

## Exploiting Multiple Secondary Reinforcers in Policy Gradient Reinforcement Learning

**Greg Grudic**

IRCS and GRASP Lab  
University of Pennsylvania  
Philadelphia, PA, USA  
grudic@linc.cis.upenn.edu

**Lyle Ungar**

Computer and Information Science  
University of Pennsylvania  
Philadelphia, PA, USA  
ungar@cis.upenn.edu

### Abstract

Most formulations of Reinforcement Learning depend on a single reinforcement reward value to guide the search for the optimal policy solution. If observation of this reward is rare or expensive, converging to a solution can be impractically slow. One way to exploit additional domain knowledge is to use more readily available, but related quantities as **secondary reinforcers** to guide the search through the space of all policies. We propose a method to augment Policy Gradient Reinforcement Learning algorithms by using prior domain knowledge to estimate desired relative levels of a set of secondary reinforcement quantities. RL can then be applied to determine a policy which will establish these levels. The primary reinforcement reward is then sampled to calculate a gradient for each secondary reinforcer, in the direction of increased primary reward. These gradients are used to improve the estimate of relative secondary values, and the process iterates until reward is maximized. We prove that the algorithm converges to a local optimum in secondary reward space, and that the rate of convergence of the performance gradient estimate in secondary reward space is independent of the size of the state space. Experimental results demonstrate that the algorithm can converge many orders of magnitude faster than standard policy gradient formulations.

### 1 Introduction

Reinforcement Learning (RL) is a trial and error procedure by which an agent learns to improve the reward it receives from the environment. At the core of all RL algorithms is a search procedure that directs the hypothesis and test process by which the agent discovers how to improve its behaviour. The efficiency of an RL algorithm is therefore governed by the size of the agent's search space (i.e. problem domain size) and the effectiveness of the search procedure used. When the problem domain is small enough, or enough prior knowledge is used to direct search, RL algorithms can yield effective solutions which are not easily achievable by other means [Kaelbling *et al.*, 1996].

There is little doubt that mammals (as well as other biological systems) successfully use some form of reinforcement learning [Sutton and Barto, 1990]. Furthermore, the environments where most mammals live are varied, complex, and unpredictable, implying a large problem domain that cannot be blindly searched to the animals advantage. Thus, mammals must employ intelligent search strategies in order to learn how to improve the rewards they receive. One search strategy employed by mammals involves the use of *secondary reinforcers*, which are stimuli that can be associated with the primary reinforcement reward, and thus have similar reinforcing characteristics. Secondary reinforcers are useful to an organism if it can more easily learn how to obtain the stimuli associated with secondary reinforcement than those associated with a primary reinforcement. In general, organisms seek the goals of successful breeding and long term survival. These however can rarely be sampled and therefore provide little guidance in the search for a survival 'policy'. More immediate stimuli such as fear, hunger, cold, pain or pleasure can be much more readily observed and have direct consequences for the animal's long term reward. For example, if a mouse learns that hanging around a kitchen (a secondary reinforcer) is sometimes associated with the primary reinforcer of assuaging hunger (and not starving), then, because the kitchen is always there while the food is not, the mouse can more easily learn to modify its behavior to attain the secondary reinforcer than the primary one.

Minsky [Minsky, 1963] was one of the first to argue that the use of secondary reinforcers may be an important ingredient to creating an artificial intelligence that is able to learn through reinforcement learning. Indeed, one of the motivating factors behind the widely successful temporal-difference reinforcement learning algorithm (TD( $\lambda$ )) is the notion of secondary reinforcers [Sutton and Barto, 1998].

In this work we propose to extend the use of secondary reinforcers to improve the rate of convergence of RL algorithms. To date, the use of secondary reinforcers in RL has largely been limited to simply noting which stimuli are correlated with the primary reward and having the agent learn to maximize these stimuli independently. However, this simple notion of blindly maximizing all secondary reinforcer stimuli is naive because there are potentially complicated interactions among secondary reinforcers associated with a problem domain. In fact, the agent is likely to gain an advantage by con-

trolling the *relative* amounts of stimuli from each reinforcer, rather than concentrating on achieving maximal values. Even in the simple example of a mouse learning to spend time in the kitchen because food may fall to the floor, the mouse may not be best served by spending all of his time in the kitchen because a cat may also at times wander in, and greatly curtail the mouse’s long term survival. Thus the mouse might want to control the percentage of time it spends in kitchen in order to minimize the risk of running into the cat. Hence, even in simple examples, controlling how much of each secondary reinforcer the agent should experience is important.

Consider the example of a student who must write a set of 4 hour exams (one per day), with 20 hours between when one exam finishes and the next exam is scheduled to begin. She knows that both studying and getting a good night’s sleep are associated with getting good marks on an exam, so there are two secondary reinforcers which she can use. She can modify her behavior in such a way that she will study (the first secondary reinforcer) for a longer or shorter period, or conversely she will learn how to get a longer or shorter night’s sleep (the other secondary reinforcer). However, if she simply spends the next 20 hours sleeping, or if she forces herself to study for the next 20 hours, she will do poorly on the exam. Thus, in order to maximize her exam marks, she will need to determine what policy actions (drink more coffee, spend more time at the library, drink more warm milk etc) she must perform in order to get the right ratio of the two secondary reinforcers (i.e. sleep and study).

### 1.1 SRPG Method Overview

In this work we propose a new algorithm called *Secondary Reinforcers Policy Gradient* (SRPG) for solving the problem of determining the relative amounts of secondary reinforcers an agent needs to experience in order to increase its primary reward. Our framework assumes that the agent starts with prior domain knowledge defining a set of secondary reinforcers and in what relative amounts these should be experienced. We further assume that the secondary reinforcers are more readily observed than the primary reward, and thus the agent can more easily learn strategies which allow it to experience these stimuli in the appropriate ratios.

Our algorithm consists of the following repeated steps:

**STEP I:** Use prior domain knowledge to establish the initial estimate of desired relative amounts of secondary reinforcer stimuli. Use RL to learn a policy which will yield these levels of stimuli. During this step, the agent need never experience the primary stimulus, only the secondary ones.

**STEP II:** Based on the policy established in STEP I a gradient on the primary reward is calculated with respect to each secondary reinforcer stimulus. These gradients are then used to update the desired relative amounts of the secondary reinforcers and STEP I of the algorithm is repeated.

If the important secondary reinforcers and the initial guess at their relative ratios is chosen wisely, then we show both theoretically and experimentally that the agent can improve the reward it receives significantly faster than if no such prior

knowledge is used. Because in RL problems it is often the case that primary reinforcers are infrequently experienced by the agent, the rate of convergence measure used in this work is the number of times the agent needs to experience a primary reward to achieve a specific amount of reward.

In applying our proposed framework to the exam studying example given above, the student may start by knowing that sleep and study are both important stimuli, and that she will likely need eight hours of sleep and twelve hours of study. Thus the student can spend many 20 hour periods learning which actions allow her to sleep for eight hours and study for twelve, without ever taking an exam. But by the time she does take an exam, she is quite good at sleeping for eight hours and studying for twelve. Once she observes her first exam grades, she can learn to maximize her grade by systematically adjusting her sleep/wake ratio (i.e. her secondary reinforcers) which prior knowledge tells her are directly related to her mark, rather than just adjusting actions such as drinking milk or coffee based on a primary reward (i.e. her exam mark) which she only rarely observes.

The proposed algorithm is based on methods derived from Policy Gradient Reinforcement Learning (PGRL) [Williams, 1987; 1992; Baird and Moore, 1999; Sutton *et al.*, 2000; Konda and Tsitsiklis, 2000; Baxter and Bartlett, 2000; Grudic and Ungar, 2000b; 2000a], rather than Value Function Reinforcement Learning (VFRL) methods such as  $Q$  learning [Sutton and Barto, 1998]. The motivation for this is three-fold. 1) VFRL methods work by learning a state-action value function which defines the value of executing each action in each state, thus allowing the agent to choose the most valuable action in each state. As a result, the search space grows exponentially with both states and actions and therefore in large problem domains learning a state-action value function can be infeasible [Kaelbling *et al.*, 1996]. Function approximation techniques have been proposed as a method for generalizing state-action value functions in large state spaces, however, this is still very much an open research problem. Conversely, PGRL methods work by starting with an initial parameterized policy which defines the probability of executing a given action in a given state. Typically, far fewer parameters are used to define the policy than there are states, thus directly addressing the need for generalization in large state spaces. PGRL algorithms work by calculating a gradient along a direction of increased reward in parameter space, and then incrementally modifying the policy parameters. PGRL algorithms are theoretically guaranteed to converge to only locally optimal policies, whereas VFRL algorithms can find globally optimal solutions. However, in practice it is usually not feasible to converge to globally optimal solutions in large problem domains in any case. Furthermore, the computational cost of PGRL algorithms grows linearly in the number of parameters used to define the policy, which is in sharp contrast to the exponential growth associated with VFRL algorithms. 2) Because PGRL algorithms start with a parameterized policy, it is relatively simple to choose a policy which incorporates prior knowledge via an appropriate choice of the parametric form of the policy. For example, if we know that in certain states only specific actions are possible, we can choose a policy parameterization which reflects this. The use of prior knowl-

edge in VFRL algorithms is not as easily realized. Finally, 3) many real problem domains are only partially observable (i.e. the agent’s current state cannot be directly observed, but rather must be inferred through observations taken over many time steps), and VFRL algorithms are known to be difficult to implement in such domains. Conversely, PGRL algorithms have been shown to work effectively in partially observable problem domains [Peshkin *et al.*, 2000].

## 2 Theoretical Framework

### 2.1 MDP Assumptions

We make the usual Markov Decision Process (MDP) assumptions and for simplicity (and without loss of generality), we assume that the agent interacts with the environment in a series of episodes of duration  $T$ . The agent’s state at time  $t \in \{1, 2, \dots, T\}$  is given by  $s_t \in S$ ,  $S \subseteq \mathfrak{R}^d$ . Note that the state space can either be discrete or continuous. At each time step the agent chooses from one of  $M > 1$  discrete actions  $a_t \in A$ . The dynamics of the environment are characterized by transition probabilities  $P_{ss'}^a = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ .

Because we are using a policy gradient algorithm, we further assume that the policy followed by the agent is characterized by a parameter vector  $\Theta = (\theta_1, \dots, \theta_K) \in \mathfrak{R}^K$ , and the probability that the agent executes action  $a$  in state  $s$  is given by  $\pi(s, a; \Theta) = Pr\{a_t = a | s_t = s; \Theta\}$ ,  $\forall s \in S, a \in A$  [Sutton *et al.*, 2000]. In addition we assume that  $\pi(s, a; \Theta)$  is differentiable with respect to  $\Theta$ .

### 2.2 Primary Reinforcer

We assume the primary reward the agent receives at time  $t$  is given by  $r_t \in \mathfrak{R}$ . The agent’s goal is to optimize either the average reward function given by:

$$\rho(\Theta) = E \left\{ \frac{1}{T} \sum_{t=1}^T r_t \middle| \Theta \right\} \quad (1)$$

or the discounted reward function given by:

$$\rho(\Theta) = E \left\{ \sum_{t=1}^T \gamma^{t-1} r_t \middle| s_0, \Theta \right\} \quad (2)$$

where  $0 < \gamma < 1$  is a discount factor and  $s_0$  is some initial starting state. Note that both reward functions are assumed to be a function of the agent’s policy which is parameterized by  $\Theta$ .

### 2.3 Secondary Reinforcers

At each time  $t$  the agent senses one of  $N$  stimuli denoted by  $((sr)_t^1, \dots, (sr)_t^N)$ , where  $(sr)_t^n \in \mathfrak{R}$ ,  $\forall n = \{1, 2, \dots, N\}$ . These  $N$  stimuli constitute the agent’s secondary reinforcers, and the average expected stimuli over an episode for each reinforcer, here termed *secondary reinforcer frequencies*, is given by:

$$F_n(\Theta) = E \left\{ \frac{1}{T} \sum_{t=1}^T (sr)_t^n \middle| \Theta \right\} \quad (3)$$

$\forall n = \{1, 2, \dots, N\}$ . As with the primary reward functions (1) and (2), these secondary reward stimulus functions are a function of the agent’s policy which is defined by the parameters  $\Theta$ .

### 2.4 SRPG Algorithm

Let  $i$  indicate the number of times each step in the algorithm has been executed, and  $\Phi^i = (\phi_1^i, \dots, \phi_N^i)$  be the current desired secondary reinforcer frequencies corresponding to  $\mathbf{F}(\Theta) = (F_1^i(\Theta), \dots, F_N^i(\Theta))$ . The Secondary Reinforcers Policy Gradient (SRPG) algorithm consists of the following two steps:

**STEP 1:** *Learn a policy that achieves the currently desired frequency of secondary reinforcers*: In this step the agent establishes these secondary reinforcer frequencies by learning a set of policy parameters  $\Theta_i$  that solve the system of  $N$  nonlinear equations defined by:

$$\mathbf{F}(\Theta_i) = \Phi^i \quad (4)$$

**STEP 2:** *Calculate the gradient of increased primary reward in secondary reinforcers space and update desired secondary reinforcer frequencies*: Estimate the gradient of the primary reward function  $\rho(\Theta_i)$  with respect to the desired secondary reinforcer frequencies  $\Phi^i$ :

$$\frac{\partial \rho(\Theta_i)}{\partial \Phi^i} \quad (5)$$

Given this estimate, the next estimate for the secondary reinforcer frequencies,  $\Phi^{i+1}$ , is given by:

$$\Phi^{i+1} = \Phi^i + \alpha \frac{\partial \rho(\Theta_i)}{\partial \Phi^i} \quad (6)$$

where  $\alpha \in \mathfrak{R}$  is a small positive step size. Increment  $i$  and goto *STEP 1*.

Note that in *STEP 1* of the algorithm the agent does not need to observe the primary reward  $r_t$ . Instead the agent must solve the nonlinear system of equations (4). This can be done using a number of different algorithms [Press *et al.*, 1988]; here we briefly describe one such procedure. The algorithm used in this paper is to first estimate the gradient:

$$\frac{\partial (\mathbf{F}(\Theta^j) - \Phi^i)}{\partial \Theta^j} = -\frac{1}{2} \sum_{n=1}^N (F_n(\Theta^j) - \varphi_n^i) \frac{\partial F_n(\Theta^j)}{\partial \Theta^j}$$

and then use the update equation:

$$\Theta^{j+1} = \Theta^j + \beta \frac{\partial (\mathbf{F}(\Theta^j) - \Phi^i)}{\partial \Theta^j} \quad (7)$$

where  $\beta \in \mathfrak{R}$  is a small positive step size. Assuming that the algorithm doesn’t run into a local minimum,  $\Theta^j$  will converge to  $\Theta^i$  as  $j$  becomes large. See Section 4 for a discussion of what happens when the algorithm encounters a local minimum. Note that the algorithm requires estimates of:

$$\frac{\partial F_n(\Theta^j)}{\partial \Theta^j}$$

for  $n = (1, \dots, N)$ . These estimates can be made using standard policy gradient algorithms [Williams, 1987; 1992;

Sutton *et al.*, 2000; Konda and Tsitsiklis, 2000; Baxter and Bartlett, 2000]. For example, if we use the formulation defined in [Sutton *et al.*, 2000], then

$$\frac{\partial F_n(\Theta^j)}{\partial \Theta^j} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s, a; \theta)}{\partial \theta} Q_{F_n}^\pi(s, a) \quad (8)$$

where  $d^\pi(s)$  is the stationary distribution of states under  $\pi$  and  $Q_{F_n}^\pi(s, a)$  is given by:

$$Q_{F_n}^\pi(s, a) = E \left\{ \frac{1}{T} \sum_{k=1}^T (sr)_{t+k}^n \mid s_t = s, a_t = a, \pi \right\}$$

In *STEP 2* of the algorithm the agent must numerically calculate the gradient (5). This is numerically estimated  $\forall n = 1, \dots, N$  using the following:

$$\frac{\partial \rho(\Theta_i)}{\partial \phi_n^i} = \lim_{\delta_n \rightarrow 0} \frac{\rho(\Theta | (\mathbf{F}(\Theta^j) \equiv \Phi^i + \Delta_n)) - \rho(\Theta_i)}{\delta_n} \quad (9)$$

where  $\Delta_n \in \mathbb{R}^N$ , such that every element of the vector is zero except for element  $n$ . Therefore (5) is estimated using small perturbations (i.e.  $\Delta_n$ ) in each of the elements of  $\Phi^i$ , solving the system  $\mathbf{F}(\Theta_i^*) = \Phi^i + \Delta_n$  for  $\Theta_i^*$  as outlined above, and then observing the associated  $\rho(\Theta | (\mathbf{F}(\Theta^j) \equiv \Phi^i + \Delta_n))$ . Thus in *STEP 2* the agent must observe the primary reward function  $\rho$  at least  $N + 1$  times (once for  $\rho(\Theta_i)$  and once for each of the  $N$  secondary stimuli for  $\rho(\Theta | (\mathbf{F}(\Theta^j) \equiv \Phi^i + \Delta_n))$ ).

## 2.5 Convergence Results

We present two theorems. The first establishes the conditions under which the SRPG algorithm converges.

**Theorem 1:** Let  $\alpha_i \in \mathbb{R}$ ,  $\forall i = 0, 1, \dots$ , such that  $\lim_{i \rightarrow \infty} \alpha_i = 0$  and  $\sum_i \alpha_i = \infty$ . Assume that  $\max_{s,a,k,j,\theta} \left| \frac{\partial^2 \pi(s,a;\theta)}{\partial \theta_k \partial \theta_j} \right| < K < \infty$ ,  $\max_{s,a,k,j,\phi} \left| \frac{\partial^2 \rho(\Theta_i)}{\partial \phi_k^i \partial \phi_j^i} \right| < K < \infty$  and that  $\Theta_i$  exists such that (4) is satisfied. Then, the sequence of  $\Theta_i$  defined by the SRPG algorithm converges such that for any MDP with bounded primary reward  $\lim_{i \rightarrow \infty} \frac{\partial \rho(\Theta_i)}{\partial \Phi^i} = 0$ .

**Proof:** The necessary condition on the bound  $\frac{\partial^2 F(\Theta_i)}{\partial \theta_k \partial \theta_j}$  is established by the bound on  $\frac{\partial^2 \pi(s,a;\theta)}{\partial \theta_k \partial \theta_j}$  due to equation (8). The remainder of the proof then follows directly from Proposition 3.5 on page 96 of [Bertsekas and Tsitsiklis, 1996].  $\square$

The second theorem proves that the rate of convergence of the gradient estimate (9) (with respect to the number of primary reinforcers observed) is independent of the size of the agent's state space.

**Theorem 2:**  $\forall n = 1, \dots, N$  and  $\Delta_n \in \mathbb{R}^N$  assume that there exists a finite  $k \in \mathbb{R}$   $|\rho(\Theta | (\mathbf{F}(\Theta^j) \equiv \Phi^i + \Delta_n)) - \rho(\Theta_i)| \leq k \|\Delta_n\|$  (i.e. Lipschitz smoothness condition). Assume also that the variance in the primary reinforcer estimate  $\rho$  is  $\sigma_\rho^2$ . Then, for any arbitrary small positive  $\epsilon \in \mathbb{R}$ , there exists a  $\Delta_n$  such

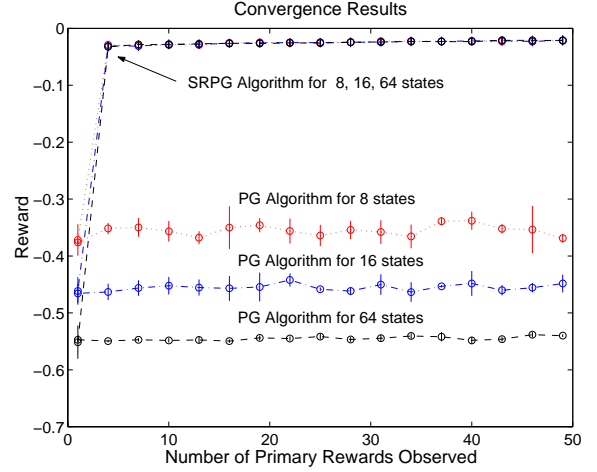


Figure 1: Convergence results, as a function the number of observations of the primary reinforcer, for MDP problems with 8, 16, and 64 states are given.

that

$$\left| \frac{\rho(\Theta | (\mathbf{F}(\Theta^j) \equiv \Phi^i + \Delta_n)) - \rho(\Theta_i)}{\delta_n} - \frac{\partial \rho(\Theta_i)}{\partial \phi_n^i} \right| \leq \epsilon \quad (10)$$

Further, assume that each observation of  $\rho$  is independent, then the variance in the gradient is given by:

$$V \left[ \frac{\rho(\Theta | (\mathbf{F}(\Theta^j) \equiv \Phi^i + \Delta_n)) - \rho(\Theta_i)}{\delta_n} \right] = \frac{2}{M \delta_n^2} \sigma_\rho^2 \quad (11)$$

where  $M$  is the number of times  $\rho(\Theta | (\mathbf{F}(\Theta^j) \equiv \Phi^i + \Delta_n))$  and  $\rho(\Theta_i)$  are observed.

**Proof:** Equation (10) is satisfied by choosing  $\delta_n$  such that  $k \|\Delta_n\| \leq \epsilon$ . Equation (11) is obtained by noting that if  $U = a_1 Y_1 + a_2 Y_2$  and if  $Y_1$  and  $Y_2$  are independent, then  $V(U) = a_1^2 V(Y_1) + a_2^2 V(Y_2)$ . Further, if  $V(Y) = \sigma_Y^2$  and  $\hat{Y}$  is observations of  $Y$ , then if  $Y$  is observed  $M$  times, then  $\hat{Y} = \frac{\sigma_Y^2}{M}$ .  $\square$

## 3 Experimental Results

We implemented MDP examples with 8, 16 and 64 discrete states respectively. The MDPs were fully connected and therefore in the 8 state example the agent had 8 actions to choose from in each state, in the 16 state example the agent had 16 actions in each state, and so on. The actions are chosen using a Boltzmann distribution as follows:

$$\pi(s = i, a = j; \Theta) = \frac{e^{\theta_{ij}}}{\sum_{k=1}^d e^{\theta_{ik}}}$$

where  $d$  is the total number of states. The transition probabilities of the MDP are as follows: if action  $a_t = j$  is chosen in state  $s_t$  at time  $t$ , then  $p(s_{t+1} = j | s_t, a_t = j) = 0.95$ . The agent interacts with the MDP in a series of episodes lasting 10,000 time steps each, always starting in state  $s = 1$ .

The following reward is given to the agent at the end of each episode:

$$\rho(\Theta) = r = - \left( (N_{s=1} - 0.7)^2 + (N_{s=2} - 0.3)^2 \right)$$

where  $N_{s=j} = \frac{\text{no of visits to state } j}{10,000}$ . Thus, in all three examples, the agent achieves maximum reward if it spends 70% of the time in state  $s = 1$  and 30% of the time in state  $s = 2$ . Initially, the values for all the policy parameters  $\theta_{sa}$  are set to 1.0.

Two secondary reinforcers are used by the SRPG algorithm:  $\phi_1 = N_{s=1}$  and  $\phi_2 = N_{s=2}$ . The agent starts with the desired secondary reinforcer frequencies  $\phi_1^0 = 0.5$  and  $\phi_2^0 = 0.5$  in STEP 1 of the SRPG algorithm. The SRPG algorithm learning rates are set to  $\alpha = \beta = 0.01$ . Figure 1 shows learning curves averaged over 10 runs with error bars indicating standard deviation. Note that, as predicted by **Theorem 1**, the convergence of the algorithm as a function of the number of observed primary rewards is not affected by the number of states in the MDP.

The rate of convergence of the SRPG algorithm was compared to the the PG algorithm proposed in [Sutton *et al.*, 2000]:

$$\frac{\partial \rho(\Theta^j)}{\partial \Theta^j} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s, a; \theta)}{\partial \theta} Q^\pi(s, a)$$

where  $d^\pi(s)$  is the stationary distribution of states under  $\pi$  (estimated after each episode by counting the number of times each state is visited) and  $Q^\pi(s, a)$  is given by:

$$Q^\pi(s, a) = E \left\{ \frac{1}{T} \sum_{k=1}^T r_{t+k} \mid s_t = s, a_t = a, \pi \right\}$$

A typical learning curve, as a function of the number of observations of the primary reinforcer, for this PG algorithm for the 8 state MPD is given in Figure 2 (as with the SRPG algorithm the learning step size was set to  $\alpha = 0.01$ ). Note that convergence took approximately 5000 observations of the primary reward function  $\rho$ . Convergence of the PG algorithm for the 16 state example took about 28,000 observations of  $\rho$ . For the 64 state example, convergence was not observed after 1,000,000 observations of  $\rho$ , at which time the algorithm was stopped.

Figure 1 also shows the learning curves (averaged over 10 runs with error bars showing standard deviation) for the PG algorithm after the first 50 observations of the primary reward  $\rho$ . Note that although the SRPG algorithm converges after about 50 runs for all three examples, the PG algorithm does not appreciably improve the reward received by agent over this number of observations of  $\rho$ .

## 4 Conclusion and Discussion

In large problem domains, prior knowledge is essential for an agent to effectively learn via reinforcement reward. However, how domain specific knowledge should be added to RL algorithms to the agent’s advantage is still an open research problem. Secondary reinforcers are stimuli that are directly

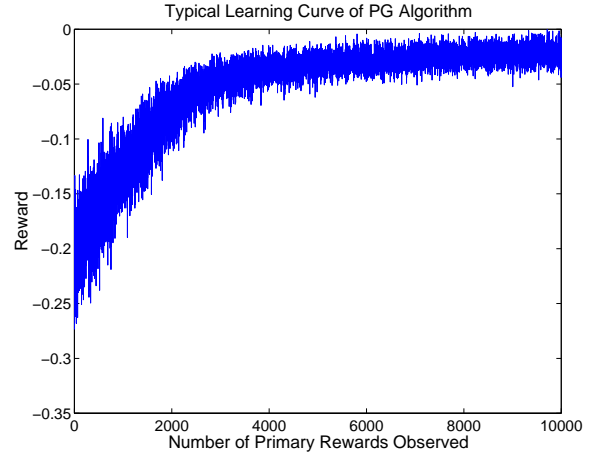


Figure 2: A typical learning curve for the PG algorithm (as a function the number of observations of the primary reinforcer) on an 8 state MDP problem.

related to a primary reinforcement, and thus their relative reward properties can be measured with respect to agent’s global reward function. In this paper we show that secondary reinforcers are a fruitful source of domain specific knowledge which can be effectively used to guide search. In particular, if the primary reinforcement is observed only rarely, while secondary reinforcers are more readily observable, the rate of convergence of RL algorithms, with respect to the number of observations of a primary reward, can be significantly improved.

However, not all secondary reinforcers should be blindly maximized. Some are more important than others, and in order to make full use of secondary reinforcers, an agent should control the relative amounts of this environmental feedback. This paper presents a new policy gradient algorithm, called Secondary Reinforcer Policy Gradient (SRPG), that iterates in the space of secondary reinforcer stimuli to a locally optimal policy. The algorithm consists of two steps. The first step starts with an assumption of the relative amounts of secondary reinforcer stimuli that the agent will need to establish to achieve good primary reward. In this first step the agent learns a policy which achieves these relative amounts of stimuli using a standard policy gradient RL algorithm. A key property of SRPG is that a primary reward need not be observed during this step of the algorithm, and therefore the convergence of algorithm is not slowed because the primary reward is rarely observed. The second step of the algorithm involves the calculation of a gradient in the direction increased primary reward, in the space secondary reinforcer stimuli. If there are  $N$  secondary reinforcers, then step two of the algorithm requires that the primary reinforcer be sampled  $N + 1$  times. However, we present theory showing that the rate of convergence of the gradient estimate is independent of the size of the state space, making the algorithm viable on high dimensional problems. This estimated gradient is used to update the relative desired amounts of secondary reinforcers, and step one of the algorithm is then executed to establish a policy to achieve these relative amounts of stim-

uli, thus improving the primary reward. Step one and two are repeated until the policy converges to a local optimum.

Experimental results are presented which show that the SRPG algorithm can improve the convergence of Policy Gradient (PG) algorithms, requiring many orders of magnitude fewer observations of the primary reward to converge. In addition, we present theory showing that the SRPG algorithm converges to a locally optimal policy (with respect to the primary reward) in the space of secondary reinforcers. However, in order for this theorem to hold, step one of the algorithm must be able to establish a policy that achieves the specified relative amounts of secondary reinforcer stimuli. The algorithm used in this paper to solve this nonlinear system of equations is based on gradient descent, and is therefore prone to convergence to local minimum. However, in the experimental results presented here, the SRPG algorithm converged to close to optimal primary reward, even when local minima were encountered in step one of the algorithm. The reason for this is that these local minima sufficiently improved the primary reward, thus allowing the overall algorithm to continue to converge. Therefore, establishing a more relaxed set of conditions under which the SRPG algorithm will converge is an open research topic.

Further open research questions include the use of secondary reinforcers within the actor-critic reinforcement learning framework [Sutton and Barto, 1998]. Actor critic algorithms combine policy gradient and value function methods and thus can often achieve successful policies when either method alone would fail. Such algorithms may further benefit from domain specific knowledge derived from secondary reinforcers. Finally, an important open research topic is, given a large list of potential secondary reinforcers, formulate an algorithm that finds a small subset which can be used to improve RL algorithms.

## Acknowledgements

Thanks to Jane Mulligan for useful discussions. This work was funded by the GRASP Lab, the IRCS at the University of Pennsylvania, and by the DARPA ITO MARS grant no. DABT63-99-1-0017.

## References

- [Baird and Moore, 1999] L. Baird and A. W. Moore. Gradient descent for general reinforcement learning. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 11, Cambridge, MA, 1999. MIT Press.
- [Baxter and Bartlett, 2000] Jonathan Baxter and Peter L. Bartlett. Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'2000)*, pages 41–48, Stanford University, CA, June 2000.
- [Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [Grudic and Ungar, 2000a] G. Z. Grudic and L. H. Ungar. Localizing policy gradient estimates to action transitions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, volume 17, pages 343–350. Morgan Kaufmann, June 29 - July 2 2000.
- [Grudic and Ungar, 2000b] G. Z. Grudic and L. H. Ungar. Localizing search in reinforcement learning. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, volume 17, pages 590–595. Menlo Park, CA: AAAI Press / Cambridge, MA: MIT Press, July 30 - August 3 2000.
- [Kaelbling *et al.*, 1996] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [Konda and Tsitsiklis, 2000] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K.-R. Miller, editors, *Advances in Neural Information Processing Systems*, volume 12, Cambridge, MA, 2000. MIT Press.
- [Minsky, 1963] Marvin Minsky. Steps toward artificial intelligence. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill Book Company, New York, N.Y., 1963.
- [Peshkin *et al.*, 2000] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. In *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence (UAI 2000)*, Stanford University, CA, June 2000.
- [Press *et al.*, 1988] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, New York NY, 1988.
- [Sutton and Barto, 1990] R. S. Sutton and A. G. Barto. Time-derivative models of Pavlovian reinforcement. In M. Gabriel and J. Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pages 497–537. MIT Press, 1990.
- [Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [Sutton *et al.*, 2000] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K.-R. Miller, editors, *Advances in Neural Information Processing Systems*, volume 12, Cambridge, MA, 2000. MIT Press.
- [Williams, 1987] R. J. Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks*, San Diego, CA, 1987.
- [Williams, 1992] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.