The ease with which a good solution can be found will depend on how an agent collects information about the problem it faces. When the agent has full knowledge of the environment it is in, efficient methods for finding optimal decisions are well known. When the agent does not have full knowledge of the domain, it must learn about it through experiencing the consequences of decisions. The decisions that an agent makes in order to learn about the domain are referred to as exploration.

1.1 Exploration or exploitation

When the agent does not have enough information to build an accurate model of its environment, it must make a choice as to how to sample. Is it better to sample in order to build a more accurate representation, or is it better to exploit the knowledge

already available in order to accumulate reward? This is the fundamental exploration/ exploitation dilemma that appears throughout decision theory.

Several strategies have been proposed for balancing exploration with exploitation. There is a class of simple strategies known as greedy in the limit of infinite exploration (GLIE). These strategies randomly choose between making the decision thought to be best (the greedy choice), or to take some other one. They guarantee that after an infinite number of samples, an accurate representation of the domain is learned. Also, as more samples are collected, the probability the greedy choice is taken approaches one. Because each decision will eventually be sampled enough to accurately estimate the expected long-term reward, the greedy choice becomes optimal. See [Thrun, 1992] for a review of such methods.

In many plausible situations, interleaving exploration and exploitation is not sensible [Langford et al., 2002]. Suppose an agent only has a finite number of decisions to make. Also, assume that each decision used for exploration can only improve the result of future exploitations. In this situation, it is obvious that the expected reward for any set of decisions composed of m explorations and n exploitations is maximized when all explorations happen before any exploitations.

The algorithms presented in this paper build on this intuition. They attempt to explore as efficiently as possible, and then return with knowledge of make near optimal decisions in the future. The challenge of designing these algorithms is not only in finding how to explore quickly, but also to identify when a sufficient amount of exploration has taken place.

The first comprehensive analysis of algorithms for efficient exploration was compiled in Sham Kakade's doctoral thesis [Kakade, 2003]. This survey will rely heavily on the structure of this work, and will cover many of the same algorithms. In addition to the content of this thesis, we will examine algorithms for a special class of reinforcement learning problems called bandit problems. We will also survey advances in algorithms for reinforcement learning since the thesis' publication.

2 Reinforcement learning preliminaries

The domain the agent makes its decisions in is usually modeled as a Markov Decision Process (MDP). An MDP M is a 4-tuple $\langle S, A, P, R \rangle$:

- S is a set of states. Each state must satisfy the *Markov property*: Given the state the agent is currently in, the agent's history of past states and decisions is independent of the future states the agent finds itself in. Unless stated otherwise, we assume that the states are finite set, and the agent is able to determine the state it is in. See [Littman et al., 2001] for a discussion of methods for deriving a set of states when these assumptions are not met.
- A is a set of decisions available to the agent. We will assume that this set is finite, and that all decisions are available to the agent at each state.
- $P(s'|s, a)$ is a function that defines the probability of transitioning to state s' after decision a is taken when the agent is in state s. Because of the Markov property, the probability of a transition to state s' only depends on the prior state and decision made.
- The function $R(s, a)$ determines the probability of receiving reward r after choosing a in state s. We will assume rewards range between 0 and 1.

A policy is a description of how an agent makes decisions. In general, a policy can be any function that maps an agent's total history to a probability distribution over the

decisions [Puterman, 1994]. In this paper, we will restrict ourselves to policies that are deterministic, non-stationary and Markov. A Markov policy does not make use of the agent's history to make a decision. A deterministic policy will always make the same decision, given the current input. A non-stationary policy may change depending on the number of decisions to be made in the future. We will write a non-stationary policy as $\pi(s \in S, t \in \mathbf{I}^+) \rightarrow a \in A$, where t is the number of decisions to be made in the future. A stationary policy will not change its decision based on the number of future decisions. We write these policies as $\pi(s \in S) \rightarrow a \in A$.

2.1 T-step reward

An agent wants a policy that is expected to accumulate a large amount of reward over a number of decisions. A natural way to formalize this is to have the agent maximize the expected reward received after making T decisions. We write this as

$$E[\sum_{t=1}^{T} \frac{r_t}{T}],$$

where r_t is the reward received after decision t. Note that if rewards range in $[0, 1]$, so does this expected sum. Many other objective functions for measuring long-term reward have been proposed, but most can be approximated with T-step reward. See [Puterman, 1994] for a thorough analysis of different objectives for reinforcement learning.

The expected sum of reward over the next T decisions will depend on the current state the agent is in. We define the state value function for policy π starting in state s with t decisions to make as $V^\pi(s, t)$. The value function can be written as:

$$V^\pi(s, t) = E[\sum_{i=1}^{t} \frac{r_i}{t} | s_1 = s, a_i = \pi(s_i, t - i + 1)].$$

The value function also satisfies a recursion relation:

$$V^\pi(s, t) = \frac{1}{t} E[R(s, \pi(s, t))] + \frac{t - 1}{t} \sum_{s'} P(s'|s, \pi(s, t)) V^\pi(s', t - 1).$$

For any MDP, there is some Markov, deterministic, non-stationary policy π^* that optimizes T-step reward [Puterman, 1994]. A policy π' is said to be ϵ-optimal at state s if

$$V^*(s, T) - V'(s, T) \leq \epsilon.$$

Above we use the convention that the value function of a policy will share the same superscript.

2.2 Discounted reward

In the literature, the most common objective for RL is to maximize the expected infinite sum of discounted rewards:

$$E[(1 - \gamma) \sum_{t=1}^{\infty} \gamma^{t-1} r_t].$$

Here γ is a discount factor in the range $[0, 1)$. As γ decreases, immediate reward contributes more to the sum. We add a normalizing factor $(1 - \gamma)$ to ensure this objective will be in the same range as the individual rewards[Kakade, 2003].

As with the T-step objective, we define a state value function for a policy π and discount factor γ:

$$V^{\pi}(s,\gamma) = E[(1-\gamma)\sum_{t=1}^{\infty}\gamma^{t-1}r_t|s_1 = s, a_i = \pi(s_i)].$$

This value function also has an associated recursion relation:

$$V^{\pi}(s,\gamma) = (1-\gamma)E[R(s,\pi(s))] + \gamma\sum_{s'}T(s'|s,\pi(s))V^{\pi}(s',\gamma).$$

For any fixed γ, there is a Markov, deterministic and stationary policy π^* that maximizes discounted reward [Puterman, 1994]. Policies that are ϵ-optimal for discounted reward are defined analogous to the T-step objective.

The discounted reward criteria offers some advantages over T-step reward. First, discounted reward considers the influence of future decisions over an infinite horizon instead of a fixed horizon. An agent optimizing T-step reward has no incentive to wind up in a "good" state after T decisions are made. Also, an optimal T-step policy may choose differently in the same state, depending on how distant the horizon is. Thus, an optimal T-step policy may take a factor T more memory to store.

In spite of the advantages described above, the T-step objective has become more popular in recent work. There are many reasons for this. First, the theoretical properties of algorithms for finding optimal T-step policies are better understood. Also, policies for T-step reward tend to be more robust when approximation is required. See [Bagnell et al., 2003] for a discussion of the advantages of T-step reward, and see [Fern et al., 2003] for an example of T-step reward being used to approximate discounted reward.

3 PAC analysis

We will be analyzing algorithms for finding policies that have performance guarantees in the PAC learning framework [Kearns and Vazirani, 1994]. Under the PAC framework, two parameters are provided to the algorithm: ϵ, the amount the algorithms solution can be suboptimal, and δ, the maximum tolerable probability of failing to find such a solution. When a PAC algorithm terminates, the policy that is returned achieves an expected return of $(1-\epsilon)$ with probability greater than $(1-\delta)$.

In order to prove an algorithm is PAC learner, we must be able to bound the probability that the algorithm will return a bad answer. We will do this by using bounds from probability theory.

The union bound states that the probability of event A or event B occurring is less than the sum of the probabilities the individual events will occur:

$$Pr(A \text{ OR } B) \leq P(A) + P(B).$$

This bound makes no assumption about the independence of the two events.

The second probability bound we will use is known as the Markov bound. This states that for any nonnegative random variable x and positive number k:

$$Pr(x \geq k \cdot E[x]) \leq \frac{1}{k}.$$

The last bound needed for our analysis is the Hoeffingdon bound (sometimes referred to as the Chernoff bound). This bound states that the sum of m random variables,

each having value in the range $[0, 1]$ cannot deviate from the expectation by a large degree. Call the expectation of the individual variables p, and the observed sum S. The probability that the sum deviates from it's expectation by an additive factor of η is bounded by the following equations:

$$\begin{aligned} Pr(S > (p+\eta)m) &\leq \exp(-2m\eta^2), \\ Pr(S < (p-\eta)m) &\leq \exp(-2m\eta^2). \end{aligned}$$

4 Finding π^* with the MDP

We first consider algorithms that have full access to the MDP. This allows the direct use of the functions P and R in order to find an optimal policy. Because the full model is available, exploration is not necessary. We examine algorithms for this case for two reasons. First, an algorithm that can efficiently calculate a good policy from a model is run in an inner loop in subsequent algorithms. Second, many algorithms designed for more restrictive problem setups are based on algorithms for this case.

Value iteration is a method for finding an optimal policy by calculating the state value function. For the T-step reward criteria, the algorithm takes the following form:

T-step Value Iteration $(M = \langle S, A, P, R \rangle, T)$

1. Set $V^*(s, 0) \leftarrow 0$ for all $s \in S$
2. For $t = 1, \ldots, T$:
3. For all $s \in S$:

$$\begin{aligned} V^*(s,t) &\leftarrow \max_{a \in A} \left(\frac{1}{t} E[R(s,a)] + \frac{t-1}{t} \sum_{s'} P(s'|s,a) V^*(s', t-1) \right) \\ \pi^*(s,t) &\leftarrow \operatorname*{argmax}_{a \in A} \left(\frac{1}{t} E[R(s,a)] + \frac{t-1}{t} \sum_{s'} P(s'|s,a) V^*(s', t-1) \right) \end{aligned}$$

4. Return π^*.

It is straightforward to show that this algorithm runs in $O(|S|^2|A|T)$.

Though it is possible to use a similar algorithm to derive an optimal policy for the discounted reward criteria, a different algorithm is usually used. Policy Iteration finds the optimal policy by systematically improving an initial policy until no changes are made:

Policy Iteration $(M = \langle S, A, P, R \rangle, \gamma)$

1. Set $\pi^0(s, \gamma)$ to some initial decision
2. $i \leftarrow 0$
3. Repeat
4. For all $s \in S$:

$$\begin{aligned} &\text{Calculate} && V^i(s,\gamma) \\ &\pi^{i+1}(s) &\leftarrow & \operatorname*{argmax}_{a \in A} \left((1-\gamma) E[R(s,a)] + \gamma \sum_{s'} P(s'|s,a) V^i(s', \gamma) \right) \end{aligned}$$

   $i++$.
5. Until $\pi^i = \pi^{i-1}$
6. Return π^i.

In order to execute this algorithm, a method for finding the state value of policy π^i is required. This is usually calculated by solving the recursion relations described above as system of linear equations [Littman et al., 1995]. The runtime for each loop is dominated by this calculation, which involves inverting an $|S| \times |S|$ matrix. Though the algorithm is not known to terminate in polynomial time, the algorithm usually terminates after iterating through a small number of policies.

5 Generative model

A generative model allows the agent to sample experiences generated from the MDP. An experience is a 4-tuple (s, a, r, s'), where s is the sampled state, a is the decision made, r is a reward drawn from $R(s, a)$, and s' is the resulting state, drawn from $P(\cdot|s, a)$. We will assume the function **Sample**$(s, a) \rightarrow (s, a, r, s')$ is provided as a means to sample from the MDP.

Most algorithms for the generative model case use samples from the MDP to build an approximation of P and R, or an approximation of V^{pi} for a good policy π. Because we only have approximations of these functions, we can no longer guarantee that we have found the optimal policy after a finite number of samples.

If the MDP is drawn from a prior distribution known the the agent, the Bayesian concept of optimal decision making can be employed. This involves planning over a parametric representation of the distribution the MDP is drawn from. The strategy resulting from planning over the distribution of all MDPs results in a policy that leads to highest total expected reward. See [Duff and Barto, 1997] for a recent discussion of this framework. Also see [Dearden et al., 1999] for a practical approximation of this scheme.

If the agent does not have access to a prior on the problem, a more adversarial analysis will be necessary. PAC analysis provides such a framework. When the RL problem is analyzed in the this framework, the exploration/exploitation problem is implicitly solved. If the agent does not have enough samples to reliably construct a near-optimal policy, more exploration is needed. Once the pre-specified error and robustness criteria are met, the algorithm will halt and return a near-optimal policy with high probability.

In the following sections, we will examine PAC algorithms for three cases of the generative model. First, we examine the case where there is only one state. Second we show how algorithms for the one state case can be extended to find good policies in MDPs with multiple states. Last, we examine the online model, where the agent can only sample from the last state the agent experienced.

6 Exploration in the one-state case

The one-state MDP, also known as the N-armed bandit, is a classic domain in decision theory. For these problems, the transition function simply states that all decisions lead to a transition back to the same state. The decision making agent is faced with $n = |A|$ possible decision. Each decision results in a reward drawn from a probability distribution over $[0, 1]$. The agent's goal is to find a policy that is nearly as good as always making the most rewarding choice. In the bandit case, the T-step and discounted reward criteria are equivalent. The value of the optimal policy is simply the highest expected reward for any one choice.

Though the N-armed bandit problem may seem overly simple, the model captures many interesting problems. One prominent example of this is classifier selection. As-

sume that an agent is faced with a classification problem, and has n classifiers to choose from. The agent would like to choose a classifier with performance near the best of the group. In order to do this, an agent can sample the performance of each classifier using a validation set drawn from the input distribution of the classification problem. If the classifier classified the input correctly, a reward of 1 is received, otherwise the reward is 0. We would like a sampling algorithm that needs as small a validation set as possible to choose a classifier within ϵ of the best with confidence $(1-\delta)$. See [Lizotte et al., 2003] for recent work on treating classifier selection as a bandit problem.

Recent work has introduced and analyzed efficient PAC algorithms for the bandit problem [Even-Dar et al., 2002]. The authors begin by analyzing a very simple algorithm. The algorithm is paraphrased below, with a cleaner analysis provided.

NaivePACbandit (M, ϵ, δ)

1. For each decision a, call **Sample**(s, a) $f(\epsilon, \delta, n)$ times.
2. Output the decision that accumulated the most reward.

Now we must specify $f()$ such that the algorithm meets our requirements efficiently. Assume a^* is the optimal decision with expected reward r^*. We must bound the probability that the algorithm outputs some other decision a' with expected reward $r' < r^* - \epsilon$ to less than δ. Call the sum or rewards from the two decisions S^* and S', and the number of samples taken from each arm l. The analysis begins with the recognition that in order for a significantly suboptimal decision to be chosen, either the expected reward for a suboptimal decision was overestimated or the expect reward for the optimal decision was underestimated:

$$\begin{aligned} P[\exists a' | S' > S^* \text{ and } r^* > r' + \epsilon] &< P\big[\big(\exists a' \neq a^* | S' > (r' + \epsilon/2)l\big) \\ &\quad \text{OR } \big(S^* < (r^* - \epsilon/2)l\big)\big] \\ &< (n-1)P[S' > (r' + \epsilon/2)l] \\ &\quad + P[S^* < (r^* - \epsilon/2)l] \\ &\leq (n)\exp\big(-2l(\epsilon/2)^2\big) \end{aligned}$$

The second inequality is due to the union bound and the last inequality is given by the Hoeffinding bound. We now must find the value of l that bounds the failure probability to δ. A simple calculation shows $l = \frac{2}{\epsilon^2}\log\frac{n}{\delta} = f(\epsilon, \delta, n)$ is sufficient to meet our requirements. Thus the total number of samples **NaivePACbandit** makes is $O(\frac{n}{\epsilon^2}\log\frac{n}{\delta})$.

A more efficient PAC algorithm is presented in [Even-Dar et al., 2002]. The algorithm gains efficiency by sampling less promising decisions fewer times than decisions thought to be near optimal.

MedianElimination (ϵ, δ)

1. Set A_1, the set of all decisions currently considered, to A.
2. Set $\epsilon_1 \leftarrow \epsilon/4; \delta_1 \leftarrow \delta/2; l \leftarrow 1$.
3. For each decision a_i, call **Sample**(s, a_i) for $t_l = \frac{\log(3/\delta_l)}{(\epsilon_l/2)^2}$ times; Call the sum of rewards for this decision S_l^i.
4. Find the median of the sums; call it S_l'.
5. $A_{l+1} \leftarrow A_l - \{a_i : S_l^i < S_l'\}; \quad \epsilon_{l+1} \leftarrow \frac{3}{4}\epsilon_l; \quad \delta_{l+1} \leftarrow \delta_l/2; l++$.
6. If $|A_l| = 1$, output A_l; else go to step 3.

A more thorough modified analysis of this algorithm follows. In order to bound the probability of failure, we will bound the chance that for each round $l+1$, all decisions with expected reward within ϵ_l of the best decision the previous round have been eliminated. Call the best decision remaining in round l as r'_l.

We will break the analysis into two cases. First we calculate the probability that r'_l is underestimated and assume it results in failure:

$$\begin{aligned} P[r'_l - \epsilon_l > r'_{l+1}] &< P[S'_l < t_l(r'_l - \epsilon_l/2)] \\ &\leq \exp(-\frac{\log(3/\delta_l)}{(\epsilon_l/2)^2}(\epsilon_l/2)^2 \\ &< \frac{\delta_l}{3} \end{aligned}$$

Next we calculate the chance that r'_l is not significantly underestimated, but is still below the median. We will make use of the Markov inequality in this calculation:

$$\begin{aligned} P\big[|\{a^i : a^i \in V_l \text{ and } r^i < r'_l - \epsilon_l \text{ and } S^i_l > S'_l\}| > |V_l|/2\big] &\leq \frac{E[\text{set size}]}{|V_l|/2} \\ &\leq \frac{|V_l| \cdot \delta_l/3}{|V_l|/2} \\ &= \frac{2}{3}\delta_l \end{aligned}$$

By simply using the union bound, we can bound the chance of either failure occurring to δ_l. Now we must sum the chance of failure and accumulated errors over all rounds:

$$\begin{aligned} \sum_l \delta_l &= \sum_{l=1}^{\log n} \delta/2^l \\ &\leq \delta \\ \sum_l \epsilon_l &= \sum_{l=1}^{\log n} (\frac{3}{4})^l \epsilon/4 \\ &\leq (\frac{1}{1-3/4})\epsilon/4 \\ &= \epsilon \end{aligned}$$

Compared to the naive algorithm, **MedianElimination** takes asymptotically fewer samples to achieve the same PAC bound.

$$\begin{aligned} SAMPLES &= \sum_{l=1}^{\log n} \frac{|V_l| \log(3/\delta_l)}{(\epsilon_l/2)^2} \\ &= 4\sum_{l=1}^{\log n} n/2^{l-1} \frac{\log((3 \cdot 2^l)/\delta)}{(.75^{l-1}\epsilon/4)^2} \\ &= \frac{64n}{\epsilon^2} \sum_{l=1}^{\log n} \Big(\frac{8}{9}\Big)^{l-1} \big(\log(2^l) + \log(3) + \log(1/\delta)\big) \\ &= O\Big(\frac{n\log(1/\delta)}{\epsilon^2}\Big) \times \sum_{l=1}^{\log n} \Big(\frac{8}{9}\Big)^l (l) \\ &= O\Big(\frac{n\log(1/\delta)}{\epsilon^2}\Big) \end{aligned}$$

It was proven in [Mannor and Tsitsiklis, 2004] that this bound is tight. This is done by presenting a family of N-armed bandit problems with different sets of ϵ-optimal decisions. They prove that any algorithm requires $O\Big(\frac{n\log(1/\delta)}{\epsilon^2}\Big)$ to disambiguate the bandit problems in this family.

7 Exploration with a full generative model

Below we present two algorithms for learning a near-optimal policy when the agent can sample any state-decision pair at any time. One case where this situation arises is when the problem domain is simulated using a stochastic dynamic system [Ng and Jordan, 2000]. For these systems, explicitly calculating the distribution $P(\cdot|s,a)$ may be much more difficult than sampling from it. Also, the state space may be so large that operating with an exact representation of P is intractable.

7.1 Many bandits algorithm

As mentioned before, a bandit problem can be thought of as an MDP where there is only one state, and all decisions transition back to this state. Likewise, an MDP can be treated as a set of bandit problems, where the reward generated from a decision is the sum of the immediate reward, plus the reward from successive decisons. This relation can be used explicitly for optimizing T-step reward We assume that the agent has a PAC algorithm for solving N-armed bandit problems. The algorithm presented below is an adaptation of one presented in [Even-Dar et al., 2002] to the T-step objective.

MDPbandit(ϵ,δ)

1. For t from 1 to T:
2. For all states s:
3. Run a bandit algorithm $B(\epsilon',\delta')$ for the N-armed bandit on state s:
   (a) Each time B requests a sample of decison a, use **sample** to simulate taking a in s, and following π for $t-1$ steps thereafter. Return $r=\sum_{t'=0}^{t-1}\frac{r_t}{T}$ to B.
   (b) Set $\pi(s,t)$ to the decision B returns.

In order to guarantee that an agent starting in some state s_1 is expected to receive a T-step reward no more than ϵ less than optimal with probability $1-\delta$, we will set ϵ' to $\frac{\epsilon}{T}$ and δ' to $\frac{\delta}{T}$.

The agent will make T decisions, each of which is likely to produce an expected outcome that is within $\frac{\epsilon}{T}$ of the optimal expected outcome, given that future decisions are fixed by π. Thus, with high probability, the approximation error of the agent's decision at time t can be only ϵ/T more than it's error at time $t+1$. This bounds the total approximation error to ϵ.

The agent will be in a maximum of T states during it's interaction with the MDP. Each state could have a bad choice of ϵ-optimal decision with probability δ/T. By the union bound, the probability of being in a poorly estimated state can be at most δ.

The samples required by this algorithm is $O(TS)$ times the sample needed by the PAC algorithm. If we use **MedianElimination** with the modified ϵ and δ values, this brings the total samples requited to

$$O\Big(\frac{T^3SA\log(T/\delta)}{\epsilon^2}\Big).$$

A similar algorithm with a larger runtime is given in [Kakade, 2003]. The algorithm presented in that work has runtime equal **MDPbandit** using the naive bandit algorithm.

7.2 Sparse Sampling

If the size of S is prohibitively large, all algorithms mentioned so far would not be applicable. It is shown in [Kearns et al., 2002] that a finding a near-optimal decision for a single state can be found in time independent of $|S|$. Unfortunately, the tradeoff is a runtime exponential in T. The intuition behind this result can seen by examining the behavior of **MDPbandit** running in reverse. If the bandit learner samples each of $|A|$ decisions in state s_1 m times, at most $m|A|$ subsequent states will be observed. The policy executed in these subsequent states are also determined by an execution of the bandit learner. The Sparse sampling algorithm essentially builds this tree of possible trajectories through the MDP, and then chooses the decision that leads to the best set of possible future trajectories [Kearns et al., 2002]. The details of the algorithm and its analysis are rather complex, and will be omitted. If an agent uses the sparse sampling algorithm as it's policy, then it will have value within ϵ at the cost of

$$\Big(\frac{|A|T}{\epsilon}\Big)^{O(T\log T)}$$

samples for every decision made.

8 Exploration with an online model

In the online model, the agent can only sample from the state it is currently in. In other words, if **Sample**(s, a) is called, yielding experience (s, a, r, s'), only calls of **Sample**(s', a') for some a' are permitted for the next call. Of all the cases considered, this is the one that most resembles what an agent is likely to face when it learns to make decisions in the real world.

Under this situation, it is no longer possible to guarantee we will find a policy that is near optimal when started at a particular state. If the agent samples some decision early in exploration, it may have an irrevocable effect on the set of states that the agent can reach in the future. Our criteria for sufficient exploration will instead be that the agent has arrived in a state where it is likely a near-optimal T-step policy can be followed.

8.1 R-Max

The R-max algorithm [Brafman and Tennenholtz, 2002], is a direct descendent of E^3 [Kearns and Singh, 2002], the first algorithm proven to find a near optimal policy with high probability after a polynomial number of online samples. The differences between the algorithms are largely superficial, and are due to the fact that E^3 does not assume rewards and value functions are bounded in $[0, 1]$.

The R-max algorithm assumes that we have access to a planner that is able to take an MDP description and return an optimal policy for it. For this we can use either value iteration or policy iteration, depending on whether we wish to optimize for T-step or discounted reward. The algorithm will be described for T-step objective, but the modifications for discounted reward are trivial.

Under the original R-max algorithm, the agent either follows an optimal policy based on an approximate model of the known states, or find a way to reach a promising

unknown state quickly. Below we present a modified version of R-max that will halt when the exploitation strategy is likely to be ϵ-optimal.

The idea behind R-max is to divide the states of the MDP into known and unknown sets. States are known when enough samples have been taken in order to sufficiently approximate the local reward and transition functions for all decisions.

The algorithm will keep two approximate models of the domain. $\overline{M}$ is an "approximately optimistic" model. This model is initialized so that all decisions in unknown states yield maximum reward and transition back to themselves. Note that the unknown states have a state value of 1 for any policy. A "pessimistic" approximation $\underline{M}$ is also kept. It is initialized like the optimistic model, except all rewards are minimal. In this model, unknown states have value 0. Whenever R-max has sampled each decision in state s enough to build an accurate model of $P(\cdot|s,a)$ and $R(s,a)$, s is marked as known and the local estimates of P and R are incorporated into $\overline{M}$ and $\underline{M}$. The full algorithm is given below, followed by an analysis.

Halting R-max$(\epsilon, \delta, s_0 \in S)$

1. Mark all states as unknown. Set:

$$\begin{aligned} \hat{P}(s|s,a) &= 1 \text{ for all } s \\ \hat{P}(s'|s,a) &= 0 \text{ for all } s' \neq s \\ \overline{R}(s,a) &= 1 \text{ for all } s,a \\ \underline{R}(s,a) &= 0 \text{ for all } s,a \end{aligned}$$

2. Repeat for $t = 0, 1, 2 \ldots$
3. If current state s_t is unknown:
   (a) Make decision a attempted fewest times for this state. Call **Sample**(s_t, a).
   (b) If each decision was attempted $m(\epsilon, \delta)$ times in s_t, mark s_t known. For all a and s', set

$$\begin{aligned} \hat{P}(s'|s_t,a) &= \frac{\#\text{ samples of form } (s_t, a, \cdot, s')}{\#\text{ samples of form } (s_t, a, \cdot, \cdot)} \\ \underline{R}(s_t,a) = \overline{R}(s_t,a) &= \frac{\sum_{(s_t,a,r,\cdot)} r}{\#\text{ samples of form } (s_t, a, \cdot, \cdot)} \end{aligned}$$

4. If current state is known:
   (a) Calculate the optimal policy π^r for $\overline{M} = \langle S, A, \hat{T}, \overline{R} \rangle$ and the policy's value at state s_t. Call it $\overline{V}(s_t)$.
   (b) Calculate the state value at s_t for π^r in $\underline{M} = \langle S, A, \hat{T}, \underline{R} \rangle$. Call it $\underline{V}(s_t)$.
   (c) If $\overline{V}(s_t) - \underline{V}(s_t) < \epsilon/2$ halt and return π^r;
   (d) Else call **Sample**$(s_t, \pi(s_t, T - t \mod T))$.

8.1.1 Analysis

We will divide the analysis in two parts. First we examine the behavior of π when the approximate models are correct. Define V^π as the value of the policy π calculated by R-max on the true MDP M. For an arbitrary policy π, if P and R are modeled perfectly for known states, the following properties hold:

- $\overline{V}^\pi(s,T) \geq V^\pi(s,T)$
- $V^\pi(s,T) \geq \overline{V}^\pi(s,T) - Pr(\pi \text{ transitions to unknown state in} \leq T \text{ steps})$

The first statement is true because the models only differ in the unknown states, and the unknown states have maximum value.

The second statement is proved below

$$\begin{aligned}
\overline{V}^{\pi}(s,T) - V^{\pi}(s,T) &= E[\sum_{t=1}^{T} \frac{\overline{R}(s_t,a_t)}{T}] \\
&\quad -E[\sum_{t=1}^{T} \frac{R(s_t,a_t)}{T}] \\
&= \frac{1}{T}\sum_{t=1}^{T} \Big(Pr(s_t \in \text{Unknown } |\overline{M}) \times 1 \\
&\quad +Pr(s_t \in \text{Known } |\overline{M}) E[R(s_t,a_t)] \\
&\quad -Pr(s_t \in \text{Unknown } |M) E[R(s_t,a_t)] \\
&\quad -Pr(s_t \in \text{Known } |M) E[R(s_t,a_t)] \Big) \\
&\leq \frac{1}{T}\sum_{t=1}^{T} \Big(Pr(s_t \in \text{Unknown } |\overline{M}) E[1 - R(s_t,a_t)] \\
&\quad +Pr(s_t \in \text{Known } |\overline{M}) E[R(s_t,a_t) - R(s_t,a_t)] \Big) \\
&\leq \frac{1}{T}\sum_{t=1}^{T} Pr(s_t \in \text{Unknown } |\overline{M}) \\
&\leq Pr(\exists s_i \in \text{ Unknown})
\end{aligned}$$

The first equation is given by our definition of the value function. The second equation breaks the expectation between the known and unknown states. Next we force the value of π in the original MDP to be evaluated based on the frequency states are visited in the optimistic model. We know that the inequality relation holds, because there is a difference in reward in the unknown states, and the unknown states are visited more often in $\overline{M}$ due to the our transition function. Next we simply eliminate the subtraction of reward received when π is in an unknown state in model M. The last inequality holds because the probability that s_t is an unknown state can only increase as t increases. Averaging the probability that an unknown was entered over all t underestimates the probability that the even occurred.

Now we examine the properties of $\underline{V}^{\pi}(s,T)$. If P and R are modeled perfectly, for any policy π:

- $V^{\pi}(s,T) \geq \underline{V}^{\pi}(s,T)$
- $\underline{V}^{\pi}(s,T) + Pr(\pi \text{ transitions to unknown state in} \leq T \text{ steps}) \geq V^{\pi}(s,T)$

The proofs of these statements are similar to the proofs for $\overline{V}^{\pi}$ given above.

By combining the properties stated above, we get the following result:

$$2 \times Pr(\pi \text{ transitions to unknown state in} \leq T \text{ steps}) \geq \overline{V}^{\pi}(s_t) - \underline{V}^{\pi}(s_t)$$

for any policy π. This gives us a lower bound on the chance that a successful exploration will occur in T steps. Because there are only $|S|$ states, and $|A|$ decisions per state, only $|S||A|m(\epsilon,\delta)$ explorations can occur before before every state is known. We can treat each attempt at exploration made by R-max as a random variable with success probability at least ϵ. The expected number of attempts before $|S||A|m(\epsilon,\delta)$ successes

is $\frac{|S||A|m(\epsilon,\delta)}{\epsilon}$. We can use the Hoeffingding bound to show that we can bound the probability of needing more than $O(\frac{|S||A|m(\epsilon,\delta)}{\epsilon})$ to $(1-\delta)$.

The prior analysis assumed that the known states were modeled correctly by R-max. We now set $m(\epsilon,\delta)$ such that the above analysis holds with high probability. We desire an approximation of P that is within ϵ of the true distribution over next states. An approximation $\hat{P}(\cdot|s,a)$, $\hat{R}(s,a)$ is an ϵ-approximation if

$$\sum_{s'} |\hat{P}(s'|s,a) - P(s'|s,a)| + |\hat{R}(s,a) - E[R(s,a)]| \leq \epsilon.$$

If all local transition models are ϵ-approximated, the following property holds for all policies:

$$|\hat{V}^\pi(s,T) - V^\pi(s,T)| \leq \epsilon T$$

This is proved as follows

$$\begin{aligned}
|\hat{V}^\pi(s,T) - V^\pi(s,T)| &= \Big|\Big(\frac{\hat{R}(s,a)}{T} + \frac{T-1}{T}\sum_{s'}\hat{P}(s'|s,a)\hat{V}^\pi(s',T-1)\Big) \\
&\quad -\Big(\frac{E[R(s,a)]}{T} + \frac{T-1}{T}\sum_{s'}P(s'|s,a)V^\pi(s',T-1)\Big)\Big| \\
&\leq \Big|\frac{\hat{R}(s,a)}{T} - \frac{E[R(s,a)]}{T}\Big| \\
&\quad +\frac{T-1}{T}\sum_{s'}\Big|\hat{P}(s'|s,a)\hat{V}^\pi(s,T-1) - P(s'|s,a)V^\pi(s,T-1)\Big| \\
&\leq \Big|\frac{\hat{R}(s,a)}{T} - \frac{E[R(s,a)]}{T}\Big| \\
&\quad +\frac{T-1}{T}\sum_{s'}\Big|\hat{P}(s'|s,a)V^\pi(s,T-1) - P(s'|s,a)V^\pi(s,T-1)\Big| \\
&\quad +\frac{T-1}{T}\sum_{s'}\Big|P(s'|s,a)\hat{V}^\pi(s,T-1) - P(s'|s,a)V^\pi(s,T-1)\Big| \\
&= \Big|\frac{\hat{R}(s,a)}{T} - \frac{E[R(s,a)]}{T}\Big| \\
&\quad +\frac{T-1}{T}\sum_{s'}\Big|\hat{P}(s'|s,a) - P(s'|s,a)\Big|V^\pi(s,T-1) \\
&\quad +\frac{T-1}{T}\sum_{s'}P(s'|s,a)\Big|\hat{V}^\pi(s,T-1) - V^\pi(s,T-1)\Big| \\
&\leq \epsilon + E\Big[\Big|\hat{V}^\pi(s,T-1) - V^\pi(s,T-1)\Big|\Big]
\end{aligned}$$

We solve the recursion relation to finish the proof. Above we used some properties of absolute differences, as well as the fact that V^π is always less than 1.

Now we must find a sample size $m(\epsilon,\delta)$ that will give us an ϵ-approximation in one state with probability $(1-\delta)$. In the end, we are interested in bounding the deviance in value of the policy found using an ϵ-approximation, so we will set $\epsilon \leftarrow \epsilon/T$. A naive strategy for calculating m would bound the deviation of each $P(s'|s,a)$ entry to $\epsilon/(|S|+1)$, thus guaranteeing that the expected deviation from the true model is no more than ϵ. Such an analysis would suggest sampling $m(\epsilon,\delta) = O((\frac{|S|T}{\epsilon})^2 \log\frac{|S|A}{\delta})$. A more nuanced analysis provided in [Kakade, 2003] reduces the samples required by a factor of $|S|$. The details of this analysis are rather involved and will be omitted.

For the final algorithm, we must tune all of the components' ϵ and δ values so that the total chance of failure to return an ϵ-optimal policy is less than δ. By applying the union bound to all possible failures, we wind up with a PAC algorithm that requires $O(\frac{|S|^2|A|T^3}{\epsilon^3}(\log \frac{|S||A|}{\delta})^2)$ online samples.

8.2 Extensions of R-Max

We have usually assumed a model of the environment where $|S|$ is finite and manageable, and the transition function P is represented a full multinomial over $|S|$ possible next states. In most real-world problems, this assumption is unreasonable. Two extensions of R-max have been developed for situations where a special class of models is applicable.

In [Kearns and Koller, 1999] an efficient algorithm for learning in factored MDPs is proposed. Factored MDPs assume that states are encoded as a n-dimensional vector. The transition function P is a graphical model that takes the state vector and decision as input, and calculates a probability distribution on each parameter of the subsequent state vector. Factored MDPs allow for a compact representation of the transition function, even when the n-dimensional state-space is very large. Unfortunately, the compactness of P does not in general lead to compact optimal policies, or make finding these policies easier [Allender et al., 2002]. An approximate algorithm such as sparse sampling could be used, however.

A metric model of P is analyzed in [Kakade et al., 2003]. This work assumes there is some metric defined on states that measures the similarity of their transition functions. The metric has the property that if the agent has m sample experiences that are all within some metric distance α of s, then an α-approximate model of P and R can be built by using the empirical outcomes of the m samples. This modeling assumption is well suited for continuous state-spaces, where the effects of decisions change smoothly over the space. This work introduces the concept of the effective size of a state-space, which is equal to the maximum number of states that are at least α distant from each other. The bound on the number of exploration steps required for these problems is defined in terms of the effect size of the state-space, not the (possibly infinite) number of distinct states. As is the case for factored MDPs, no algorithm is known to compute optimal policies in metric spaces. Sparse sampling could be used if no alternative is available.

9 Discussion

We have examined the number of samples required for an agent to learn an ϵ-optimal policy with probability $(1 - \delta)$. When a full generative model of the domain is available, $O(\frac{T^3SA}{\epsilon^2}\log(T/\delta))$ samples are required by **PAC-MDP**. When an online model is used, however, $O(\frac{|S|^2|A|T^3}{\epsilon^3}(\log \frac{|S||A|}{\delta})^2)$ are needed by **Halting R-max**. To my knowledge, these are the best runtime bounds known for the generative and online cases. Interestingly, the tightest lower bound on samples needed for both of these problems is $\Omega(\frac{|S||A|T}{\epsilon}\log\frac{1}{\delta})$ [Kakade, 2003]. While the upper and lower bounds for the generative model case are fairly close in terms of S and A, there is a gap of $O(|S|)$ in the upper and lower bounds for the online case.

At a high level, the difference in runtime between the generative algorithm and the online algorithm is due to the fact that the generative model only stores the policy it is learning, while the online algorithm is storing an entire MDP. It is reasonable to assume that any algorithm which approximates the $O(|S|^2|A|)$ parameters in M will require $O(|S|^2|A|)$ samples.

It is an open question, however, whether an approximation of the model is needed at all. Our generative algorithm gets around approximating M by using sampled trajectories to approximate $V^\pi(s)$ directly.

The online algorithm does not have this option, because it costly to return to a trajectory's starting state to sample another. Instead, R-max approximates the value function by solving its recurrence relation on an approximate M.

9.1 From PAC to Practical

One possibility for improving the performance of online reinforcement learning algorithms is to have the algorithm explore more promising states and decisions instead of ones that are almost surely not visited by good policies. The value functions $\overline{V}$ and $\underline{V}$ calculated by R-max can be though of as crude confidence intervals of the likely value of the optimal policy. Can the performance of R-max be improved if $\overline{V}$ and $\underline{V}$ are replaced with tighter confidence intervals? This question is addressed in the very recent work of [Strehl and Littman, 2004]. They compared an empirically optimized variant of R-max to a new algorithm that explicitly uses confidence intervals to make the next decision. The Model-based Interval Estimator algorithm uses a policy that is optimal for the upper bounds on expected value. As these decisions are sampled more frequently, the bound on their expected value becomes tighter, which encourages sampling other potentially optimal decisions. This algorithm outperformed R-max on several synthetic domains. A PAC analysis of this algorithm has not been performed, however. It would be interesting to see if this algorithm runs asymptoticly faster than R-max, or if it just performs better in practice.

9.2 Robust exploitation

An interesting related problem in reinforcement learning is how to make the best use of a fixed number of samples that are provided to the learner. The agent must make the best use of the information it has available in order to form a good exploitation policy. This problem arises when databases of domain experience are "mined" for situations where altering a decision would likely yield higher total reward. A case-study of such a situation is addressed in [Abe et al., 2002].

When an arbitrary sample is collected, it is foolish to assume that there is enough information to find an ϵ-optimal policy. Two alternate criteria are proposed. One criteria is to see if a policy can be found that improves upon the policy used to collect the sample. A good deal of ad-hoc work has been done on this subject, but some of the first theoretical results on this problem have been derived only recently [Langford and Zadrozny, 2003].

The second criteria is to find a policy and value function such that with high probability, the policy achieves at least this value. This is problem is known as robust reinforcement learning. The usual approach to this problem is to use the samples to find a set of domain models that will generate the observed sample with probability above some threshold. The agent then performs a game theoretic computation to determine the model in the set where the agent's optimal policy performs worst. As in the exploration problem, confidence intervals have been used to define a set of likely models [Nilim and Ghaoui, 2003]. As of yet, there is little theoretical analysis of such methods.

A compelling direction for future research on exploration and exploitation would be to develop algorithms that unify the ideas from recent confidence interval algorithms into a system that can seamlessly switch between exploration and exploitation. The analysis

provided in robust reinforcement learning could provide better conditions for when exploration is no longer beneficial. Also, the exploration component of the algorithm would be a very efficient active sampler for improving the very thing the exploitation algorithm must optimize: the confidence intervals.

References

[Abe et al., 2002] Abe, N., Pednault, E., Wang, H., Zadrozny, B., Fan, W., and Apte, C. (2002). Empirical comparison of various reinforcement learning strategies for sequential targeted marketing. In *Proceedings of the IEEE International Conference on Data Mining*.

[Allender et al., 2002] Allender, E., Arora, S., Kearns, M., Moore, C., and Russell, A. (2002). A note on the representational incompatibility of function approximation and factored dynamics. In *Advances in Neural Information Processing Systems*.

[Bagnell et al., 2003] Bagnell, D., Kakade, S., Ng, A., and Schneider, J. (2003). Policy search by dynamic programming. In *Advances in Neural Information Processing Systems*.

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. T. (1996). *Neuro-dynamic Programming*. Athena Scientific.

[Brafman and Tennenholtz, 2002] Brafman, R. and Tennenholtz, M. (2002). R-max: A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3.

[Dearden et al., 1999] Dearden, R., Friedman, N., and Andre, D. (1999). Model-based bayesian exploration. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

[Duff and Barto, 1997] Duff, M. O. and Barto, A. G. (1997). Local bandit approximation for optimal learning problems. In Mozer, M. C., Jordan, M. I., and Petsche, T., (Eds.), *Advances in Neural Information Processing Systems*, volume 9, page 1019. The MIT Press.

[Even-Dar et al., 2002] Even-Dar, E., Mannor, S., and Mansour, Y. (2002). Pac bounds for multi-armed bandit and markov decision processes. In *Proceedings of the Conference on Learning Theory*, pages 255–270.

[Fern et al., 2003] Fern, A., Yoon, S., and Givan, R. (2003). Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing Systems*.

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

[Kakade et al., 2003] Kakade, S., Kearns, M., and Langford, J. (2003). Exploration in metric state spaces. In *Proceedings of the International Conference on Machine Learning*.

[Kearns and Koller, 1999] Kearns, M. and Koller, D. (1999). Efficient reinforcement learning in factored mdps. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Kearns et al., 2002] Kearns, M., Mansour, Y., and Ng, A. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49.

[Kearns and Singh, 2002] Kearns, M. and Singh, S. (2002). Near-optimal reinforcement leaning in polynomial time. *Machine Learning*, 49.

[Kearns and Vazirani, 1994] Kearns, M. and Vazirani, U. (1994). *An introduction to learning Theory*. The MIT Press.

[Langford and Zadrozny, 2003] Langford, J. and Zadrozny, B. (2003). Reducing t-step reinforcement learning to classification. In *submitted*.

[Langford et al., 2002] Langford, J., Zinkevich, M., and Kakade, S. (2002). Competitive analysis of the explore/exploit tradeoff. In *Proceedings of the International Conference on Machine Learning*.

[Littman et al., 1995] Littman, M., Dean, L., and Kaebling, L. (1995). On the complexity of solving markov decision problems. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*.

[Littman et al., 2001] Littman, M., Sutton, R., and Singh, S. (2001). Predictive representations of state. In *Advances in Neural Information Processing Systems*.

[Lizotte et al., 2003] Lizotte, D., Madani, O., and Greiner, R. (2003). Budgeted learning of naive-bayes classifiers. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

[Mannor and Tsitsiklis, 2004] Mannor, S. and Tsitsiklis, J. (2004). The sample complexity of exploration in the multi-armed bandit problem. *submitted*.

[Ng and Jordan, 2000] Ng, A. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

[Nilim and Ghaoui, 2003] Nilim, A. and Ghaoui, L. (2003). Robust markov decision problems with uncertain transition matrices. In *Advances in Neural Information Processing Systems*.

[Puterman, 1994] Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons.

[Strehl and Littman, 2004] Strehl, A. and Littman, M. (2004). Exploration via model-based interval estimation. In *submitted*.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

[Thrun, 1992] Thrun, S. B. (1992). The role of exploration in learning control with neural networks. In *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold.