Machine Learning, 22, 159–196 (1996)

Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results

SRIDHAR MAHADEVAN mahadeva@samuel.csee.usf.edu
Department of Computer Science and Engineering, University of South Florida, Tampa, Florida 33620

Editor: Leslie Kaelbling

Abstract. This paper presents a detailed study of average reward reinforcement learning, an undiscounted optimality framework that is more appropriate for cyclical tasks than the much better studied discounted framework. A wide spectrum of average reward algorithms are described, ranging from synchronous dynamic programming methods to several (provably convergent) asynchronous algorithms from optimal control and learning automata. A general sensitive discount optimality metric called *n-discount-optimality* is introduced, and used to compare the various algorithms. The overview identifies a key similarity across several asynchronous algorithms that is crucial to their convergence, namely independent estimation of the average reward and the relative values. The overview also uncovers a surprising limitation shared by the different algorithms: while several algorithms can provably generate *gain-optimal* policies that maximize average reward, none of them can reliably filter these to produce *bias-optimal* (or *T-optimal*) policies that also maximize the finite reward to absorbing goal states. This paper also presents a detailed empirical study of R-learning, an average reward reinforcement learning method, using two empirical testbeds: a stochastic grid world domain and a simulated robot environment. A detailed sensitivity analysis of R-learning is carried out to test its dependence on learning rates and exploration levels. The results suggest that R-learning is quite sensitive to exploration strategies, and can fall into sub-optimal limit cycles. The performance of R-learning is also compared with that of Q-learning, the best studied discounted RL method. Here, the results suggest that R-learning can be fine-tuned to give better performance than Q-learning in both domains.

Keywords: Reinforcement learning, Markov decision processes

1. Introduction

Machine learning techniques that enable autonomous agents to adapt to dynamic environments would find use in many applications, ranging from flexible robots for automating chores [11] to customized software apprentices for managing information [10]. Recently, an adaptive control paradigm called reinforcement learning (RL) has received much attention [3], [19], [36]. Here, an agent is placed in an initially unknown task environment, and learns by trial and error to choose actions that maximize, over the long run, rewards that it receives. RL has been shown to scale much better than related dynamic programming (DP) methods for solving Markov decision processes, such as policy iteration [15]. For example, Tesauro [39] recently produced a grandmaster backgammon program using Sutton's TD(λ) temporal difference learning algorithm [34].

Most of the research in RL has studied a problem formulation where agents maximize the cumulative sum of rewards. However, this approach cannot handle infinite horizon tasks, where there are no absorbing goal states, without discounting future rewards. Traditionally, discounting has served two purposes. In some domains, such as economics, discounting can be used to represent "interest" earned on rewards, so that an action that generates an immediate reward will be preferred over one that generates the same reward some steps into the future. However, the typical domains studied in RL, such as robotics or games, do not fall in this category. In fact, many RL tasks have absorbing goal states, where the aim of the agent is to get to a given goal state as quickly as possible. As Kaelbling showed [18], such tasks can be solved using undiscounted RL methods.

Clearly, discounting is only really necessary in cyclical tasks, where the cumulative reward sum can be unbounded. Examples of such tasks include a robot learning to avoid obstacles [25] or an automated guided vehicle (AGV) serving multiple queues [38]. Discounted RL methods can and have been applied to such tasks, but can lead to sub-optimal behavior if there is a short term mediocre payoff solution that looks more attractive than a more long-term high reward one (see Figure 1).

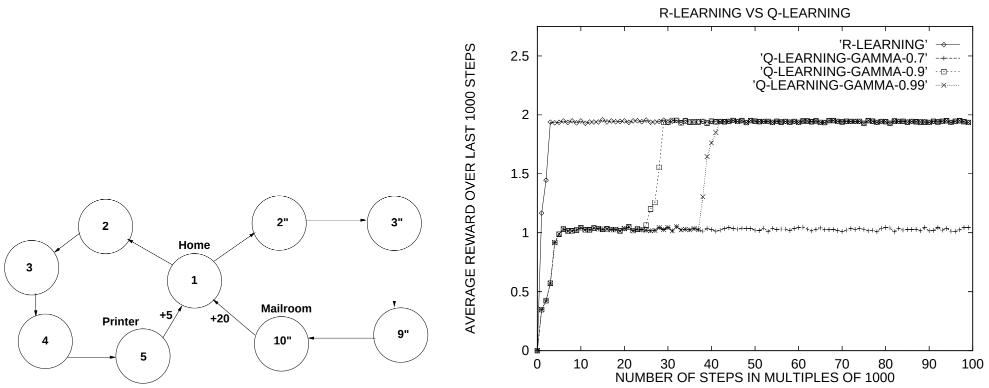

Figure 1. An example to illustrate why an average-reward RL method (R-learning) is preferable to a discounted RL method (Q-learning) for cyclical tasks. Consider a robot that is rewarded by +5 if it makes a roundtrip service from "Home" to the "Printer", but rewarded +20 for servicing the more distant "Mailroom". The only action choice is in the "Home" state, where the robot must decide the service location. The accompanying graph shows that if the discount factor γ is set too low (0.7), Q-learning converges to a suboptimal solution of servicing the "Printer". As γ is increased, Q-learning infers the optimal policy of servicing the "Mailroom", but converges much more slowly than R-learning.

A more natural long-term measure of optimality exists for such cyclical tasks, based on maximizing the *average* reward per action. There are well known classical DP algorithms for finding optimal average reward policies, such as policy iteration [15] and value iteration [45]. However, these algorithms require knowledge of the state transition probability matrices, and are also computationally intractable. RL methods for producing optimal average reward policies should scale much better than classical DP methods. Unfortunately, the study of average reward RL is currently at an early stage. The first average-reward RL method (R-learning) was proposed only relatively recently by Schwartz [31].

Schwartz hypothesized several reasons why R-learning should outperform discounted methods, such as Q-learning [43], but did not provide any detailed experimental results to support them. Instead, he used simplified examples to illustrate his arguments, such as the one in Figure 1. Here, Q-learning converges much more slowly than R-learning, because it initially chooses the closer but more mediocre reward over the more distant but better reward.[1] An important question, however, is whether the superiority of R-learning will similarly manifest itself on larger and more realistic problems. A related question is whether average-reward RL algorithms can be theoretically shown to converge to optimal policies, analogous to Q-learning.

This paper undertakes a detailed examination of average reward reinforcement learning. First, a detailed overview of average reward Markov decision processes is presented, covering a wide range of algorithms from dynamic programming, adaptive control, learning automata, and reinforcement learning. A general optimality metric called *n-discount-optimality* [42] is introduced to relate the discounted and average reward frameworks. This metric is used to compare the strengths and limitations of the various average reward algorithms. The key finding is that while several of the algorithms described can provably yield optimal average reward (or *gain-optimal* [15]) policies, none of them can also reliably discriminate among these to obtain a *bias-optimal* [6] (or what Schwartz [31] referred to as *T-optimal*) policy. Bias-optimal policies maximize the finite reward incurred in getting to absorbing goal states.

While we do not provide a convergence proof for R-learning, we lay the groundwork for such a proof. We discuss a well known counter-example by Tsitsiklis (described in [4]) and show why it does not rule out provably convergent gain-optimal RL methods. In fact, we describe several provably convergent asynchronous algorithms, and identify a key difference between these algorithms and the counter-example, namely independent estimation of the average reward and the relative values. Since R-learning shares this similarity, it suggests that a convergence proof for R-learning is possible.

The second contribution of this paper is a detailed empirical study of R-learning using two testbeds: a stochastic grid world domain with membranes, and a simulated robot environment. The sensitivity of R-learning to learning rates and exploration levels is also studied. The performance of R-learning is compared with that of Q-learning, the best studied discounted RL method. The results can be summarized into two findings: R-learning is more sensitive than Q-learning to exploration strategies, and can get trapped in limit cycles; however, R-learning can be fine-tuned to outperform Q-learning in both domains.

The rest of this paper is organized as follows. Section 2 provides a detailed overview of average reward Markov decision processes, including algorithms from DP, adaptive control, and learning automata, and shows how R-learning is related to them. Section 3 presents a detailed experimental test of R-learning using two domains, a stochastic grid world domain with membranes, and a simulated robot domain. Section 4 draws some conclusions regarding average reward methods, in general, and R-learning, in particular. Finally, Section 5 outlines some directions for future work.

2. Average Reward Markov Decision Processes

This section introduces the reader to the study of average reward Markov decision processes (MDP). It describes a wide spectrum of algorithms ranging from synchronous DP algorithms to asynchronous methods from optimal control and learning automata. A general sensitive discount optimality metric is introduced, which nicely generalizes both average reward and discounted frameworks, and which is useful in comparing the various algorithms. The overview clarifies the relationship of R-learning to the (very extensive) previous literature on average reward MDP. It also highlights a key similarity between R-learning and several provably convergent asynchronous average reward methods, which may help in its further analysis.

The material in this section is drawn from a variety of sources. The literature on average reward methods in DP is quite voluminous. Howard [15] pioneered the study of average reward MDP's, and introduced the policy iteration algorithm. Bias optimality was introduced by Blackwell [6], who pioneered the study of average reward MDP as a limiting case of the discounted MDP framework. The concept of n-discount-optimality was introduced by Veinott [42]. We have drawn much of our discussion, including the key definitions and illustrative examples, from Puterman's recent book [28], which contains an excellent treatment of average reward MDP.

By comparison, the work on average reward methods in RL is in its infancy. Schwartz's original paper on R-learning [31] sparked interest in this area in the RL community. Other more recent work include that of Singh [33], Tadepalli and Ok [38], and Mahadevan [23]. There has been much work in the area of adaptive control on algorithms for computing optimal average reward policies; we discuss only one algorithm from [16]. Finally, we have also included here a provably convergent asynchronous algorithm from learning automata [27], which has only recently come to our attention, and which to our knowledge has not been discussed in the previous RL literature. Average reward MDP has also drawn attention in recent work on decision-theoretic planning (e.g. see Boutilier and Puterman [7]).

2.1. Markov Decision Processes

A Markov Decision Process (MDP) consists of a (finite or infinite) set of states S, and a (finite or infinite) set of actions A for moving between states. In this paper we will assume that S and A are finite. Associated with each action a is a state transition matrix $P(a)$, where $P_{xy}(a)$ represents the probability of moving from state x to y under action a. There is also a reward or payoff function $r : S \times A \rightarrow \mathcal{R}$, where $r(x, a)$ is the *expected* reward for doing action a in state x.

A *stationary deterministic policy* is a mapping $\pi : S \rightarrow A$ from states to actions. In this paper we restrict our discussion to algorithms for such policies, with the exception of the learning automata method, where we allow stationary randomized policies. In any case, we do not allow nonstationary policies. [2] Any policy induces a state transition matrix $P(\pi)$, where $P_{xy}(\pi) = P_{xy}(\pi(x))$. Thus, any policy yields a Markov chain $(S, P(\pi))$. A key difference between discounted and average reward frameworks is that the policy chain structure plays a critical role in average reward methods. All the algorithms described here incorporate some important assumptions about the underlying MDP, which need to be defined before presenting the algorithms.

Two states x and y *communicate* under a policy π if there is a positive probability of reaching (through zero or more transitions) each state from the other. It is easy to see that the communication relation is an equivalence relation on states, and thus partitions the state space into communicating classes. A state is *recurrent* if starting from the state, the probability of eventually reentering it is 1. Note that this implies that recurrent states will be visited forever. A non-recurrent state is called *transient*, since at some finite point in time the state will never be visited again. Any finite MDP must have recurrent states, since not all states can be transient. If a recurrent state x communicates with another state y, then y has to be recurrent also.

An *ergodic* or recurrent class of states is a set of recurrent states that all communicate with each other, and do not communicate with any state outside this class. If the set of all states forms an ergodic class, the Markov chain is termed *irreducible*. Let $p^n_{s,s}(\pi)$ denote the probability of reaching state s from itself in n steps using policy π. The period of a state s under policy π is the greatest common divisor of all n for which $p^n_{s,s}(\pi) > 0$. A state is termed periodic if its period exceeds 1, else it is aperiodic. States in a given recurrence class all have the same period.

An *ergodic or recurrent* MDP is one where the transition matrix corresponding to every (deterministic stationary) policy has a single recurrent class. An MDP is termed *unichain* if the transition matrix corresponding to every policy contains a single recurrent class, and a (possibly empty) set of transient states. An MDP is *communicating* if every pair of states communicate under some stationary policy. Finally, an MDP is *multichain* if there is at least one policy whose transition matrix has two or more recurrent classes. We will not discuss algorithms for multichain MDP's in this paper.

Figure 2 contains two running examples that we will use to illustrate these concepts, as well as to explain the following algorithms. These examples originally appeared in [28]. For simplicity, in both MDP's, there is a choice of action only in state **A**. Each transition is labeled with the action taken, and a pair of numbers. The first number is the immediate reward, and the second number represents the transition probability. Both MDP's are unichain. In the two-state MDP, state **A** is transient under either policy (that is, doing action **a1** or **a2** in state **A**), and state

B is recurrent. Such recurrent single-state classes are often called *absorbing* states. Both states are aperiodic under the policy that selects **a1**.

In the three-state MDP, there are two recurrent classes, one for each policy. If action **a1** is taken in state **A**, the recurrent class is formed by **A** and **B**, and is periodic with period 2. If action **a2** is taken, the recurrent class is formed by **A** and **C**, which is also periodic with period 2. As we will show below, these differences between the two MDP's will highlight themselves in the operation of the algorithms.

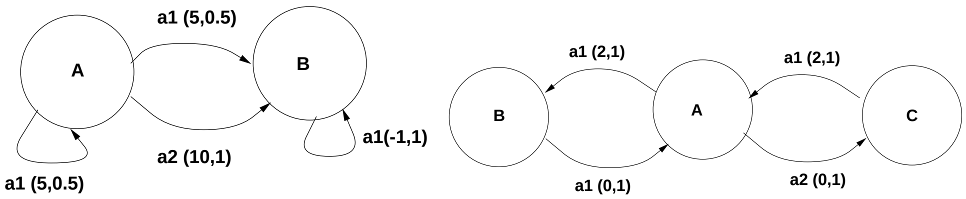

Figure 2. Two simple MDP's that illustrate the underlying concepts and algorithms in this section. Both policies in the two MDP's maximize average reward and are gain-optimal. However, only the policy that selects action **a1** in state **A** is bias-optimal (T-optimal) in both MDP's. While most of the algorithms described here can compute the bias-optimal policy for the 2 state MDP, none of them can do so for the 3 state MDP.

2.2. Gain and Bias Optimality

The aim of average reward MDP is to compute policies that yield the highest expected payoff per step. The average reward $\rho^\pi(x)$ associated with a particular policy π at a state x is defined as [3]

$$\rho^\pi(x) = \lim_{N\to\infty} \frac{E(\sum_{t=0}^{N-1} R_t^\pi(x))}{N},$$

where $R_t^\pi(x)$ is the reward received at time t starting from state x, and actions are chosen using policy π. $E(.)$ denotes the expected value. We define a *gain-optimal* policy π^* as one that maximizes the average reward over all states, that is $\rho^{\pi^*}(x) \geq \rho^\pi(x)$ over all policies π and states x.

A key observation that greatly simplifies the design of average reward algorithms is that for unichain MDP's, the average reward of any policy is state independent. That is,

$$\rho^\pi(x) = \rho^\pi(y) = \rho^\pi.$$

The reason is that unichain policies generate a single recurrent class of states, and possibly a set of transient states. Since states in the recurrent class will be visited forever, the expected average reward cannot differ across these states. Since the transient states will eventually never be reentered, they can at most accumulate a finite total expected reward before entering a recurrent state, which vanishes under the limit.

For the example MDP's in Figure 2, note that the average rewards of the two policies in the two-state MDP are both -1. In the three-state MDP, once again, the average rewards of the two policies are identical and equal to 1.

2.2.1. *Bias Optimality*

Since the aim of solving average reward MDP's is to compute policies that maximize the expected payoff per step, this suggests defining an optimal policy π^* as one that is gain-optimal. Figure 3 illustrates a problem with this definition. MDP's of this type naturally occur in goal-oriented tasks. For example, consider a two-dimensional grid-world problem, where the agent is rewarded by $+1$ if an absorbing goal state is reached, and is rewarded -1 otherwise. Clearly, in this case, all policies that reach the goal will have the same average reward, but we are mainly interested in policies that reach the goal in the shortest time, i.e. those that maximize the finite reward incurred in reaching the goal.

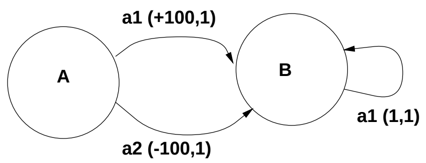

Figure 3. In this MDP, both policies yield the same average reward, but doing action **a1** is clearly preferrable to doing **a2** in state **A**.

The notion of *bias optimality* is needed to address such problems. We need to define a few terms before introducing this optimality metric. A *value* function $V : S \to \mathcal{R}$ is any mapping from states to real numbers. In the traditional discounted framework, any policy π induces a value function that represents the cumulative discounted sum of rewards earned following that policy starting from any state s

$$V_\gamma^\pi(s) = \lim_{N\to\infty} E(\sum_{t=0}^{N-1} \gamma^t R_t^\pi(s)),$$

where $\gamma < 1$ is the discount factor, and $R_t^\pi(s)$ is the reward received at time t starting from state s under policy π. An *optimal discounted policy* π^* maximizes the above value function over all states x and policies π, i.e. $V_\gamma^{\pi^*}(x) \geq V_\gamma^\pi(x)$. In average reward MDP, since the undiscounted sum of rewards can be unbounded, a policy π is measured using a different value function, namely the *average adjusted* sum of rewards earned following that policy, [4]

$$V^\pi(s) = \lim_{N\to\infty} E(\sum_{t=0}^{N-1} (R_t^\pi(s) - \rho^\pi)),$$

where ρ^π is the average reward associated with policy π. The term $V^\pi(x)$ is usually referred to as the *bias* value, or the *relative* value, since it represents the relative difference in total reward gained from starting in state s as opposed to some other state x. In particular, it can be easily shown that

$$V^\pi(s) - V^\pi(x) = \lim_{N\to\infty}\left(E(\sum_{t=0}^{N-1} R_t^\pi(s)) - E(\sum_{t=0}^{N-1} R_t^\pi(x))\right).$$

A policy π^* is termed *bias-optimal* if it is gain-optimal, and it also maximizes bias values, that is $V^{\pi^*}(x) \geq V^\pi(x)$ over all $x \in S$ and policies π. Bias optimality was referred to as *T-optimality* by Schwartz [31]. The notion of bias-optimality was introduced by Blackwell [6], who also pioneered the study of average reward MDP as the limiting case of the discounted framework. In particular, he showed how the gain and bias terms relate to the total expected discounted reward using a Laurent series expansion. A key result is the following truncated Laurent series expansion of the discounted value function V_γ^π [28]:

LEMMA 1 *Given any policy π and state s,*

$$V_\gamma^\pi(s) = \frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s) + e^\pi(s,\gamma)$$

where $\lim_{\gamma\to 1} e^\pi(s,\gamma) = 0$.

We will use this corollary shortly. First, we define a general optimality metric that relates discounted and average reward RL, which is due to Veinott [42]:

DEFINITION 1 *A policy π^* is* n-discount-optimal, *for $n = -1, 0, 1, ...$, if for each state s, and over all policies π,*

$$\lim_{\gamma\to 1}(1-\gamma)^{-n}\left(V_\gamma^{\pi^*}(s) - V_\gamma^\pi(s)\right) \geq 0.$$

We now consider some special cases of this general optimality metric to illustrate how it nicely generalizes the optimality metrics used previously in both standard discounted RL and average reward RL. In particular, we show below (see Lemma 2) that gain optimality is equivalent to *-1-discount-optimality.* Furthermore, we also show (see Lemma 3) that bias optimality is equivalent to *0-discount-optimality.* Finally, the traditional discounted MDP framework can be viewed as studying 0-discounted-optimality for a *fixed* value of γ.

In the two-state MDP in Figure 2, the two policies have the same average reward of -1, and are both gain-optimal. However, the policy π that selects action **a1** in state **A** results in bias values $V^\pi(A) = 12$ and $V^\pi(B) = 0$. The policy π' that selects action **a2** in state **A** results in bias values $V^{\pi'}(A) = 11$ and $V^{\pi'}(B) = 0$. Thus, π is better because it is also bias-optimal. In the 3-state MDP, both policies are again gain-optimal since they yield an average reward of 1. However, the policy π that selects action **a1** in state **A** generates bias values $V^\pi(A) = 0.5$, $V^\pi(B) = -0.5$, and $V^\pi(C) = 1.5$. The policy π is bias-optimal because the only other policy is π' that selects action **a2** in state **A**, and generates bias values $V^{\pi'}(A) = -0.5$, $V^\pi(B) = -1.5$, and $V^\pi(C) = 0.5$.

It is easy to see why we might prefer bias-optimal policies, but when do we need to consider a more selective criterion than bias optimality? Figure 4 shows a simple MDP that illustrates why *n-discount-optimality* is useful for $n > 0$.

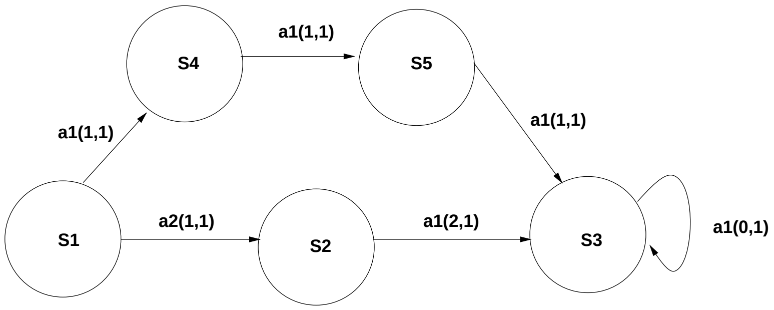

Figure 4. A simple MDP that clarifies the meaning of *n-discount-optimality*. There are only 2 possible policies, corresponding to the two actions in state **S1**. Both policies yield an average reward of 0, and are *-1-discount-optimal*. Both policies yield a cumulative reward of 3 in reaching the absorbing goal state **S3**, and are also *0-discount-optimal*. However, the bottom policy reaches the goal quicker than the top policy, and is *1-discount-optimal*. Finally, the bottom policy is also Blackwell optimal, since it is *n-discount-optimal*, for all $n \geq -1$.

For brevity, we only include proofs showing one side of the equivalences for bias optimality and gain optimality. More detailed proofs can be found in [28].

LEMMA 2 *If* π^* *is a* -1-discount-optimal *policy, then* π^* *is a gain-optimal policy.*

Proof: Let π^* be -1-discount-optimal, and let π be any other policy. It follows from Definition 1 that over all states s

$$\lim_{\gamma \to 1} (1-\gamma)(V_\gamma^{\pi^*}(s) - V_\gamma^\pi(s)) \geq 0.$$

Using Lemma 1, we can transform the above equation to

$$\lim_{\gamma \to 1} (1-\gamma) \left(\frac{\rho^{\pi^*}(s) - \rho^\pi(s)}{1-\gamma} + V^{\pi^*}(s) + e^{\pi^*}(s,\gamma) - V^\pi(s) - e^\pi(s,\gamma) \right) \geq 0.$$

Noting that $e^{\pi^*}(s,\gamma)$ and $e^\pi(s,\gamma)$ both approach 0 as $\gamma \to 1$, the above equation implies that

$$(\rho^{\pi^*}(s) - \rho^\pi(s)) \geq 0.$$

■

LEMMA 3 *If* π^* *is a* 0-discount-optimal *policy, then* π^* *is a bias-optimal (or T-optimal) policy.*

Proof: Note that we need only consider gain-optimal policies, since as $\gamma \to 1$, the first term on the right hand side in Lemma 1 dominates, and hence we can ignore all policies where $\rho^\pi < \rho^{\pi^*}$. From the definition, it follows that if π^* is a 0-discount-optimal policy, then for all gain-optimal policies π, over all states s

$$\lim_{\gamma \to 1} (V_\gamma^{\pi^*}(s) - V_\gamma^\pi(s)) \geq 0.$$

As before, we can expand this to yield

$$\lim_{\gamma \to 1} \left((\frac{\rho^{\pi^*}(s)}{1-\gamma} + V^{\pi^*}(s) + e^{\pi^*}(s,\gamma)) - (\frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s) + e^\pi(s,\gamma)) \right) \geq 0.$$

Since π^* and π are both gain-optimal, $\rho^{\pi^*} = \rho^\pi$, and hence

$$\lim_{\gamma \to 1} \left(((V^{\pi^*}(s) + e^{\pi^*}(s,\gamma)) - (V^\pi(s) + e^\pi(s,\gamma)) \right) \geq 0.$$

Since both $e^{\pi^*}(s,\gamma)$ and $e^\pi(s,\gamma)$ approach 0 as $\gamma \to 1$, the above equation implies that

$$(V^{\pi^*}(s) - V^\pi(s)) \geq 0.$$

■

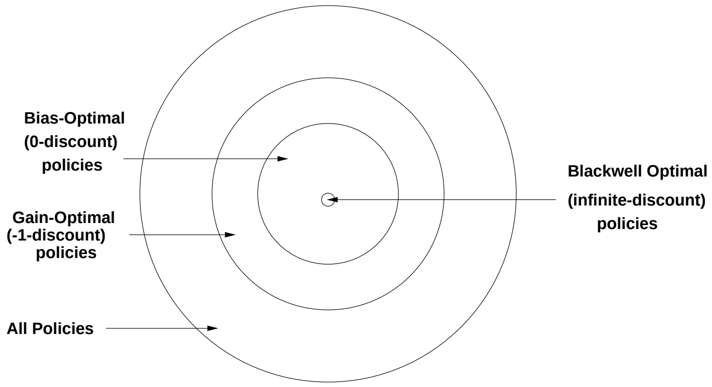

Figure 5. This diagram illustrates the concept of *n-discount-optimal* policies. Gain-optimal policies are *-1-discount-optimal* policies. Bias-optimal (or T-optimal) policies are *0-discount-optimal* policies. Note that *n-discount-optimal* policies get increasingly selective as n increases. Blackwell optimal policies are *n-discount-optimal* for all $n \geq -1$.

Figure 5 illustrates the structure of *n-discount-optimal* policies. The most selective optimality criterion is *Blackwell optimality*, and corresponds to a policy π^*

that is *n-discount-optimal* for all $n \geq -1$. It is interesting to note that a policy that is *m-discount-optimal* will also be *n-discount-optimal* for all $n < m$. Thus, bias optimality is a more selective optimality metric than gain optimality. This suggests a computational procedure for producing Blackwell-optimal policies, by starting with *-1-discount-optimal* policies, and successively pruning these to produce *i-discount-optimal* policies, for $i > -1$. Exactly such a policy iteration procedure for multichain MDP's is described in [28]. In this paper, we restrict our attention to -1-discount-optimal and 0-discount-optimal policies.

2.3. Average Reward Bellman Equation

Clearly, it is useful to know the class of finite state MDP's for which (gain or bias) optimal policies exist. A key result, which essentially provides an average reward Bellman optimality equation, is the following theorem (see [5], [28] for a proof):

THEOREM 1 *For any MDP that is either unichain or communicating, there exist a value function V^* and a scalar ρ^* satisfying the equation*

$$V^*(x) + \rho^* = \max_a \left(r(x,a) + \sum_y P_{xy}(a) V^*(y) \right) \tag{1}$$

such that the greedy policy π^* resulting from V^* achieves the optimal average reward $\rho^* = \rho^{\pi^*}$ where $\rho^{\pi^*} \geq \rho^{\pi}$ over all policies π.

Here, "greedy" policy means selecting actions that maximize the right hand side of the above Bellman equation. If the MDP is multichain, there is an additional optimality equation that needs to be specified to compute optimal policies, but this is outside the scope of this paper (see [15], [28]). Although this result assures us that stationary deterministic optimal policies exist that achieve the maximum average reward, it does not tell us how to find them. We turn to discuss a variety of algorithms for computing optimal average reward policies.

2.4. Average Reward DP Algorithms

2.4.1. Unichain Policy Iteration

The first algorithm, introduced by Howard [15], is called policy iteration. Policy iteration iterates over two phases: policy evaluation and policy improvement.

1. Initialize $k = 0$ and set π^0 to some arbitrary policy.
2. *Policy Evaluation:* Given a policy π^k, solve the following set of $|S|$ linear equations for the average reward ρ^{π^k} and relative values $V^{\pi^k}(x)$, by setting the value of a reference state $V(s) = 0$.

$$V^{\pi^k}(x) + \rho^{\pi^k} = r(s, \pi^k(x)) + \sum_y P_{xy}(\pi^k(x)) V^{\pi^k}(y).$$

3. *Policy Improvement:* Given a value function V^{π^k}, compute an improved policy π^{k+1} by selecting an action maximizing the following quantity at each state,

$$\max_a \left(r(x,a) + \sum_y P_{xy}(a) V^{\pi^k}(y) \right).$$

   setting, if possible, $\pi^{k+1}(x) = \pi^k(x)$.

4. If $\pi^k(x) \neq \pi^{k+1}(x)$ for some state x, increment k and return to step 2.

$V(s)$ is set to 0 because there are $|S|+1$ unknown variables, but only $|S|$ equations. Howard [15] also provided a policy iteration algorithm for multichain MDP's and proved that both algorithms would converge in finitely many steps to yield a gain-optimal policy. More recently, Haviv and Puterman [13] have developed a more efficient variant of Howard's policy iteration algorithm for communicating MDP's.

The interesting question now is: Does policy iteration produce bias-optimal policies, or only gain-optimal policies? It turns out that policy iteration will find bias-optimal policies if the *first* gain-optimal policy it finds has the same set of recurrent states as a bias-optimal policy [28]. Essentially, policy iteration first searches for a gain-optimal policy that achieves the highest average reward. Subsequent improvements in the policy can be shown to improve only the relative values, or bias, over the transient states.

An example will clarify this point. Consider the two-state MDP in Figure 2. Let us start with the initial policy π^0, which selects action **a2** in state **A**. The policy evaluation phase results in the relative values $V^{\pi^0}(A) = 11$ and $V^{\pi^0}(B) = 0$ (since **B** is chosen as the reference state), and the average reward $\rho^{\pi^0} = -1$. Clearly, policy iteration has already found a gain-optimal policy. However, in the next step of policy improvement, a better policy π^1 is found, which selects action **a1** in state **A**. Evaluating this policy subsequently reveals that the relative values in state **A** has been improved to $V^{\pi^1}(A) = 12$, with the average reward staying unchanged at $\rho^{\pi^1} = -1$. Policy iteration will converge at this point because no improvement in gain or bias is possible.

However, policy iteration will not always produce bias-optimal policies, as the three-state MDP in Figure 2 shows. Here, if we set the value of state **A** to 0, then both policies evaluate to identical bias values $V(A) = 0$, $V(B) = -1$, and $V(C) = 1$. Thus, if we start with the policy π^0 which selects action **a2** in state **A**, and carry out the policy evaluation and subsequent policy improvement step, no change in policy occurs. But, as we explained above, policy π^0 is only gain-optimal and not bias-optimal. The policy that selects action **a1** in state **A** is better since it is bias-optimal.

2.4.2. Value Iteration

The difficulty with policy iteration is that it requires solving $|S|$ equations at every iteration, which is computationally intractable when $|S|$ is large. Although some shortcuts have been proposed that reduce the computation [28], a more attractive approach is to iteratively solve for the relative values and the average reward. Such algorithms are typically referred to in DP as *value iteration* methods.

Let us denote by $T(V)(x)$ the mapping obtained by applying the right hand side of Bellman's equation:

$$T(V)(x) = \max_a \left(r(x,a) + \sum_y P_{xy}(a) V(y) \right).$$

It is well known (e.g. see [5]) that T is a *monotone* mapping, i.e. given two value functions $V(x)$ and $V'(x)$, where $V(x) \leq V'(x)$ over all states x, it follows that $T(V)(x) \leq T(V')(x)$. Monotonicity is a crucial property in DP, and it is the basis for showing the convergence of many algorithms [5]. The value iteration algorithm is as follows:

1. Initialize $V^0(t) = 0$ for all states t, and select an $\epsilon > 0$. Set $k = 0$.
2. Set $V^{k+1}(x) = T(V^k)(x)$ over all $x \in S$.
3. If $sp(V^{k+1} - V^k) > \epsilon$, increment k and go to step 2.
4. For each $x \in S$, choose $\pi(x) = a$ to maximize $\left(r(x,a) + \sum_y P_{xy}(a) V^k(y) \right)$.

The stopping criterion in step 3 uses the *span* semi-norm function $sp(f(x)) = max_x(f(x)) - min_x(f(x))$. Note that the value iteration algorithm does not explicitly compute the average reward, but this can be estimated as $V^{n+1}(x) - V^n(x)$ for large n.

The above value iteration algorithm can also be applied to communicating MDP's, where only optimal policies are guaranteed to have state-independent average reward. However, value iteration has the disadvantage that the values $V(x)$ can grow very large, causing numerical instability. A more stable *relative value iteration* algorithm proposed by White [45] subtracts out the value of a reference state at every step from the values of other states. That is, in White's algorithm, step 2 in value iteration is replaced by

$$V^{k+1}(x) = T(V^k)(x) - T(V^k)(s) \quad \forall x \in S$$

where s is some reference state. Note that $V^k(s) = 0$ holds for all time steps k.

Another disadvantage of (relative) value iteration is that it cannot be directly applied to MDP's where states are periodic under some policies (such as the 3-state MDP in Figure 2). Such MDP's can be handled by a modified value iteration procedure proposed by Hordijk and Tijms [14], namely

$$V^{k+1}(x) = \max_a \left(r(x,a) + \alpha_k \sum_y P_{xy}(a) V^k(y) \right),$$

where $\alpha_k \to 1$ as $k \to \infty$. Alternatively, periodic MDP's can be transformed using an aperiodicity transformation [5], [28] and then solved using value iteration.

Value iteration can be shown to converge to produce an ϵ-optimal policy in finitely many iterations, but the conditions are stricter than for policy iteration. In particular, it will converge if the MDP is aperiodic under all policies, or if there exists a state $s \in S$ that is reachable from every other state under all stationary policies. Like policy iteration, value iteration finds the bias-optimal policy for the 2 state MDP in Figure 2. Unfortunately, like policy iteration, value iteration cannot discriminate between the bias-optimal and gain-optimal policies in the 3-state example MDP.

2.4.3. Asynchronous Value Iteration

Policy iteration and value iteration are both *synchronous*, meaning that they operate by conducting an exhaustive sweep over the whole state space at each step. When updating the relative value of a state, only the old values of the other states are used. In contrast, RL methods are *asynchronous* because states are updated at random intervals, depending on where the agent happens to be. A general model of asynchronous DP has been studied by Bertsekas [4]. Essentially, we can imagine one processor for every state, which keeps track of the relative value of the state. Each processor can be in one of three states: idle, compute the next iterate, or broadcast its value to other processors. It turns out that the asynchronous version of value iteration for discounted MDP's converges under very general protocols for communication and computation.

A natural question, therefore, is whether the natural asynchronous version of value iteration and relative value iteration will similarly converge for the average reward framework. Tsitsiklis provided a counter-example (described in Bertsekas [4]) to show that, quite surprisingly, asynchronous relative value iteration diverges. We discuss this example in some detail because it calls into question whether provably convergent asynchronous algorithms can exist at all for the average reward framework. As we show below, such provably convergent algorithms do exist, and they differ from asynchronous relative value iteration in a crucial detail.

Figure 6 shows a simple two-state MDP where there are no action choices, and hence the problem is to evaluate the single policy to determine the relative values. For this particular MDP, the mapping T can be written as

$$T(V)(0) = g0 + p0V(1) + (1 - p0)V(0)$$
$$T(V)(1) = g1 + p1V(1) + (1 - p1)V(0).$$

If we let state 0 be the reference state, White's algorithm can be written as

$$T(V^{k+1})(0) = g0 + p0(T(V^k)(1) - T(V^k)(0))$$

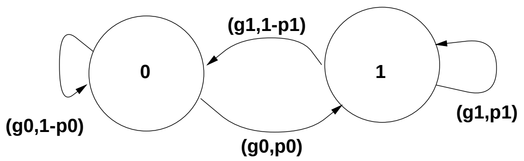

Figure 6. A simple MDP to illustrate that the asynchronous version of White's relative value iteration algorithm diverges, because the underlying mapping is non-monotonic. Both p0 and p1 can lie anywhere in the interval (0,1). g0 and g1 are arbitrary expected rewards. Asynchronous value iteration can be made to converge, however, if it is implemented by a monotonic mapping.

$$T(V^{k+1})(1) = g1 + p1(T(V^k)(1) - T(V^k)(0)).$$

Now, it can be shown that this mapping has a fixpoint V^* such that $V^*(0)$ yields the optimal average reward. However, it is easy to show that this mapping is not monotonic! Thus, although the above iteration will converge when it is solved synchronously, divergence occurs even if we update the values in a Gauss-Seidel fashion, that is use the just computed value of $T(V^k(0))$ when computing for $T(V^k(1))$! Note that the asynchronous algorithm diverges, not only because the mapping is not monotonic, but also because it is not a *contraction* mapping with respect to the maximum norm.[5] For a more detailed analysis, see [4]. There are many cases when divergence can occur, but we have experimentally confirmed that divergence occurs if $g0 = g1 = 1$ and $p0 = p1 > \frac{1}{\sqrt{2}}$.

However, this counter-example should not be viewed as ruling out provably convergent asynchronous average reward algorithms. The key is to ensure that the underlying mapping remains monotonic or a contraction with respect to the maximum norm. Jalali and Ferguson [17] describe an asynchronous value iteration algorithm that provably converges to produce a gain-optimal policy if there is a state $s \in S$ that is reachable from every other state under all stationary policies. The mapping underlying their algorithm is

$$\begin{aligned} V^{t+1}(x) = T(V^t)(x) - \rho^t \quad & \forall x \neq s \\ V^t(s) = 0 \quad & \forall t. \end{aligned}$$

where ρ^t, the estimate of the average reward at time t, is *independently estimated* without using the relative values. Note that the mapping is monotonic since T is monotonic and ρ^t is a scalar value.

There are several ways of independently estimating average reward. We discuss below two provably convergent adaptive algorithms, where the average reward is estimated by online averaging over sample rewards. This similarity is critical to the convergence of the two asynchronous algorithms below, and since R-learning shares this similarity, it suggests that a convergence proof for R-learning is possible.

2.5. An Asynchronous Adaptive Control Method

One fundamental limitation of the above algorithms is that they require complete knowledge of the state transition matrices $P(a)$, for each action a, and the expected payoffs $r(x,a)$ for each action a and state x. This limitation can be partially overcome by inferring the matrices from sample transitions. This approach has been studied very thoroughly in the adaptive control literature. We focus on one particular algorithm by Jalali and Ferguson [16]. They describe two algorithms (named A and B), which both assume that the underlying MDP can be identified by a maximum likelihood estimator (MLE), and differ only in how the average reward is estimated. We will focus mainly on Algorithm B, since that is most relevant to this paper. Algorithm B, which is provably convergent under the assumption that the MDP is ergodic under all stationary policies, is as follows:

1. At time step $t = 0$, initialize the current state to some state x, cumulative reward to $K^0 = 0$, relative values $V^0(x) = 0$ for all states x, and average reward $\rho^0 = 0$. Fix $V^t(s) = 0$ for some reference state s for all t. The expected rewards $r(x,a)$ are assumed to be known.

2. Choose an action a that maximizes $\left(r(x,a) + \sum_y P^t_{xy}(a)V^t(y)\right)$.

3. Update the relative value $V^{t+1}(x) = T(V^t)(x) - \rho^t$, if $x \neq s$.

4. Update the average reward $\rho^{t+1} = \frac{K^{t+1}}{t+1}$, where $K^{t+1} = K^t + r(x,a)$.

5. Carry out action a, and let the resulting state be z. Update the probability transition matrix entry $P^t_{xz}(a)$ using a maximum-likelihood estimator. Increment t, set the current state x to z, and go to step 2.

This algorithm is asynchronous because relative values of states are updated only when they are visited. As discussed above, a key reason for the convergence of this algorithm is that the mapping underlying the algorithm is monotonic. However, some additional assumptions are also necessary to guarantee convergence, such as that the MDP is *identifiable* by an MLE-estimator (for details of the proof, see [16]).

Tadepalli and Ok [38] have undertaken a detailed study of a variant of this algorithm, which includes two modifications. The first is to estimate the expected rewards $r(x,a)$ from sample rewards. The second is to occasionally take random actions in step 2 (this strategy is called *semi-uniform* exploration – see Section 3.3). This modification allows the algorithm to be applied to unichain MDP's (assuming of course that we also break up a learning run into a sequence of trials, and start in a random state in each trial). In our experiments, which have been confirmed by Tadepalli [37], we found that the modified algorithm cannot discriminate between the gain-optimal and bias-optimal policy for the 3-state MDP in Figure 2.

2.6. An Asynchronous Learning Automata Method

Thus far, all the algorithms we have discussed are *model-based*; that is, they involve transition probability matrices (which are either known or estimated). *Model-free* methods eliminate this requirement, and can learn optimal policies directly without the need for a model. We now discuss a provably convergent model-free algorithm by Wheeler and Narendra [44] that learns an optimal average reward policy for any ergodic MDP with unknown state transition probabilities and payoff functions. Thus, this algorithm tackles the same learning problem addressed by R-learning, but as we will see below, is quite different from most RL algorithms.

We need to first briefly describe the framework underlying learning automata [27]. Unlike the previous algorithms, most learning automata algorithms work with randomized policies. In particular, actions are chosen using a vector of probabilities. The probability vector is updated using a learning algorithm. Although many different algorithms have been studied, we will focus on one particular algorithm called *linear reward-inaction* or L_{R-I} for short. Let us assume that there are n actions $a_1, \ldots, a_n$. Let the current probability vector be $p = (p_1, \ldots, p_n)$, where we of course require that $\sum_{i=1}^{n} p_i = 1$. The automata carries out an action a according to the probability distribution p, and receives a response from the environment $0 \leq \beta \leq 1$. The probability vector is updated as follows:

1. If action a is carried out, then $p_a^{t+1} = p_a^t + \alpha\beta^t(1 - p_a^t)$.

2. For all actions $b \neq a$, $p_b^{t+1} = p_b^t - \alpha\beta^t p_b^t$.

Here α is a learning rate in the interval $(0,1)$. Thus, the probability of doing the chosen action is updated in proportion to the probability of not doing the action, weighted by the environmental response β^t at time step t. So far, we have not considered the influence of state. Let us now assume that for each state an independent copy of the L_{R-I} algorithm exists. So for each state, a different probability vector is maintained, which is used to select the best action. Now the key (and surprising) step in the Wheeler-Narendra algorithm is in computing the environmental response β. In particular, the procedure ignores the immediate reward received, and instead computes the average reward over repeated visits to a state under a particular action. The following global and local statistics need to be maintained.

1. Let c^t be the cumulative reward at time step t, and let the current state be s.

2. The local learning automaton in state s uses c^t as well as the global time t to update the following local statistics: $\delta_r^s(a)$, the incremental reward received since state s was last exited under action a, and $\delta_t^s(a)$ the elapsed time since state s was last exited under action a. These incremental values are used to compute the cumulative statistics, that is $r^s(a) \leftarrow r^s(a) + \delta_r^s(a)$, and $t^s(a) \leftarrow t^s(a) + \delta_t^s(a)$.

3. Finally, the learning automaton updates the action probabilities with the L_{R-I} algorithm using as environmental response the estimated average reward $r^s(a)/t^s(a)$.

Wheeler and Narendra prove that this algorithm converges to an optimal average reward policy with probability arbitrarily close to 1 (i.e. w.p. $1-\epsilon$, where ϵ can be made as small as desired). The details of the proof are in [44], which uses some interesting ideas from game theory, such as Nash equilibria. Unfortunately, this restriction to ergodic MDP's means it cannot handle either of the example MDP's in Figure 2. Also, for convergence to be guaranteed, the algorithm requires using a very small learning rate α which makes it converge very slowly.

2.7. R-learning: A Model-Free Average Reward RL Method

Schwartz [31] proposed an average-reward RL technique called R-learning. Like its counterpart, Q-learning [43] (see page 194 for a description of Q-learning), R-learning uses the *action value* representation. The action value $R^\pi(x,a)$ represents the average adjusted value of doing an action a in state x once, and then following policy π subsequently. That is,

$$R^\pi(x,a) = r(x,a) - \rho^\pi + \sum_y P_{xy}(a)V^\pi(y),$$

where $V^\pi(y) = \max_{a\in A} R^\pi(y,a)$, and ρ^π is the average reward of policy π. R-learning consists of the following steps:

1. Let time step $t=0$. Initialize all the $R_t(x,a)$ values (say to 0). Let the current state be x.

2. Choose the action a that has the highest $R_t(x,a)$ value with some probability, else let a be a random exploratory action.

3. Carry out action a. Let the next state be y, and the reward be $r_{imm}(x,y)$. Update the R values and the average reward ρ using the following rules:

$$R_{t+1}(x,a) \leftarrow R_t(x,a)(1-\beta) + \beta(r_{imm}(x,y) - \rho_t + \max_{a\in A} R_t(y,a))$$
$$\rho_{t+1} \leftarrow \rho_t(1-\alpha) + \alpha[r_{imm}(x,a) + \max_{a\in A} R_t(y,a) - \max_{a\in A} R_t(x,a)].$$

4. Set the current state to y and go to step 2.

Here $0\le\beta\le 1$ is the learning rate controlling how quickly errors in the estimated action values are corrected, and $0\le\alpha\le 1$ is the learning rate for updating ρ. One key point is that ρ is updated only when a non-exploratory action is performed.

Singh [33] proposed some variations on the basic R-learning method, such as estimating average reward as the sample mean of the actual rewards, updating the average reward on every step, and finally, grounding a reference $R(x,a)$ value to 0. We have not as yet conducted any systematic experiments to test the effectiveness of these modifications.

Table 1. Summary of the average reward algorithms described in this paper. Only the convergence conditions for gain optimality are shown.

ALGORITHM	GAIN OPTIMALITY
Unichain Policy Iteration (Section 2.4.1) [15]	MDP is unichain
(Relative) Value Iteration (Section 2.4.2) [28], [45]	MDP is communicating
Asynchronous Relative Value Iteration (Section 2.4.3) [4]	Does not converge
Asynchronous Value Iteration (Section 2.4.3) [17] with Online Gain Estimation	A state s is reachable under every policy
Asynchronous Adaptive Control (Section 2.5) [16] with Online Gain Estimation	MDP is ergodic and MLE-Identifiable
Asynchronous Learning Automata (Section 2.6) [44]	MDP is ergodic
R-Learning (Section 2.7) [31]	Convergence unknown

2.7.1. R-learning on Sample MDP Problems

We found that in the 2-state problem, R-learning reliably learns the bias-optimal policy of selecting action **a1** in state **A**. However, in the 3-state problem, like all the preceding algorithms, it is unable to differentiate the bias-optimal policy from the gain-optimal policy.

Baird [1] has shown that there exists a fixpoint of the $R(x,a)$ values for the 3-state MDP. Consider the following assignment of $R(x,a)$ values for the 3-state MDP in Figure 2. Assume $R(A,a1) = 100$, $R(A,a2) = 100$, $R(B,a1) = 99$, and $R(C,a1) = 101$. Also, assume that $\rho = 1$. These values satisfy the definition of $R(x,a)$ for all states and actions in the problem, but the resulting greedy policy does not discriminate between the two actions in state **A**. Thus, this is a clear counter-example that demonstrates that R-learning can converge to a policy with *sub-optimal* bias for unichain MDP's.[6]

2.7.2. Convergence Proof for R-learning

Of course, the question of whether R-learning can be guaranteed to produce gain-optimal policies remains unresolved. From the above discussion, it is clear that to show convergence, we need to determine when the mapping underlying R-learning satisfies some key properties, such as monotonicity and contraction. We are currently developing such a proof, which will also exploit the existing convergence results underlying Jalali and Ferguson's B algorithm (see Section 2.5) and Q-learning (for the undiscounted case).

2.8. Summary of Average Reward Methods

Table 1 summarizes the average reward algorithms described in this section, and lists the known convergence properties. While several convergent synchronous and asynchronous algorithms for producing gain-optimal policies exist, none of them is guaranteed to find bias-optimal policies. In fact, they all fail on the 3-state MDP in Figure 2 for the same reason, namely that bias optimality really requires solving an additional optimality equation [28]. An important problem for future research is to design RL algorithms that can yield bias-optimal policies. We discuss this issue in more detail in Section 5.

3. Experimental Results

In this section we present an experimental study of R-learning. We use two empirical testbeds, a stochastic grid-world domain with one-way membranes, and a simulated robot environment. The grid-world task involves reaching a specific goal state, while the robot task does not have any specific goal states. We use an idealized example to show how R-learning can get into limit cycles given insufficient exploration, and illustrate where such limit-cycle situations occur in the grid-world domain and the robot domain. We show, however, that provided sufficient exploration is carried out, R-learning can perform better than Q-learning in both these testbeds. Finally, we present a detailed sensitivity analysis of R-learning using the grid world domain.

We first presented the limit cycle behavior of R-learning in an earlier paper [23]. However, that paper mainly reported negative results for R-learning using the robot domain. This paper contains several new experimental results, including positive results where R-learning outperforms Q-learning, and a detailed sensitivity analysis of R-learning with respect to exploration and learning rate decay. Since we will not repeat our earlier experimental results, the interested reader is referred to our earlier paper for additional results on R-learning for the robot domain.

3.1. A Grid World Domain with One-Way Membranes

Figure 7 illustrates a grid-world environment, similar to that used in many previous RL systems [18], [32], [35]. Although such grid-world tasks are somewhat simplistic "toy" domains, they are quite useful for conducting controlled experimental tests of RL algorithms. The domain parameters can be easily varied, allowing a detailed sensitivity analysis of any RL algorithm.

The agent has to learn a policy that will move it from any initial location to the goal location (marked by the black square in the figure). The starting state can thus be any location. At each step, the agent can move to any square adjacent (row-wise or column-wise) to its current location. We also include "one way membranes", which allow the agent to move in one direction but not in the other (the agent can move from the lighter side of the wall to the darker side). The membrane wall is shown in the figure as an "inverted cup" shape. If the agent "enters" the cup, it cannot reach the goal by going up, but has to go down to leave the membrane. Thus, the membrane serves as a sort of local maxima. The environment is made stochastic

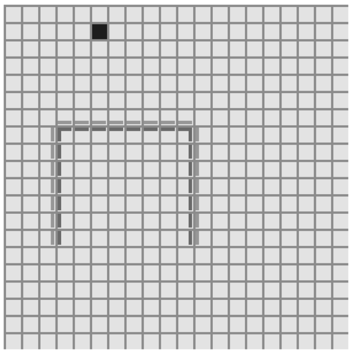

Figure 7. A grid world environment with "one-way" membranes.

by adding a controlled degree of randomness to every transition. In particular, the agent moves to the correct square with probability p, and either stays at its current position or moves to an incorrect adjacent square with probability $(1-p)/N_a$, where N_a is the number of adjacent squares. Thus, if $p = 0.75$ and the robot is in a square with 4 adjacent squares, and it decides to move "up", then it may stay where it currently is, or move "left" or "down" or "right" with equal probability 0.25/4.

The agent receives a reward of +100 for reaching the goal, and a reward of +1 for traversing a membrane. The default reward is -1. Upon reaching the goal, the agent is "transported" to a random starting location to commence another trial. Thus, although this task involves reaching a particular goal state, the average reward obtained during a learning run does not really reflect the true average reward that would result if the goal were made absorbing. In the latter case, since every policy would eventually reach the goal, all policies result in a single recurrent class (namely, the goal state). Since all non-goal states are transient, this task is more closely related to the 2 state example MDP in Figure 2 than the 3 state MDP. This suggests that R-learning should be able to perform quite well on this task, as the experimental results demonstrate.

3.2. A Simulated Robot Environment

We also compare R-learning and Q-learning on a more realistic task, namely avoiding obstacles in a simulated robot environment illustrated in Figure 8. This task provides a nice contrast with the above grid world task because it does not involve reaching any goal states, but instead requires learning a policy that maximizes the average reward.

This testbed provides a good benchmark because we previously studied Q-learning using this environment [25]. The robot is shown as a circular figure; the "nose" indicates the orientation of the robot. Dark objects represent immovable obstacles.

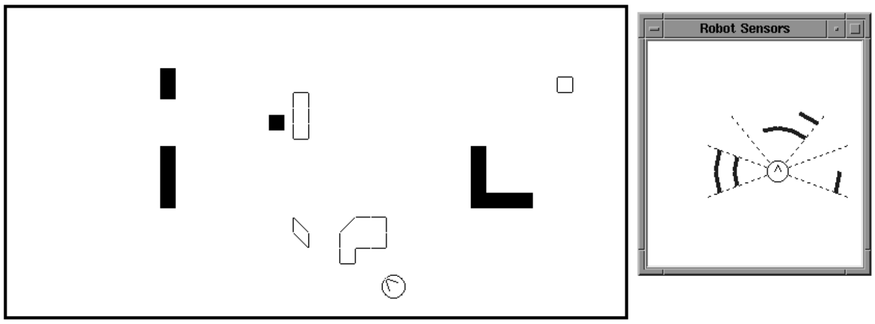

Figure 8. A simulated robot environment modeled after a real robot environment.

Outline figures represent movable boxes. The robot uses eight simulated "sonar" sensors arranged in a ring. For the experiment described below, the sonar data was compressed as follows. The eight sonar values were thresholded to 16 sonar bits, one "near" bit and one "far" bit in each of 8 radial directions. The figure illustrates what the robot actually "sees" at its present location. Dark bars represent the near and far bits that are on. The 16 sonar bits were further reduced to 10 bits by disjoining adjacent near and far bits.[7]

There are also 2 bits of non-sonar information, BUMP and STUCK, which indicate whether the robot is touching something and whether it is wedged. Thus, there are a total of 12 state bits, and 4096 resulting states. The robot moves about the simulator environment by taking one of five discrete actions: forward or turn left and right by either 45 or 90 degrees. The reinforcement function for teaching the robot to avoid obstacles is to reward it by +10 for moving forward to "clear" states (i.e., where the front near sonar bits are off), and punish it by -3 for becoming wedged (i.e., where STUCK is on). This task is very representative of many recurrent robot tasks which have no absorbing (i.e. terminal) goal states.

A learning run is broken down into a sequence of trials. For the obstacle avoidance task, a trial consists of 100 steps. After 100 steps, the environment and the robot are reset, except that at every trial the robot is placed at a random unoccupied location in the environment facing a random orientation. The location of boxes and obstacles does not vary across trials. A learning run lasts for 300 trials. We typically carried out 30 runs to get average performance estimates.

The simulator is a simplification of the real world situation (which we studied previously [25]) in several important respects. First, boxes in the simulator can only move translationally. Thus a box will move without rotation even if a robot pushes the box with a force which is not aligned with the axis of symmetry. Second, the sonars on the real robot are prone to various types of noise, whereas the sensors on a simulator robot are "clean". Finally, the underlying dynamics of robot motion and box movement are deterministic, although given its impoverished sensors, the world does appear stochastic to the robot. Even though this simulator is a fairly inaccurate model of reality, it is sufficiently complex to illustrate the issues raised in this paper.

3.3. Exploration Strategies

RL methods usually require that all actions be tried in all states infinitely often for asymptotic convergence. In practice, this is usually implemented by using an "exploration" method to occasionally take sub-optimal actions. We can divide exploration methods into *undirected* and *directed* methods. Undirected exploration methods do not use the results of learning to guide exploration; they merely select a random action some of the time. Directed exploration methods use the results of learning to decide where to concentrate the exploration efforts. We will be using four exploration methods in our experimental tests of R-learning: two undirected exploration methods, Boltzmann exploration and semi-uniform exploration, and two directed exploration methods, recency-based [35] and uncertainty exploration (UE). A detailed comparison of undirected and directed exploration methods is given in [40].

3.3.1. Semi-Uniform Exploration

Let $p(x, a)$ denote the probability of choosing action a in state x. Let $U(x, a)$ denote a generic state action value, which could be a $Q(x, a)$ value or a $R(x, a)$ value. Denote by a_{best} the action that maximizes $U(x, a)$. Semi-uniform exploration is defined as $p(x, a_{best}) = p_{exp} + \frac{1-p_{exp}}{|A|}$. If $x \neq a_{best}$, $p(x, a) = \frac{1-p_{exp}}{|A|}$. In other words, the best action is chosen with a fixed probability p_{exp}. With probability $1 - p_{exp}$, a random action is carried out.

3.3.2. Boltzmann Exploration

The Boltzmann exploration function [20], [35] assigns the probability of doing an action a in state x as $p(x, a) = \frac{e^{\frac{U(x,a)}{T}}}{\sum_a e^{\frac{U(x,a)}{T}}}$ where T is a "temperature" parameter that controls the degree of randomness. In our experiments, the temperature T was gradually decayed from an initial fixed value using a decaying scheme similar to that described in [3].

3.3.3. Recency-based Exploration

In recency-based exploration [35], the action selected is one that maximizes the quantity $U(x,a) + \epsilon\sqrt{N(x,a)}$, where $N(x,a)$ is a recency counter and represents the last time step when action a was tried in state x. ϵ is a small constant < 1.

3.3.4. UE Exploration

Finally, the second directed exploration strategy is called *uncertainty estimation* (UE). Using this strategy, with a fixed probability p, the agent picks the action a that maximizes $U(x,a) + \frac{c}{N_f(x,a)}$, where c is a constant, and $N_f(x,a)$ represents the number of times that the action a has been tried in state x. With probability $1-p$, the agent picks a random action.

3.4. Limit Cycles in R-learning

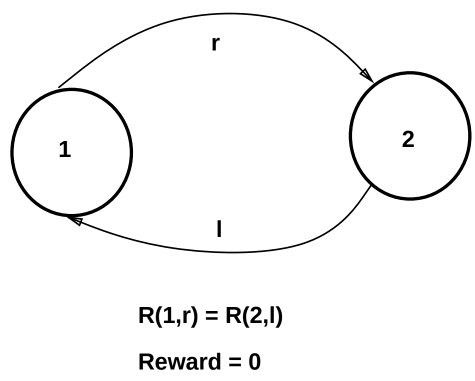

Figure 9. A simple limit cycle situation for comparing R-learning and Q-learning.

A key assumption underlying R-learning (and all the other methods discussed in Section 2) is that the average reward ρ is state independent. In this section, we explore the consequences of this assumption, under a sub-optimal exploration strategy that does not explore the state space sufficiently, creating non-ergodic multichains. Essentially, R-learning and Q-learning behave very differently when an exploration strategy creates a tight limit cycle such as illustrated in Figure 9. In particular, the performance of R-learning can greatly suffer. Later we will show that such limit cycles can easily arise in both the grid world domain and the simulated robot domain, using two different exploration strategies.

Consider the simple situation shown in Figure 9. In state 1, the only action is to go right (marked r), and in situation 2, the only action is to go left (marked l).

Finally, the immediate reward received for going left or right is 0. Such a limit cycle situation can easily result even when there are multiple actions possible, whenever the action values for going left or right are initially much higher than the values for the other actions. Under these conditions, the update equations for R-learning can be simplified to yield [8]

$$R(1,r) \leftarrow R(1,r) - \beta\rho$$
$$R(2,l) \leftarrow R(2,l) - \beta\rho.$$

Thus, $R(1,r)$ and $R(2,l)$ will decay over time. However, the average reward ρ is itself decaying, since under these conditions, the average reward update equation turns out to be

$$\rho \leftarrow (1-\alpha)\rho.$$

When ρ decays to zero, the utilities $R(1,r)$ and $R(2,l)$ will stop decaying. The relative decay rate will of course depend on β and α, but one can easily show cases where the R values will decay much slower than ρ. For example, given the values $R(1,l) = 3.0$, $\rho = 0.5$, $\alpha = 0.1$, $\beta = 0.2$, after 1000 iterations, $R(1,l) = 2$, but $\rho < 10^{-46}$!

Since the $R(x,a)$ values are not changing when ρ has decayed to zero, if the agent uses the Boltzmann exploration strategy and the temperature is sufficiently low (so that the $R(x,a)$ values are primarily used to select actions), it will simply oscillate between going left and going right. We discovered this problem in a simulated robot box-pushing task where the robot had to learn to avoid obstacles [23].

Interestingly, Q-learning will not get into the same limit cycle problem illustrated above because the Q values will always decay by a fixed amount. Under the same conditions as above, the update equation for Q-learning can be written as

$$Q(1,l) \leftarrow Q(1,l)(1-\beta(1-\gamma))$$
$$Q(2,r) \leftarrow Q(2,r)(1-\beta(1-\gamma)).$$

Now the two utilities $Q(1,r)$ and $Q(2,r)$ will continue to decay until at some point another action will be selected because its Q value will be higher.

3.4.1. Robot Task with Boltzmann Exploration

Limit cycles arise in the robot domain when the robot is learning to avoid obstacles. Consider a situation where the robot is stalled against the simulator wall, and is undecided between turning left or right. This situation is very similar to Figure 9 (in fact, we were led to the limit cycle analysis while trying to understand the lackluster performance of R-learning at learning obstacle avoidance). The limit cycles are most noticeable under Boltzmann exploration.

3.4.2. Grid World Task with Counter-based Exploration

We have observed limit cycles in the grid world domain using the UE exploration strategy. Limit cycles arise in the grid world domain at the two ends of the inverted cup membrane shape shown in Figure 7. These limit cycles involve a clockwise or counterclockwise chain of 4 states (the state left of the membrane edge, the state right of the membrane edge, and the two states below these states). Furthermore, the limit cycles arise even though the agent is getting non-zero rewards. Thus, this limit cycle is different from the idealized one shown earlier in Figure 9.

In sum, since R-learning can get into limit cycles in different tasks using different exploration strategies, this behavior is clearly not task specific or dependent on any particular exploration method. The limit cycle behavior of R-learning arises due to the fact that average reward is state independent because the underlying MDP is assumed to be unichain. An exploration method that produces multichain behavior can cause the average reward to be incorrectly estimated (for example, it can drop down to 0). This reasoning suggests that limit cycles can be avoided by using higher degrees of exploration. We show below that this is indeed the case.

3.5. Avoiding Limit Cycles by Increasing Exploration

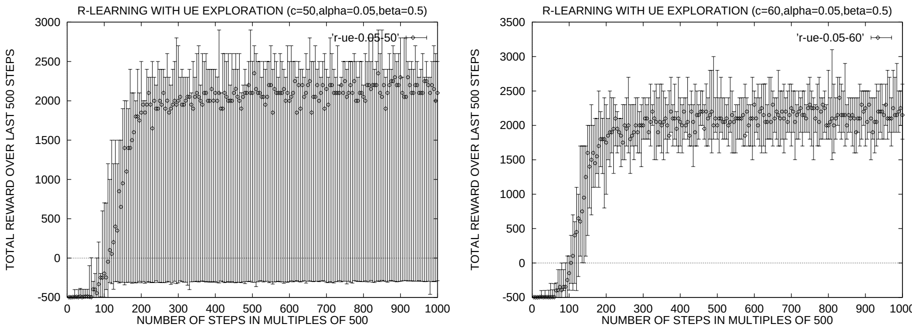

Figure 10. Comparing performance of R-learning on the grid world environment with “one-way” membranes as exploration level is increased.

Although limit cycles can seriously impair the performance of R learning, they can be avoided by choosing a suitable exploration strategy. The key here is to ensure that a sufficient level of exploration is carried out that will not hamper the estimate of the average reward. Figure 10 illustrates this point: here the constant c used in the UE exploration method is increased from 50 to 60. The errorbars indicate the range between the high and low values over 30 independent runs. Note that when $c = 50$, limit cycles arise (indicated by the high variance between low and high values), but disappear under a higher level of exploration ($c = 60$).

Similarly, we have found that limit cycles can be avoided in the robot domain using higher degrees of exploration. Next we demonstrate the improved performance of R-learning under these conditions.

3.6. Comparing R-learning and Q-learning

We now compare the performance of R-learning with Q-learning. We need to state some caveats at the outset, as such a comparison is not without some inherent difficulties. The two techniques depend on a number of different parameters, and also their performance depends on the particular exploration method used. If we optimize them across different exploration methods, then it could be argued that we are really not comparing the algorithms themselves, but the *combination* of the algorithm and the exploration method. On the other hand, using the same exploration method may be detrimental to the performance of one or both algorithms. Ideally, we would like to do both, but the number of parameter choices and alternative exploration methods can create a very large space of possibilities.

Consequently, our aim here will be more modest, and that is to provide a reasonable basis for evaluating the empirical performance of R-learning. We demonstrate that R-learning can outperform Q-learning on the robot obstacle avoidance task, even if we separately optimize the exploration method used for each technique. We also show a similar result for the grid world domain, where both techniques use the same exploration method, but with different parameter choices. We should also mention here that Tadepalli and Ok [38] have found that R-learning outperforms Q-learning on a automated guided vehicle (AGV) task, where both techniques used the semi-uniform exploration method.

3.6.1. Simulated Obstacle Avoidance Task

Figure 11 compares the performance of R-learning with that of Q-learning on the obstacle avoidance task. Here, R-learning uses a semi-uniform exploration strategy, where the robot takes random actions 2.5% of the time, since this seemed to give the best results. Q-learning is using a recency-based exploration strategy, with $\epsilon = 0.001$ which gave the best results. Clearly, R-learning is outperforming Q-learning consistently throughout the run. Note that we are measuring the performance of both algorithms on the basis of average reward, which Q-learning is not specifically designed to optimize. However, this is consistent with many previous studies in RL, which have used average reward to judge the empirical performance of Q-learning [20], [25].

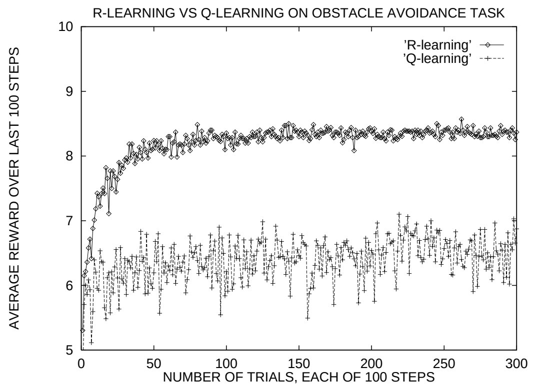

Figure 11. Comparing R and Q-learning using different exploration techniques on a simulated robot obstacle avoidance task. Here Q-learning uses a recency-based exploration method with $\epsilon = 0.001$ which gave the best results. The discount factor $\gamma = 0.99$. R-learning uses a semi-uniform exploration with random actions chosen 2.5% of the time. The curves represent median values over 30 independent runs.

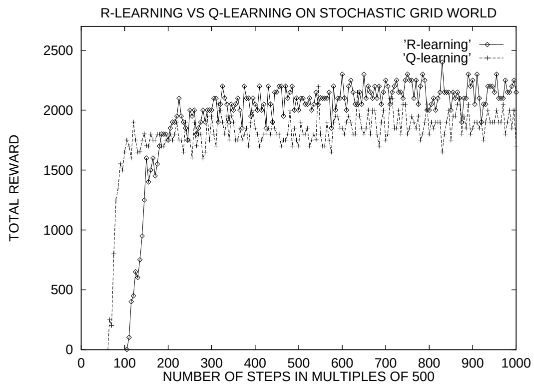

Figure 12. Comparing R vs Q-learning on a grid world environment with membranes. Here both algorithms used the UE exploration method. The UE constant for R-learning is $c = 60$, whereas for Q-learning, $c = 30$. The learning rates were set to $\beta = 0.5$ and $\alpha = 0.05$. The discount factor $\gamma = 0.995$.

3.6.2. Grid World Domain

Figure 12 compares the performance of R and Q-learning on the grid world domain. The exploration strategy used here is the UE counter-based strategy described above, which was separately optimized for each technique. R-learning is clearly outperforming Q-learning. Interestingly, in both domains, R-learning is slower to reach its optimal performance than Q-learning.

3.7. Sensitivity Analysis of R-learning

Clearly, we need to determine the sensitivity of R-learning to exploration to obtain a more complete understanding of its empirical performance. We now describe some sensitivity analysis experiments that illustrate the variation in the performance of R learning with different amounts of exploration. The experiments also illustrate the sensitivity of R-learning to the learning rate parameters α and β.

Figure 13 and Figure 14 illustrate the sensitivity of R-learning to exploration levels and learning rates for the grid world domain. Each figure shows a 3-dimensional plot of the performance of R-learning (as measured by the total cumulative reward) for different values of the two learning rates, α, for adjusting average reward ρ, and β, for adjusting the relative action values $R(x,a)$. The exploration probability parameter p was reduced gradually from an initial value of $p = 0.95$ to a final value of $p = 0.995$ in all the experiments. Each plot measures the performance for different values of the exploration constant c, and a parameter k that controls how quickly the learning rate β is decayed. β is decayed based on the number of updates of a particular $R(x,a)$ value. This state action dependent learning rate was previously used by Barto et al [3]. More precisely, the learning rate β for updating a particular $R(x,a)$ value is calculated as follows:

$$\beta(x,a) = \frac{\beta_0 k}{k + freq(x,a)},$$

where β_0 is the initial value of the β. In these experiments, the learning rate α was also decayed over time using a simpler state independent rule:

$$\alpha_{t+1} \rightarrow \alpha_t - \alpha_t \alpha_{min},$$

where α_{min} is the minimum learning rate required.

Figure 13 and Figure 14 reveal a number of interesting properties of R-learning. The first two properties can be observed by looking at each plot in isolation, whereas the second two can be observed by comparing different plots.

- *More exploration is better than less:* The degree of exploration is controlled by the parameter c. Higher values of c mean more exploration. Each plot shows that higher values of c generally produce better performance than lower values. This behavior is not surprising, given our analysis of limit cycles. Higher exploration means that R-learning will be less likely to fall into limit cycles. However, as we show below, the degree of exploration actually depends on the stochasticity of the domain, and more exploration is not always better than less.

- *Slow decay of β is better than fast decay:* Each plot also shows larger values of the parameter k (which imply that β will be decayed more slowly) produces better performance than small values of k. This behavior is similar to that of Q-learning, in that performance suffers if learning rates are decayed too quickly.

- *Low values of α are better than high values:* Another interesting pattern revealed by comparing different plots is that initializing α to smaller values (such as 0.05) clearly produces better performance as compared to larger initial values (such as 0.5). The reason for this behavior is that higher values of α cause the average reward ρ to be adjusted too frequently, causing wide fluctuations in its value over time. A low initial value of α means that ρ will be changed very slowly over time.

- *High values of β are better than low values:* Finally, comparing different plots reveals that higher initial values of β are to be preferred to lower values. The underlying reason for this behavior is not apparent, but it could be dependent on the particular grid world domain chosen for the study.

3.7.1. Degree of Exploration

The above sensitivity analysis tends to give the impression that more exploration is alway better than less. Figure 15 illustrates how the amount of exploration needed actually depends on the stochasticity of the underlying MDP. Here, the domain is a simplified grid world with no membranes. The goal is to reach the grid cell $(19, 19)$ (the bottom right corner) from any other cell in the grid world. The curves show the result of increasing semi-uniform exploration from no exploration ($p = 0.00$) to a high level of exploration ($p = 0.80$). The graph on the left shows the results for a deterministic grid world domain. Here, performance does improve as exploration is increased, but only up to a point (between $p = 0.20$ and $p = 0.40$). Beyond that, performance suffers. The graph on the right shows the same results for a stochastic grid world domain with transition probability $p = 0.75$. Here, the best performance occurs with no exploration at all, because the underlying stochasticity of the domain ensures that all states will be visited!

4. Conclusions

Based on our overview of average reward RL (Section 2) and the experimental tests on R-learning (Section 3), the main findings of this paper can be summarized as follows.

4.1. Average Reward MDP

We emphasized the distinction between gain-optimal policies that maximize average reward versus bias-optimal policies that also maximize relative values. The key finding is that while there are several synchronous and asynchronous algorithms for producing gain-optimal policies, none of these algorithms (including R-learning) can reliably produce bias-optimal policies for a general unichain MDP.

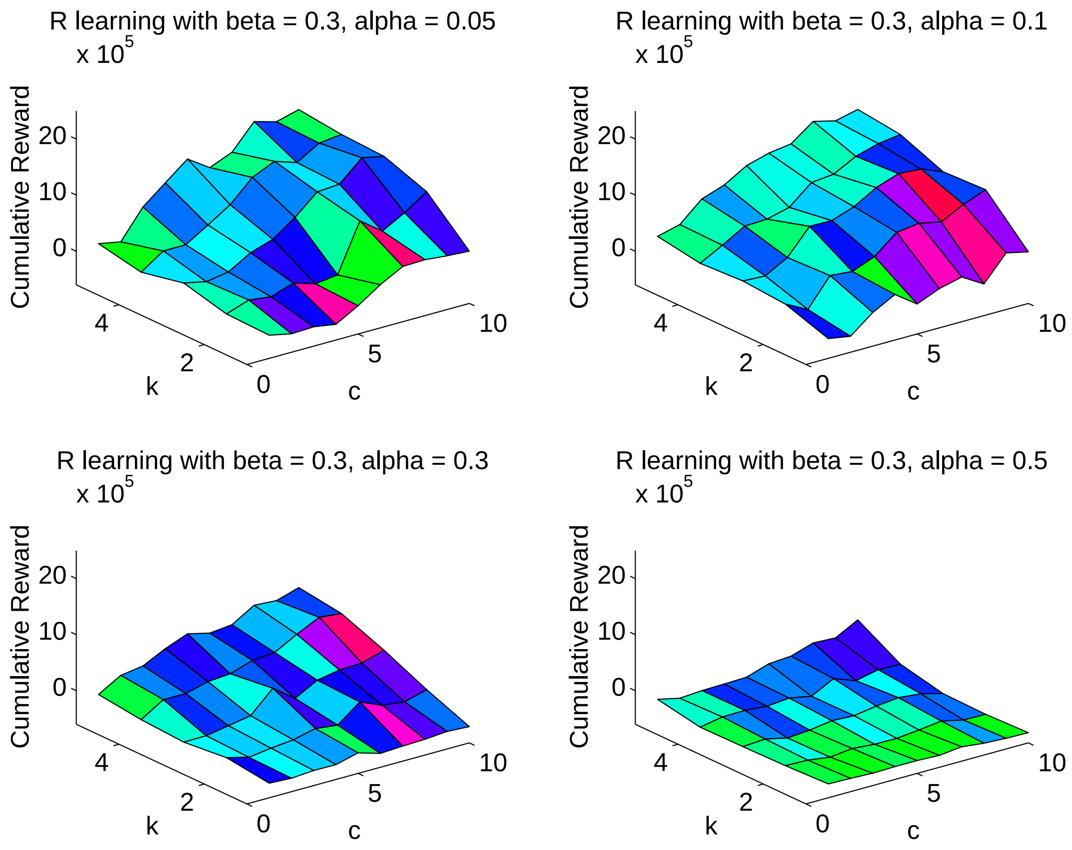

Figure 13. Performance of R-learning on a 20x20 stochastic grid world environment with one-way membranes. Each point represents the median over 30 runs of 500,000 steps. The domain was stochastic with transition probability = 0.75. The x axis represents ten different values of the constant of exploration c varied uniformly over the range [10,100]. Higher values of c mean more exploration. The y axis represents five different values of the constant k controlling the decay of the learning rate β. k was uniformly varied over the range [100,500]. Higher values of k mean more gradual decay of β.

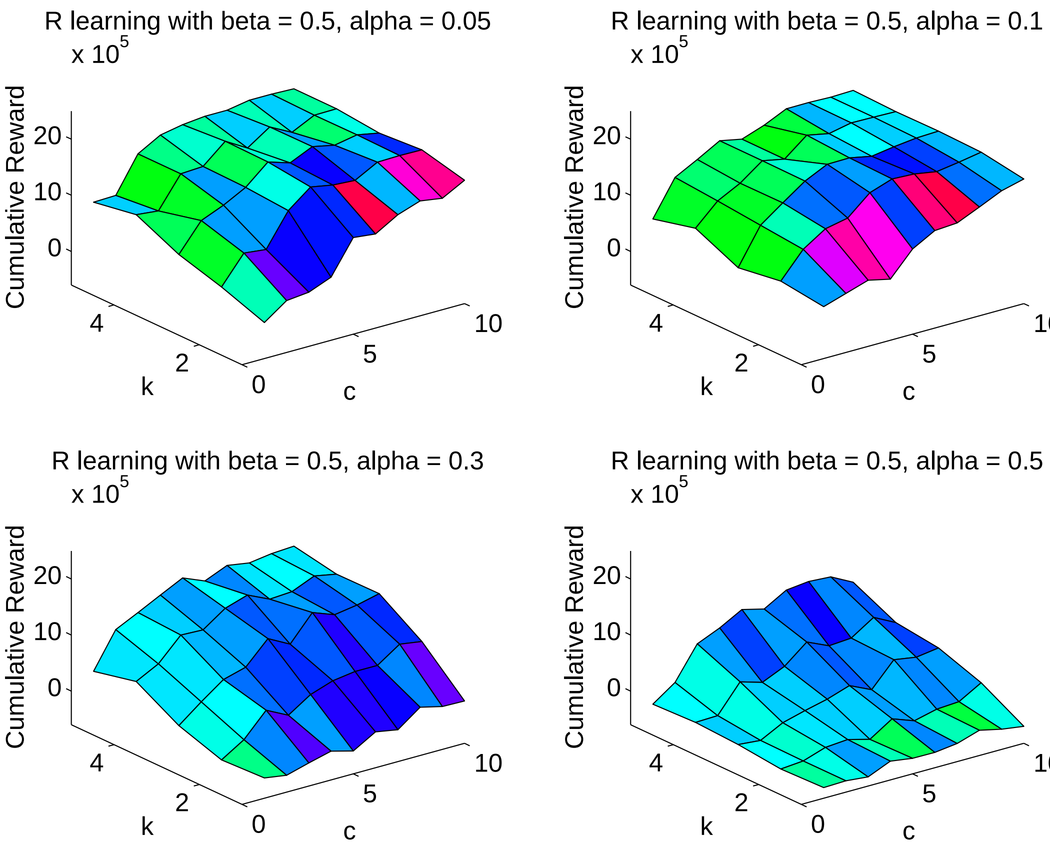

Figure 14. Performance of R-learning on a 20x20 stochastic grid world environment with one-way membranes. Each point represents the median over 30 runs of 500,000 steps. The domain was stochastic with transition probability = 0.75. The x axis plots ten different values of the constant of exploration c varied uniformly over the range [10,100]. Higher values of c mean more exploration. The y axis plots five different values of the constant k controlling the decay of the learning rate β. k was uniformly varied over the range [100,500]. Higher values of k mean more gradual decay of β.

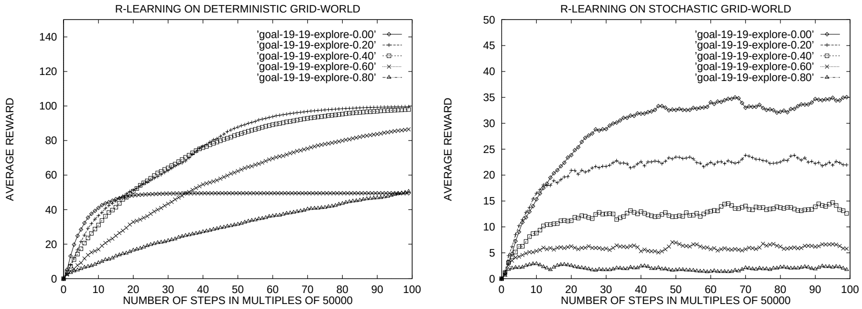

Figure 15. Performance of R-learning for varying amounts of semi-uniform exploration for a deterministic (graph on left) and stochastic (graph on right) grid world domain with no membranes. These graphs show that the best level of exploration depends on the stochasticity of the domain.

We presented a general optimality metric called *n-discount-optimality* [42] relating discounted and undiscounted MDP. Gain optimality and bias optimality correspond to the first two terms of this optimality metric. We believe that this metric offers a general framework to understand the relationship between discounted and undiscounted RL, and also opens up some very interesting directions for future work.

We described several asynchronous algorithms that are provably guaranteed to yield gain-optimal policies. We also pointed out a key difference between these algorithms and the natural asynchronous relative value iteration method, namely that the average reward is estimated independently of the relative values, such as by averaging over sample rewards. Since R-learning shares this similarity, it brightens the prospect of proving it converges to yield gain-optimal policies.

4.2. Experimental Results on R-learning

We now discuss two key conclusions that are based on our experimental results. However, we must emphasize the tentative nature of these conclusions at this point. We are planning to undertake a more detailed study of a residual gradient [2] version of R-learning using larger state space problems, which will provide a further test of these hypotheses [24].

Our first observation is that average reward methods are more sensitive to exploration than discounted methods. We showed that R learning can fall into limit cycles in two domains using two different exploration methods. We also showed that by increasing the degree of exploration, R-learning does not get into limit cycles. Finally, we presented a detailed sensitivity study of R-learning by varying the degree of exploration over different learning rates.

Our second observation is that average reward methods can be superior to discounted methods. We showed that R-learning can perform better than Q-learning in two domains, a simulated robot task and a grid world task, even if the two methods were separately optimized. More detailed tests are needed to determine if there are some generic types of domains (for example, ergodic domains) that average reward methods, such as R-learning, are inherently better at than discounted methods such as Q-learning.

Recent work by Tadepalli and Ok [38] also supports both these conclusions. They compared a *model-based* average reward method called H learning (which is a variant of Jalali and Ferguson's B algorithm [16] described in Section 2.5) with R learning, Q-learning, and ARTDP [3], using an automated guided vehicle (AGV) task. In their experiments, they primarily used semi-uniform exploration. They found that H learning performs much better as the level of exploration is increased. They also found that in some cases the two average reward methods, namely H learning and R learning, outperformed the two discounted methods, namely ARTDP and Q learning.

5. Directions for Future Research

This paper suggests many promising directions for future research in average reward RL, some of which are already bearing fruit.

- *Bias-Optimal RL Algorithms:* Clearly, one of the most pressing open problems is developing a bias-optimal RL algorithm. The key difference between the algorithms described in this paper and a bias-optimal algorithm is that the latter requires solving two nested optimality equations [28]. Several approaches to the design of a bias-optimal algorithm have been pursued in the DP literature, ranging from policy iteration [42], [28], linear programming [9], and value iteration [12]. We are currently testing a model-based bias optimality algorithm, which extends Jalali and Ferguson's B algorithm [21]. We are also working on a model-free bias optimality algorithm, which extends R-learning. We expect that bias-optimal RL algorithms will scale better than bias-optimal DP algorithms.

- *Value Function Approximation in Average Reward MDP:* In this paper, we used a simple table lookup representation for representing policies. In most realistic applications of RL, such as robotics [25] or games [39], the size of the state space rules out table-lookup representations, and necessitates using a function approximator, such as a neural net [20], [39] or kd-trees [26], [30]. We have recently developed some theoretical results showing the performance loss that results from using approximate value functions [24]. We are also working on a residual gradient [2] version of R-learning.

- *Multi-Chain Average Reward Algorithms:* In this paper we focused mainly on algorithms for unichain MDP's. This allows representing the average reward ρ by a scalar value. However, this assumption may be violated in many applications. An interesting problem is to develop RL algorithms to handle multi-chain MDP's, where different states may have different average rewards. In multichain MDP's, an additional optimality equation relating the gain of one state to other states has to be solved.

- *Modular Average Reward Methods:* A number of researchers have studied modular architectures for scaling RL [8], [20], [22], [25], [32], [46]. Such architectures decompose tasks into different sets of primitive subtasks. Learning primitive subtasks is easier because rewards are more frequent. Also, the solutions learned to subtasks can be transferred to yield faster task learning. All these systems have been based on Q-learning. However, an average reward method such as R-learning has significant advantages over Q-learning for multi-task learning. Average adjusted values satisfy a nice linearity property [31] (assuming a deterministic MDP): $V_y^\pi - V_x^\pi = \rho - r$. In other words, if rewards are constant, the difference in relative values across states is also constant. We have empirically found this linearity property to hold up well even in stochastic MDP's. Discounting, on the other hand, causes exponential non-linearities across states. We plan to develop an average-reward based modular architecture for multi-task learning.

Acknowledgements

This paper could not have been completed without the help of the following people, to whom I am indebted. Ronny Ashar implemented R-learning on the stochastic grid world domain. Leemon Baird provided several counter-examples for R-learning. I am especially grateful to the special issue editor, Leslie Kaelbling, for encouraging me to discuss average reward MDP. I thank Prasad Tadepalli for many discussions on average reward reinforcement learning, and for his detailed comments on this paper. This research was supported, in part, by the University of South Florida Research and Creative Scholarship Grant No. 21-08-936-RO, and by a National Science Foundation (NSF) Career Award Grant No. IRI-9501852.

Appendix: Discounted Reinforcement Learning

Most previous work in reinforcement learning has studied a formulation where agents maximize the *discounted* cumulative sum of rewards [36]. The discounted return of a policy π starting from a state x is defined as

$$V_\gamma^\pi(s) = \lim_{N\to\infty} E(\sum_{t=0}^{N-1} \gamma^t R_t^\pi(s)),$$

where $\gamma \leq 1$ is the discount factor, and $R_t^\pi(s)$ is the reward received at time step t starting from state s and choosing actions using policy π. An *optimal discounted policy* π^* maximizes the above value function over all states x and policies π, i.e. $V_\gamma^{\pi^*}(x) \geq V_\gamma^\pi(x)$.

The action value $Q^{\pi}_{\gamma}(x,a)$ denotes the discounted return obtained by performing action a once from state x, and thereafter following policy π

$$Q^{\pi}_{\gamma}(x,a) = r(x,a) + \gamma \sum_{y} P_{xy}(a) V^{\pi}_{\gamma}(y),$$

where $V^{\pi}_{\gamma}(y) = max_{a \in A} Q(y,a)$, and $r(x,a)$ is the expected reward for doing action a in state x.

5.0.1. Q-learning:

Watkins [43] proposed a simple iterative method for learning Q values. All the $Q(x,a)$ values are randomly initialized to some value (say 0). At time step t, the learner either chooses the action a with the maximum $Q_t(x,a)$ value, or with some probability selects a random "exploratory" action to ensure that it does not get stuck in a local maximum. If the agent moves from state x to state y, and receives an immediate reward $r_{imm}(x,a)$, the current $Q_t(x,a)$ values are updated using the rule:

$$Q_{t+1}(x,a) \leftarrow Q_t(x,a)(1-\beta) + \beta \left(r_{imm}(x,y) + \gamma \max_{a \in A} Q_t(y,a) \right) \tag{2}$$

where $0 \leq \beta \leq 1$ is the learning rate controlling how quickly errors in action values are corrected. Q-learning is guaranteed to asymptotically converge to the optimal discounted policy for a finite MDP. The precise convergence conditions are given in [41], but essentially every action must be tried in every state infinitely often, and the learning rate β must be slowly decayed to 0. Q-learning also converges when $\gamma = 1$, if we assume a zero reward absorbing state s which is reachable under all policies, and $Q_t(s,a) = 0$ for all actions a and time t.

Notes

1. The reason both techniques converge to a value slightly below 2 is because the robot takes random actions 5% of the time.
2. If the state space is not finite, we have to allow history dependent policies, since there may be no gain-optimal stationary policy. See [5], [28], [29] for some examples.
3. This limit is guaranteed to exist as long as the state space is countable, and we do not allow history dependent policies [28]. For more general policies, this limit need not exist, and two possibly different measures of average reward result from using lim sup and lim inf, respectively.
4. This limit assumes that all policies are aperiodic. For periodic policies, we need to use the Cesaro limit $V^{\pi}(s) = \lim_{N \to \infty} \frac{\sum_{k=0}^{N-1} E\left(\sum_{t=0}^{k} (R^{\pi}_t(s) - \rho^{\pi})\right)}{N}$.
5. A mapping T is a contraction mapping (w.r.t. the maximum norm) if and only if there exists a real number $0 < \delta < 1$ such that $\|T(V)(x) - T(V')(x)\| \leq \delta \|V(x) - V'(x)\|$, where $\|f(x)\| = \max_x |f(x)|$, and $V(x)$ and $V'(x)$ are any two real-valued bounded functions on S.
6. This counter-example implies that Schwartz's remark in his paper [31] – "when R-learning converges, it must produce a T-optimal policy" – is not correct for unichain MDP's.

7. Drastically compressing sensor information can easily make robot tasks non-Markovian. However, note here that the robot does not have to discriminate between boxes and non-boxes, but only avoid hitting anything and keep moving forward. Using all the sensor data would require using function approximation, which could itself create a non-Markovian problem.
8. To simplify the argument, we are assuming synchronous updating, but the results hold even for asynchronous updating.

References

1. L. Baird. Personal Communication.
2. L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.
3. A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.
4. D. Bertsekas. Distributed dynamic programming. *IEEE Transactions on Automatic Control*, AC-27(3), June 1982.
5. D. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, 1987.
6. D. Blackwell. Discrete dynamic programming. *Annals of Mathematical Statistics*, 33:719–726, 1962.
7. C. Boutilier and M. Puterman. Process-oriented planning and average-reward optimality. In *Proceedings of the Fourteenth JCAI*, pages 1096–1103. Morgan Kaufmann, 1995.
8. P. Dayan and G Hinton. Feudal reinforcement learning. In *Neural Information Processing Systems (NIPS)*, pages 271–278, 1992.
9. E. Denardo. Computing a bias-optimal policy in a discrete-time Markov decision problem. *Operations Research*, 18:272–289, 1970.
10. L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski. A personal learning apprentice. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)*, pages 96–103. MIT Press, 1992.
11. J. Engelberger. *Robotics in Service*. MIT Press, 1989.
12. A. Federgruen and P.. Schweitzer. Successive approximation methods for solving nested functional equations in Markov decision problems. *Mathematics of Operations Research*, 9:319–344, 1984.
13. M. Haviv and M. Puterman. An improved algorithm for solving communicating average reward markov decision processes. *Annals of Operations Research*, 28:229–242, 1991.
14. A. Hordijk and H. Tijms. A modified form of the iterative method of dynamic programming. *Annals of Statistics*, 3:203–208, 1975.
15. R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
16. A. Jalali and M. Ferguson. Computationally efficient adaptive control algorithms for Markov chains. In *Proceedings of the 28th IEEE Conference on Decision and Control*, pages 1283–1288, 1989.
17. A. Jalali and M. Ferguson. A distributed asynchronous algorithm for expected average cost dynamic programming. In *Proceedings of the 29th IEEE Conference on Decision and Control*, pages 1394–1395, 1990.
18. L. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173. Morgan Kaufmann, 1993.
19. L. Kaelbling. *Learning in Embedded Systems*. MIT Press., 1993.
20. L. Lin. *Reinforcement Learning for Robots using Neural Networks*. PhD thesis, Carnegie-Mellon Univ, 1993.
21. S. Mahadevan. A model-based bias-optimal reinforcement learning algorithm. In preparation.

22. S. Mahadevan. Enhancing transfer in reinforcement learning by building stochastic models of robot actions. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 290–299. Morgan Kaufmann, 1992.
23. S. Mahadevan. To discount or not to discount in reinforcement learning: A case study comparing R-learning and Q-learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 164–172. Morgan Kaufmann, 1994.
24. S. Mahadevan and L. Baird. Value function approximation in average reward reinforcement learning. In preparation.
25. S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992. Appeared originally as IBM TR RC16359, Dec 1990.
26. A. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state spaces. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 333–337. Morgan Kaufmann, 1991.
27. K. Narendra and M. Thathachar. *Learning Automata: An Introduction*. Prentice Hall, 1989.
28. M. Puterman. *Markov Decision Processes: Discrete Dynamic Stochastic Programming*. John Wiley, 1994.
29. S. Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, 1983.
30. M. Salganicoff. Density-adaptive learning and forgetting. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 276–283. Morgan Kaufmann, 1993.
31. A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 298–305. Morgan Kaufmann, 1993.
32. S. Singh. *Learning to Solve Markovian Decision Processes*. PhD thesis, Univ of Massachusetts, Amherst, 1994.
33. S. Singh. Reinforcement learning algorithms for average-payoff Markovian decision processes. In *Proceedings of the 12th AAAI*. MIT Press, 1994.
34. R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.
35. R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.
36. R. Sutton, editor. *Reinforcement Learning*. Kluwer Academic Press, 1992. Special Issue of Machine Learning Journal Vol 8, Nos 3-4, May 1992.
37. P. Tadepall. Personal Communication.
38. P. Tadepalli and D. Ok. H learning: A reinforcement learning method to optimize undiscounted average reward. Technical Report 94-30-01, Oregon State Univ., 1994.
39. G. Tesauro. Practical issues in temporal difference learning. In R. Sutton, editor, *Reinforcement Learning*. Kluwer Academic Publishers, 1992.
40. S. Thrun. The role of exploration in learning control. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold.
41. J. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1994.
42. A. Veinott. Discrete dynamic programming with sensitive discount optimality criteria. *Annals of Mathematical Statistics*, 40(5):1635–1660, 1969.
43. C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.
44. R. Wheeler and K. Narendra. Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control*, AC-31(6), June 1986.
45. D. White. Dynamic programming, markov chains, and the method of successive approximations. *Journal of Mathematical Analysis and Applications*, 6:373–376, 1963.
46. S. Whitehead, Jonas Karlsson, and Josh Tenenberg. Learning multiple goal behavior via task decomposition and dynamic policy merging. In J. Connell and S. Mahadevan, editors, *Robot Learning*. Kluwer Academic Publishers, 1993.