Auto-exploratory Average Reward Reinforcement Learning

DoKyeong Ok and **Prasad Tadepalli**
Computer Science Department
Oregon State University
Corvallis,Oregon 97331-3202
{okd,tadepalli}@research.cs.orst.edu

Abstract

We introduce a model-based average reward Reinforcement Learning method called H-learning and compare it with its discounted counterpart, Adaptive Real-Time Dynamic Programming, in a simulated robot scheduling task. We also introduce an extension to H-learning, which automatically explores the unexplored parts of the state space, while always choosing greedy actions with respect to the current value function. We show that this "Auto-exploratory H-learning" performs better than the original H-learning under previously studied exploration methods such as random, recency-based, or counter-based exploration.

Introduction

Reinforcement Learning (RL) is the study of learning agents that improve their performance at some task by receiving rewards and punishments from the environment. Most approaches to reinforcement learning, including Q-learning (Watkins and Dayan 92) and Adaptive Real-Time Dynamic Programming (ARTDP) (Barto, Bradtke, & Singh 95), optimize the total *discounted* reward the learner receives. In other words, a reward which is received after one time step is considered equivalent to a fraction of the same reward received immediately. One advantage of discounting is that it yields a finite total reward even for an infinite sequence of actions and rewards. While mathematically convenient, many real world domains to which we would like to apply RL do not have a natural interpretation or need for discounting. The natural criterion to optimize in such domains is the average reward received per time step.

Discounting encourages the learner to sacrifice long-term benefits for short-term gains, since the impact of an action choice on long-term reward decreases exponentially with time. Hence, using discounted optimization when average reward optimization is what is required *could* lead to suboptimal policies. Nevertheless, it can be argued that it is appropriate to optimize discounted total reward if that also nearly optimizes the average reward. In fact, many researchers have successfully used discounted learning to optimize average reward per step (Lin 92; Mahadevan & Connell 92). This raises the question whether and when discounted RL methods are appropriate to use to optimize the average reward.

In this paper, we describe an Average reward RL (ARL) method called H-learning, which is an undiscounted version of Adaptive Real-Time Dynamic Programming (ARTDP) (Barto, Bradtke, & Singh 95). Unlike Schwartz's R-learning (Schwartz 93) and Singh's ARL algorithms (Singh 94), it is model-based, in that it learns and uses explicit action models. We compare H-learning with its discounted counterpart ARTDP to optimize the average reward in the task of scheduling a simulated Automatic Guided Vehicle (AGV), a material handling robot used in manufacturing. Our results show that H-learning is competitive with ARTDP when the short-term (discounted with strong discounting) optimal policy also optimizes the average reward. When short-term and long-term n this optimal policies are different, ARTDP either fails to converge to the optimal average reward policy or converges too slowly if discounting is weak.

Like most other RL methods, H-learning needs exploration to find a globally optimal policy. A number of exploration strategies have been studied in RL, including occasionally executing random actions, and preferring states which are least visited (counter-based) or actions least recently executed (recency-based) (Thrun 94). We introduce a version of H-learning which has the property of automatically exploring the unexplored parts of the state space while always taking a greedy action with respect to the current value function. We show that this "Auto-exploratory H-learning" outperforms the previous version of H-learning under three different exploration strategies, including counter-based and recency-based methods.

The rest of the paper is organized as follows: Section 2 derives H-learning and compares it with ARTDP. Section 3 introduces Auto-exploratory H-learning, and compares it with a variety of exploration schemes. Section 4 is a discussion of related work and future research issues, and Section 5 is a summary.

Average Reward Reinforcement Learning

Markov Decision Processes (MDP) are described by a set of n discrete states S and a set of actions A available to an agent. The set of actions which are applicable in a state i are denoted by $U(i)$ and are called *admissible*. The Markovian assumption means that an action u in a given state $i \in S$ results in state j with some fixed probability $P_{i,j}(u)$. There is a finite immediate reward $r_i(u)$ for executing an action u in state i resulting in state j. Time is treated as a sequence of discrete steps $t = 0, 1, 2, \ldots$. A *policy* μ is a mapping from states to actions, such that $\mu(i) \in U(i)$. We only consider policies which do not change with time, which are called "stationary policies."

Let a controller using a policy μ take the agent through states $s_0, \ldots, s_t$ in time 0 thru t, with some probability, accumulating a total reward $r^\mu(s_0, t) = \sum_{k=0}^{t-1} r_{s_k}(\mu(s_k))$. The expected total reward, $E(r^\mu(s_0, t))$, is a good candidate to optimize; but if the controller has infinite horizon, i.e., as t tends to ∞, it can be unbounded. The discounted RL methods make this sum finite by multiplying each successive reward by an exponentially decaying discount factor γ. In other words, they optimize the expected discounted total reward $\lim_{t\to\infty} E(\sum_{k=0}^{t-1} \gamma^k r_{s_k}(\mu(s_k)))$, where $0 < \gamma < 1$.

Discounting, however, tends to sacrifice bigger long-term rewards in favor of smaller short-term rewards, which is undesirable in many cases. A more natural criterion is to optimize the average expected reward per step over time t as $t \to \infty$. For a given starting state s_0, and policy μ, this is denoted by $\rho^\mu(s_0)$ and is defined as:

$$\rho^\mu(s_0) = \lim_{t\to\infty} \frac{1}{t} E(r^\mu(s_0, t))$$

We say that two states *communicate* under a policy if there is a positive probability of reaching each state from the other using that policy. A *recurrent* set of states is a closed set of states that communicate with each other, i.e., they do not communicate with states not in that set. Non-recurrent set of states are called *transient*. An MDP is *ergodic* if its states form a single recurrent set under each stationary policy. It is a *unichain* if every stationary policy gives rise to a single recurrent set of states and possibly some transient states.

For unichain MDPs the expected long-term average reward per time step for any policy μ is independent of the starting state s_0. We call it the "gain" of the policy μ, denoted by $\rho(\mu)$, and consider the problem of finding a "gain-optimal policy," μ^*, that maximizes $\rho(\mu)$ (Puterman 94).

Derivation of H-learning

Even though the gain of a policy, $\rho(\mu)$, is independent of the starting state, the total expected reward in time t may not be. The total reward for a starting state s in time t for a policy μ can be conveniently denoted by $\rho(\mu)t + \epsilon_t(s)$. Although $\lim_{t\to\infty} \epsilon_t(s)$ may not exist for periodic MDPs, the Cesaro-limit of $\epsilon_t(s)$, defined as $\lim_{l\to\infty} \frac{1}{l} \sum_{t=1}^{l} \epsilon_t(s)$, always exists, and is called the *bias* of state s, denoted by $h(s)$ (Bertsekas 95). Intuitively, $h(i) - h(j)$ is the average relative advantage in long-term total reward for starting in state i as opposed to state j.

Theorem 1 *For unichain MDPs, there exist a scalar ρ and a real-valued function h over S that satisfy the recurrence relation*

$$\forall i \in S, h(i) = \max_{u\in U(i)} r_i(u) + \{\sum_{j=1}^{n} p_{i,j}(u)h(j)\} - \rho \quad (1)$$

For any solution of (1), μ^* attains the above maximum for each state i, and ρ is its gain.

Notice that any one solution to Equation (1) yields an infinite number of solutions by adding the same constant to all h values. Setting the h value of an arbitrary recurrent "reference" state to 0, guarantees a unique solution for unichain MDPs. In White's relative value iteration method, the resulting equations are solved by synchronous successive approximation (Bertsekas 95). Unfortunately, the asynchronous version of this algorithm does not always converge, as was shown by Tsitsiklis, and cannot be the basis of an ARL algorithm(Bertsekas 82). Hence, instead of using Equation (1) to solve for ρ, H-learning estimates it from on-line rewards (see Figure 1).

The agent executes the algorithm in Figure 1 in each step, where i is the current state, and $N(i, u)$ denotes the number of times u is executed in i, out of which $T(i, u, j)$ times it resulted in state j. Our implementation explicitly stores the current greedy policy in the array *GreedyActions*. This gives a small improvement in performance in some domains because the policy is more stable than the value function. Before starting, the algorithm initializes α to 1, and all other variables to 0. *GreedyActions* are initialized to the set of admissible actions.

H-learning can be seen as a cross between R-learning, which is model-free and undiscounted, and Adaptive RTDP (ARTDP), which is model-based and discounted. Like ARTDP, it estimates the probabilities $p_{i,j}(a)$ and rewards $r_i(a)$ by straightforward frequency counting, and employs the "certainty equivalence principle" by using the current estimates as the true values while updating the h values using Equation (1).

As in most RL methods, while using H-learning, the agent makes some exploratory moves – moves that do not necessarily maximize the right hand side of Equation (1) and are intended to ensure that every state in S is visited infinitely often during training. These moves make the estimation of ρ slightly complicated. Simply averaging immediate rewards over non-exploratory moves would not do, because the exploratory moves could make the system visit states that it never visits

1. Take an exploratory action or a greedy action in the current state i. Let a be the action taken, k be the resulting state, and r_{imm} be the immediate reward received.
2. $N(i,a) \leftarrow N(i,a)+1$; $T(i,a,k) \leftarrow T(i,a,k)+1$
3. $p_{i,k}(a) \leftarrow T(i,a,k)/N(i,a)$
4. $r_i(a) \leftarrow r_i(a)+(r_{imm}-r_i(a))/N(i,a)$
5. $GreedyActions(i) \leftarrow$ All actions $u \in U(i)$ that maximize $\{r_i(u)+\sum_{j=1}^{n} p_{i,j}(u)h(j)\}$
6. If $a \in GreedyActions(i)$, then
   (a) $\rho \leftarrow (1-\alpha)\rho+\alpha(r_i(u)-h(i)+h(k))$
   (b) $\alpha \leftarrow \frac{\alpha}{\alpha+1}$
7. $h(i) \leftarrow \max_{u \in U(i)}\{r_i(u)+\sum_{j=1}^{n} p_{i,j}(u)h(j)\}-\rho$
8. $i \leftarrow k$

Figure 1: The H-learning Algorithm

if it were following the greedy policy and accumulate rewards received by optimal actions in these states. Instead, we use R-learning's method of estimating the average reward (Schwartz 93). From Equation (1), for any "greedy" action u in any state i which maximizes the right hand side, $\rho = r_i(u)-h(i)+\sum_{j=1}^{n} p_{i,j}(u)h(j)$. Hence, the current ρ can be estimated by cumulatively averaging $r_i(u)-h(i)+h(j)$, whenever a greedy action u is executed in state i resulting in state j.

H-learning is very similar to Jalali and Ferguson's Algorithm B, which is proved to converge to the gain-optimal policy for ergodic MDPs (Jalali and Ferguson 89). Ergodicity assumption allows them to ignore the issue of exploration, which is otherwise crucial for convergence to the optimal policy. Indeed, the role of exploration in H-learning is to transform the original MDP into an ergodic one by making sure that every state is visited infinitely often. Secondly, to make the h values bounded, Algorithm B arbitrarily chooses a reference state and permanently sets its h value to 0. We found that this change slows down H-learning in many cases. In spite of these two differences, we believe that the convergence proof of Algorithm B can be extended to H-learning and R-learning as well.

Experimental Results on H-learning

In this section, we assume that we are in a domain where gain-optimality is the desired criterion, and experimentally study the question whether and when it may be appropriate to use discounting.

Our experimental results are based on comparing H-learning with its discounted counterpart ARTDP in a simplified task of scheduling simulated Automatic Guided Vehicles (AGVs). AGVs are mobile robots used in modern manufacturing plants to transport materials from one location to another. In our "Delivery domain" shown in Figure 2, there are two job generators on the left, one AGV, and two destination conveyor belts on the right. Each job generator produces jobs and puts them on its queue as soon as it is empty. The AGV loads and carries a single job at a time to its destination conveyor belt.

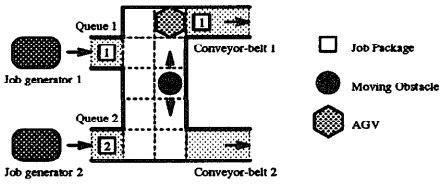

Figure 2: The Delivery domain

Each job generator can generate either of two types of jobs (when its queue is empty). Job 2, destined to belt 2, has a reward of 1 unit, while job 1, destined to belt 1, receives a reward K when delivered. The probability of generating job 1 is p for generator 1, and q for generator 2.

The AGV moves on two lanes of 5 positions each, and can take one of six actions at a time: do-nothing, load, move-up, move-down, change-lane, and unload. load and unload can only be executed from the positions next to the source and the destination of jobs respectively. An obstacle randomly moves up and down in the right lane. There is a penalty of -5 for collisions with the obstacle.

There are a total of 540 different states in this domain specified by the job numbers in the generator queues and the AGV, and the locations of the AGV and the obstacle. The goal is to maximize the average reward received per unit time.

We now present the results of comparing H-learning with ARTDP in the Delivery domain. p is set to 0.5, and q is set to 0.0. In other words, generator 1 produces both types of jobs with equal probability, while generator 2 always produces type 2 jobs. We compare the results of setting the reward ratio K to 1 and to 5 (Figure 3.) The results shown are averages of 30 trials. For exploration, with 10% probability, we executed a randomly chosen admissible action.

When $K = 1$, since both jobs have the same reward, the gain-optimal policy is to always serve the generator 2 which produces only type 2 jobs. Since the destination of these jobs is closer to their generator than type 1 jobs, it is also a discounted optimal policy. We call this type of domains "short-range domains" where the discounted optimal policy for a small value of γ coincides with the gain-optimal policy. In this case, the discounted method, ARTDP, converges to the optimal policy slightly faster than H-learning, although the difference is negligible.

When $K = 5$, the gain-optimal policy conflicts with the discounted optimal policy when $\gamma = 0.9$. Whenever the AGV is close to belt 2, ARTDP sees a short-

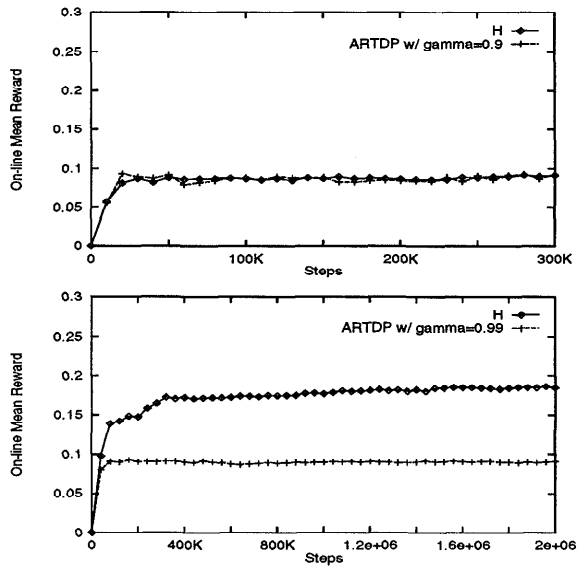

Figure 3: Average rewards per step for H-learning and ARTDP in the Delivery domain with p=0.5, q=0.0 for K=1(above) and K=5(below). Each point is the mean of 30 trials over the last 10K steps for K=1 and over the last 40K steps for K=5.

term opportunity in serving generator 2 and does not return to generator 1, thus failing to transport high reward jobs. Hence it cannot find the optimal average reward policy when $\gamma = 0.9$. To overcome this difficulty, γ is set to 0.99. Even so, it could not find the optimal policy in 2 million steps. This is because high values of γ reduce the effect of discounting and make the temporally far off rewards relevant for optimal action selection. Since it takes a long time to propagate these rewards back to the initial steps, it takes a long time for the discounted methods to converge to the true optimum. Meanwhile the short-term rewards still dominate in selecting the action. Thus, as we can infer from Figure 3, in this "long-range" domain, ARTDP served generator 2 exclusively in all the trials, getting a gain less than 0.1, while H-learning was able to find a policy of gain higher than 0.18.

We found that counter-based exploration improves the performances of both ARTDP and H. While ARTDP is still worse than H, the difference between them is smaller than with random exploration. We conclude that in long-range domains where discounted optimal policy conflicts with the gain-optimal policy, discounted methods such as ARTDP and Q-learning either take too long to converge or, if γ is too low, converge to a sub-optimal policy. When there is no such conflict, H-learning is competitive with the discounted methods. In more exhaustive experiments with 75 different parameter settings for p, q and K, it was found that H-learning always converges to the gain-optimal policy, and does so in fewer steps in all but 16 short-range cases, where ARTDP is slightly faster (Tadepalli & Ok 94). We also found similar differences between Q-learning and R-learning. Our results are consistent with those of Mahadevan who compared Q-learning and R-learning in a robot simulator domain and a maze domain and found that R-learning can be tuned to perform better (Mahadevan 96a).

Auto-exploratory H-learning

Recall that H-learning needs exploratory actions to ensure that every state is visited infinitely often during training. Unfortunately, actions executed exclusively for exploratory purpose could lead to decreased average reward, because they do not fully exploit the agent's currently known best policy.

In this section, we will describe a version of H-learning called Auto-exploratory H-learning (AH), which avoids the above problem by automatically exploring the promising parts of the state space while always executing current greedy actions. Our approach is similar to Kaelbling's Interval Estimation (IE) algorithm, and Koenig and Simmons's method of representing the reward functions using action-penalty scheme (Kaelbling 90; Koenig & Simmons 96).

We are primarily interested in non-ergodic MDPs here because ergodic MDPs do not need exploration. Unfortunately, the gain of a stationary policy for a multichain (non-unichain) MDP depends on the initial state (Puterman 94). Hence we consider some restricted classes of MDPs. An MDP is *communicating* if for every pair of states i, j, there is a stationary policy under which they communicate. For example, our Delivery domain is communicating. A *weakly communicating* MDP also allows a set of states which are transient under every stationary policy in addition (Puterman 94). Although the gain of a stationary policy for a weakly communicating MDP also depends on the initial state, the gain of an optimal policy does not. AH-learning exploits this fact, and works by using ρ as an upper bound on the optimal gain. It does this by initializing ρ to a high value and by slowly reducing it to the gain of the optimal policy. AH is applicable to find gain-optimal policies for weakly communicating MDPs, a strict superset of unichains.

There are two reasons why H-learning needs exploration: to learn accurate action and reward models, and to learn correct h values. Inadequate exploration could adversely affect the accuracy of either of these, making the system converge to a suboptimal policy.

The key observation in the design of Auto-exploratory H-learning (AH) is that the current value of ρ affects how the h values are updated for the states in the current greedy policy. Let μ be the current sub-optimal greedy policy, and $\rho(\mu)$ be its gain. Consider what happens if the current value of ρ is less than $\rho(\mu)$. Recall that $h(i)$ is updated to be $\max_{u \in U(i)}\{r_i(u) + \sum_{j=1}^{n} p_{i,j}(u)h(j)\} - \rho$. Ignoring the changes to ρ itself, the h values for states in the current

greedy policy tend to increase on the average, because the sum of immediate rewards for this policy in any n steps is likely to be higher than $n\rho$ (since $\rho < \rho(\mu)$). It is possible, under these circumstances, that the h values of *all* states in the current policy increase or stay the same. Since the h values of states not visited by this policy do not change, this implies that by executing the greedy policy, the system can never get out of this set of states. If the optimal policy involves going through states not visited by the greedy policy, it will never be learned.

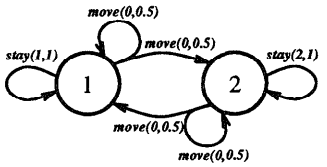

Figure 4: The Two-State domain. The notation $\mathsf{m}(r,p)$ on the arc from state a to b indicates that r is the immediate reward and p is the probability of the next state being b when action m is executed in state a.

This is illustrated clearly in the Two-State MDP in Figure 4, which is a communicating multichain. In this domain, the optimal policy μ^* is taking the action move in state 1 and stay in state 2 with $\rho(\mu^*) = 2$. Without any exploration, H-learning finds the optimal policy in approximately half of the trials for this domain – those trials in which the stay action in state 2 is executed before the stay action in state 1. If the stay action in state 1 is executed before that in state 2, it receives a reward of +1 and updates $h(1)$ to $1 + h(1) - \rho$. Since ρ is between 0 and 1, this increases the value of $h(1)$ in every update until finally ρ converges to 1. Since greedy action choice always results in the stay action in state 1, H-learning never visits state 2 and therefore converges to a suboptimal policy.

Now consider what happens if $\rho > \rho(\mu)$ for the current greedy policy μ. In this case, by the same argument as before, the h values of the states in the current greedy policy must *decrease* on the average. This means that eventually the states outside the set of states visited by the greedy policy will have their h values higher than some of those visited by the greedy policy. Since the MDP is assumed to be weakly communicating, the states with higher h values are reachable from the states with decreasing h values, and eventually will be visited, ignoring the transient states that do not affect the gain. Thus, as long as $\rho > \rho(\mu)$, there is no danger of getting stuck in a sub-optimal policy μ. This suggests changing H-learning so that it starts with a high initial ρ value, ρ_0, high enough so that it never gets below the gain of any sub-optimal policy.

In the preceding discussion, we ignored the changes to the ρ value itself. In fact, ρ is constantly changing at a rate determined by α. Hence, even though ρ was initially higher than $\rho(\mu)$, because it is now decreasing, it can become smaller than $\rho(\mu)$ after a while. To make the previous argument work, we have to adjust α so that ρ changes slowly compared to the h values. This can be done by starting with a sufficiently low initial α value, α_0. We denote H-learning with the initial values ρ_0 and α_0 by $\mathrm{H}^{\rho_0,\alpha_0}$. Hence, the H-learning of the previous section is $\mathrm{H}^{0,1}$.

So far, we have considered the effect of lack of exploration on the h-values. We now turn to its effect on the accuracy of action models. For the rest of the discussion, it is useful to define the utility $R(i,u)$ of a state action pair (i,u) to be

$$R(i,u) = r_i(u) + \sum_{j=1}^{n} p_{i,j}(u)h(j) - \rho. \qquad (2)$$

Hence, the greedy actions in state i are actions that maximize the R value in state i.

Consider the following run of $\mathrm{H}^{6,0.2}$ in the Two-State domain, where, in step 1, the agent executes the action stay in state 1. It reduces $h(1) = R(1,\mathsf{stay})$ to $1-\rho$ and takes the action move in the next step. Assume that move takes it to state 1 because it has 50% failure rate. With this limited experience, the system assumes that both the actions have the same next state in state 1, and stay has a reward +1 while move has 0. Hence, it determines that $R(1,\mathsf{stay}) = 1 + h(1) - \rho > 0 + h(1) - \rho = R(1,\mathsf{move})$ and continues to execute stay, and keeps decreasing the value of $h(1)$. Even though $h(2) > h(1)$, the agent cannot get to state 2 because it does not have the correct action model. Therefore, it cannot learn the correct action model for move, and keeps executing stay. Unfortunately, this problem cannot be fixed by changing ρ_0 or α_0.

The solution we have implemented, called "Auto-exploratory H-Learning" (AH-learning), starts with a high ρ_0 and low α_0 ($\mathrm{AH}^{\rho_0,\alpha_0}$), and stores the R values explicitly. In H-learning, all R values of the same state are effectively updated at the same time by updating the h value, which sometimes makes it converge to incorrect action models. In AH-learning, $R(i,u)$ is updated by the right hand side of Equation (2) only when action u is taken in state i.

When ρ is higher than the gain of the current greedy policy, the R value of the executed action is decreased, while the R values of the other actions remain the same. Therefore, eventually, the unexecuted actions appear to be the best in the current state, forcing the system to explore such actions.

We experimented with AH and H with different parameters ρ_0 and α_0 in our Two-State domain without any exploratory moves. Out of the 100 trials tested, $\mathrm{AH}^{6,0.2}$ found the optimal policy in all of them, whereas $\mathrm{H}^{0,1}$, and $\mathrm{H}^{6,0.2}$ found it in 57 and 69 trials respectively. This confirms our hypothesis that AH-learning explores the search space effectively while always executing greedy actions.

Experimental Results on AH-learning

In this section, we compare AH-learning with some other exploration strategies in the Delivery domain of Figure 2. Unlike H-learning, our implementation of AH-learning does not explicitly store the policy.

We compared AH to three other exploration methods: random exploration, counter-based exploration and recency-based exploration (Thrun 94). Random exploration was used as before, except that we optimized the probability with which random actions are selected. In counter-based exploration, in any state i, an action a is chosen that maximizes $R(i,a) + \delta \frac{c(i)}{\sum_{j=1}^{n} P_{i,j}(u) c(j)}$, where $c(i)$ is the number of times state i is visited, and δ is a small positive constant. In recency-based exploration, an action a is selected which maximizes $R(i,a) + \epsilon\sqrt{n(i,a)}$, where $n(i,a)$ is the number of steps since the action a is executed in state i last, and ϵ is a small constant. In all the three cases, the parameters were tuned by trail and error until they gave the best performance.

The parameters for the Delivery domain, p, q and K, were set to 0.5, 0.0 and 5. Proper exploration is particularly important in this domain for the following reasons: First, the domain is stochastic; second, it takes many steps to propagate high rewards; and third, there are many sub-optimal policies with gain close to the optimal gain. For all these reasons, it is difficult to maintain ρ consistently higher than the gain of any sub-optimal policy, which is important for AH-learning to find the optimal policy. It gave the best performance with $\rho_0 = 2$ and $\alpha_0 = 0.0002$.

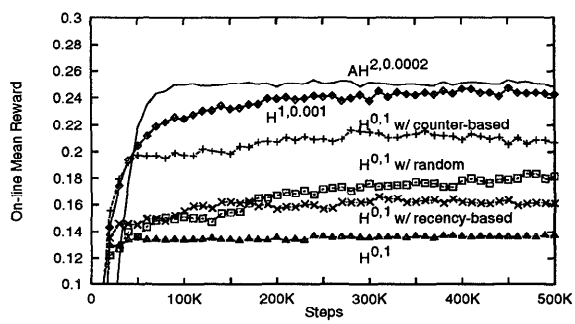

Figure 5: The on-line mean rewards of the last 10K steps averaged over 30 trials for $AH^{2,0.0002}$, $H^{1,0.001}$, and $H^{0,1}$ without exploration, and $H^{0,1}$ with random, recency-based, and counter-based exploration strategies in the Delivery domain with p=0.5, q=0.0, and K=5.

Figure 5 shows the on-line mean rewards of 30 trials of $AH^{2,0.0002}$, $H^{1,0.001}$, and $H^{0,1}$, with no exploratory actions, and of $H^{0,1}$ with 3 different exploration methods: 8% random exploration, counter-based exploration with $\delta = 0.05$, and recency-based exploration with $\epsilon = 0.02$.

Without any exploration, $H^{0,1}$ could not find the optimal policy even once. By proper tuning of ρ_0 and α_0, it improved significantly, and was only slightly worse than AH, which found the optimal policy in all 30 trials. Counter-based exploration appears much better than random exploration for this domain, while recency-based exploration seems worse. AH achieved a much better on-line reward than all other exploration methods, and did so more quickly than others.

This result suggests that with proper initialization of ρ and α, AH-learning automatically explores the state space much more effectively than the other exploration schemes tested. A particularly attractive feature of AH-learning is that it does so without sacrificing any gain, and hence should be preferred to other methods. Although AH-learning does involve tuning two parameters ρ and α, it appears that at least ρ can be automatically adjusted. One way to do this is to keep track of the currently known maximum immediate reward over all state-action pairs, and reinitialize ρ to something higher than this value whenever it changes.

Discussion and Future Work

There is an extensive literature on average reward optimization using dynamic programming approaches (Howard 60; Puterman 94; Bertsekas 95). (Mahadevan 96a) gives a useful survey of this literature from Reinforcement Learning point of view. Bias-optimality, or Schwartz's T-optimality, is a more refined notion that seeks to find a policy that maximizes the cumulated discounted total reward for all states as the discount factor $\gamma \rightarrow 1$. All bias-optimal policies are also gain-optimal, but the converse does not hold. H-learning and R-learning can find the bias-optimal policies if and only if all gain-optimal policies give rise to the same recurrent set of states, and the transient states are repeatedly visited by some trial-based exploration strategy. To find bias-optimal policies for more general unichains, it is necessary to select bias-optimal actions from among the gain-optimal ones in every state using more refined criteria. Mahadevan extends both H-learning and R-learning to find the bias-optimal policies for general unichains, and illustrates that they improve their performance in an admission control queuing system (Mahadevan 96b; Mahadevan 96c).

Auto-exploratory H-learning is similar in spirit to the action-penalty representation of reward functions analyzed by Koenig and Simmons (Koenig & Simmons 96). They showed that a minimax form of Q-learning, which always takes greedy actions, can find the shortest cost paths from all states to a goal state in $O(n^3)$ time, where n is the size of the state space. An analytical convergence result of this kind for AH-learning would be very interesting. In the *Interval Estimation* algorithm (IE) of Kaelbling (Kaelbling 90), which is similar, the agent maintains a confidence interval of the value function, and always chooses an action that maximizes its upper bound. This ensures that the chosen

action either has a high utility, or has a large confidence interval which needs exploration to reduce it.

The idea of Auto-exploratory learning can be adapted to R-learning as well. However, in our preliminary experiments we found that the value of ρ fluctuates much more in R-learning than in H-learning, unless α is initialized to be very small. Making α small will have the consequence of slowing down learning. Although we have not seen this to be a problem with H-learning, the tradeoff between the need for exploration and slow learning due to small α value deserves further study.

To scale H-learning to large domains, it is necessary to approximate its value function and action models. Elsewhere, we describe our results of learning action models in the form of conditional probability tables of a Bayesian network, and using local linear regression to approximate its value function (Tadepalli & Ok 96). Both these extensions improve the space requirement of H-learning and the number of steps needed for its convergence. We also plan to explore other ways of approximating the value function which can be effective in multi-dimensional spaces.

Summary

The premise of our work is that many real-world domains demand optimizing average reward per time step, while most work in Reinforcement Learning is focused on optimizing discounted total reward. We presented a model-based average reward RL method called H-learning that demonstrably performs better than its discounted counterpart. We also presented Auto-exploratory H-learning, which automatically explores while always picking greedy actions with respect to its current value function. We showed that it outperforms many currently known exploration methods. Value function approximation for ARL systems is currently under investigation.

References

Barto, A. G., Bradtke, S. J., and Singh, S. P. 1995. Learning to Act using Real-Time Dynamic Programming. *Artificial Intelligence*, 73(1), 81-138.

Bertsekas, D. P. 1982. Distributed Dynamic Programming. In *IEEE Transactions in Automatic Control*, AC-27(3).

Bertsekas, D. P. 1995. *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes.* MIT press, Cambridge, MA.

Jalali, A. and Ferguson, M. 1989. Computationally Efficient Adaptive Control Algorithms for Markov Chains. In *IEEE Proceedings of the 28'th Conference on Decision and Control*, Tampa, FL.

Kaelbling, L. P. 1990. *Learning in Embedded Systems*, MIT Press, Cambridge, MA.

Koenig, S., and Simmons, R. G. 1996. The Effect of Representation and Knowledge on Goal-Directed Exploration with Reinforcement-Learning Algorithms. *Machine Learning*, 22, 227-250.

Lin, L-J. 1992 Self-improving Reactive Agents Based on Reinforcement Learning, Planning, and Teaching. *Machine Learning*, 8, 293-321.

Mahadevan, S. and Connell, J. 1992. Automatic Programming of Behavior-based Robots Using Reinforcement Learning. *Artificial Intelligence*, 55:311-365.

Mahadevan, S. 1996a. Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results. *Machine Learning*, 22, 159-195.

Mahadevan, S. 1996b. An Average Reward Reinforcement Learning Algorithm for Computing Bias-optimal Policies. In *Proceedings of AAAI-96*, Portland, OR.

Mahadevan, S. 1996c. Sensitive Discount Optimality: Unifying Discounted and Average Reward Reinforcement Learning. In *Proceedings of International Machine Learning Conference*, Bari, Italy.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Dynamic Stochastic Programming.* John Wiley.

Schwartz, A. 1993. A Reinforcement Learning Method for Maximizing Undiscounted Rewards. In *Proceedings of International Machine Learning Conference*, Morgan Kaufmann, San Mateo, CA.

Singh, S. P. 1994. Reinforcement Learning Algorithms for Average-Payoff Markovian Decision Processes. In *Proceedings of AAAI-94*, MIT press.

Tadepalli, P. and Ok, D. 1994. H-learning: A Reinforcement Learning Method for Optimizing Undiscounted Average Reward. Technical Report, 94-30-1, Dept. of Computer Science, Oregon State University.

Tadepalli, P. and Ok, D. 1996. Scaling Up Average Reward Reinforcement Learning by Approximating the Domain Models and the Value Function. In *Proceedings of International Machine Learning Conference*, Bari, Italy.

Thrun, S. 1994. The Role of Exploration in Learning Control. In *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, Van Nostrand Reinhold.

Watkins, C. J. C. H. and Dayan, P. 1992. Q-learning. *Machine Learning*, 8:279-292.

Acknowledgments

We gratefully acknowledge the support of NSF under grant number IRI-9520243. We thank Tom Dietterich, Sridhar Mahadevan, and Toshi Minoura for many interesting discussions on this topic. Thanks to Sridhar Mahadevan and the reviewers of this paper for their thorough and helpful comments.