Principled Methods for Advising Reinforcement Learning Agents

Eric Wiewiora WIEWIORA@CS.UCSD.EDU
Garrison Cottrell GARY@CS.UCSD.EDU
Charles Elkan ELKAN@CS.UCSD.EDU
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA

Abstract

An important issue in reinforcement learning is how to incorporate expert knowledge in a principled manner, especially as we scale up to real-world tasks. In this paper, we present a method for incorporating arbitrary advice into the reward structure of a reinforcement learning agent without altering the optimal policy. This method extends the potential-based shaping method proposed by Ng et al. (1999) to the case of shaping functions based on both states and actions. This allows for much more specific information to guide the agent – which action to choose – without requiring the agent to discover this from the rewards on states alone. We develop two qualitatively different methods for converting a potential function into advice for the agent. We also provide theoretical and experimental justifications for choosing between these advice-giving algorithms based on the properties of the potential function.

1. Introduction

Humans rarely approach a new task without presumptions on what type of behaviors are likely to be effective. This bias is a necessary component to how we quickly learn effective behavior across various domains. Without such presumptions, it would take a very long time to stumble upon effective solutions.

In its most general definition, one can think of advice as a means of offering expectations on the usefulness of various behaviors in solving a problem. Advice is crucial during early learning so that promising behaviors are tried first. This is necessary in large domains, where reinforcement signals may be few and far between. A good example of such a problem is chess. The objective of chess is to win a match, and an appropriate reinforcement signal would be based on this. If an agent were to learn chess without prior knowledge, it would have to search for a great deal of time before stumbling onto a winning strategy. We can speed up this process by advising the agent such that it realizes that taking pieces is rewarding and losing pieces is regretful. This advice creates a much richer learning environment but also runs the risk of distracting the agent from the true goal – winning the game.

Another domain where advice is extremely important is in robotics and other real-world applications. In the real world, learning time is very expensive. In order to mitigate "thrashing" – repeatedly trying ineffective actions – rewards should be supplied as often as possible (Mataric, 1994). If the problem is inherently described by sparse rewards it is very difficult to change the reward structure of the environment without disrupting the goal.

Advice is also necessary in highly stochastic environments. In such an environment, the expected effect of an action is not immediately apparent. In order to get a fair assessment of the value of an action, the action must be tried many times. If advice can focus this exploration on actions that are likely to be optimal, a good deal of exploration time can be saved.

2. Previous Approaches

Incorporating bias or advice into reinforcement learning takes many forms. The most elementary method for biasing learning is to choose some initialization based on prior knowledge of the problem. A brief study of the effect of different Q-value initializations for one domain can be found in Hailu and Sommer (1999). The relevance of this method is highly dependent on the internal representations used by the agent. If the agent simply maintains a table, initialization is easy,

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

but if the agent uses a more complex representation, it may be very difficult or impossible to initialize the agent's Q-values to specific values.

A more subtle approach to guiding the learning of an agent is to manipulate the policy of the agent directly. The main motivation for such an approach is that learning from the outcomes of a reasonably good policy is more beneficial than learning from random exploration. A method for incorporating an arbitrary number of external policies into an agent's policy can be found in Malak and Kholsa (2001). Their system uses an elaborate policy weighting scheme to determine when following an external policy is no longer beneficial. Another twist on this idea is to learn directly from other agents' experiences (Price & Boutilier, 1999). These methods have the advantage that they do not rely on the internal representations the agent uses. On the other hand, they only allow advice to come in the form of a policy. Also, because the policy the agent is learning from may be very different from the policy an agent is trying to evaluate, the types of learning algorithms the agent can use are restricted.

If reliable advice on which actions are safe and effective is known, one can restrict the agent's available actions to these. Deriving such expert knowledge has been heavily studied in the control literature and has been applied to reinforcement learning by Perkins and Barto (2001). This method requires extensive domain knowledge and may rule out optimal actions.

The method we develop in this paper closely resembles shaping. With shaping, the rewards from the environment are augmented with additional rewards. These rewards are used to encourage behavior that will eventually lead to goals, or to discourage behavior that will later be regretted. When done haphazardly, shaping may alter the environment such that a policy that was previously suboptimal now becomes optimal with the incorporation of the new reward. The new behavior that is now optimal may be quite different from the intended policy, even when relatively small shaping rewards are added. A classic example of this is found in Randløv and Alsrøm (1998). While training an agent to control a bicycle simulation, they rewarded an agent whenever it moved towards a target destination. In response to this reward, the agent learned to ride in a tight circle, receiving reward whenever it moved in the direction of the goal. Potential-based shaping, which we will describe in detail later, was developed to prevent learning such policies (Ng et al., 1999).

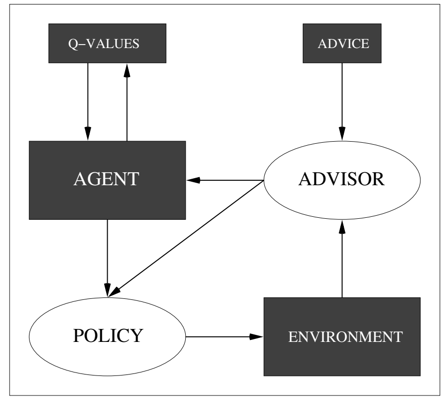


Figure 1. The model we assume for our advising system. The environment, the advice function, and the Q-value estimator are all "black boxes". We show methods for altering the policy, and the environmental feedback such that they incorporate the advice.

3. Preliminaries

We follow the standard reinforcement learning framework, making as few assumptions as possible about access to the dynamics of the environment or the internal representations of the agent. Our method alters learning by adding an advisor that is capable of changing the reinforcement the agent receives, as well as altering the agent's policy. Below we list common assumptions made about the environment and learning mechanisms used by the agent.

3.1. Terminology

Most reinforcement learning techniques model the learning environment as a Markov decision process (MDP) (see Sutton & Barto, 1998). An MDP is defined as $(S, S_0, A, T, R, \gamma)$, where S is the (possibly infinite) set of states, $S_0(s)$ is the probability of the agent starting in state s, A is the set of actions, $T(s'|s, a)$ is the probability of transitioning to state s' when performing action a in state s, $R(s, a, s')$ is a stochastic function defining reinforcement received when action a is performed in state s resulting in a transition to state s', and γ is the discount rate that weighs the importance of short term and long term reward.

The usual reinforcement learning task is to find a policy $\pi : S \to A$ that maximizes the expected total discounted reinforcement:

$$\sum_{t=0}^{\infty} \gamma^t r_t,$$

where r_t is the reinforcement received at time t. Some

MDPs contain special *terminal states* to represent accomplishing a task's goal or entering an irrecoverable situation. When an agent transitions to one of these states all further actions transition to a null state, and all further reinforcements are zero.

We focus on reinforcement learning algorithms that use Q-value estimates to determine the agent's policy. Q-values represent the expected future discounted reward after taking action a in state s. When the Q-values for a particular policy π are accurate, they satisfy the following recursion relation:

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)\big(E[R(s,a,s')] + \gamma Q^\pi(s', \pi(s'))\big)$$

The greedy policy, $\pi^g(s) = \operatorname{argmax}_a Q(s,a)$, is optimal if the Q-values are accurate for this policy.

In order to learn the proper Q-values for the greedy policy, we could use either Q-learning or Sarsa. Both of these methods update the Q-values based on experiences with the MDP. An *experience* is a quadruple $\langle s, a, r, s' \rangle$ where action a is taken in state s, resulting in reinforcement r and a transition to next state s'. For each experience, these methods update the Q-values according to the rule

$$Q(s,a)^\pi \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s', a') - Q(s,a)\big)$$

where α is the learning rate, and a' is an action specified by the specific learning method. For Sarsa learning, a' is the next action the agent will perform. Q-learning sets a' to be the greedy action for state s'. See Sutton and Barto (1998) for more details on these and other reinforcement learning algorithms.

3.2. Potential-based Shaping

Ng et al. proposed a method for adding shaping rewards to an MDP in a way that guarantees the optimal policy maintains its optimality. They define a *potential function* $\Phi()$ over the states. The shaping reward for transitioning from state s to s' is defined in terms of $\Phi()$ as:

$$F(s,s') = \gamma\Phi(s') - \Phi(s),$$

The advisor adds this shaping reward to the environmental reward for every state transition the learner experiences.

The potential function can be viewed as defining a topography over the state space. The shaping reward for transitioning from one state to another is therefore the discounted change of this potential function. Because the total discounted change in potential along any path that starts and ends at the same state is zero, this method guarantees that no cycle yields a net benefit due to the shaping. This was the problem faced in the bicycle simulation mentioned before. In fact, Ng et al. prove that any policy that is optimal for an MDP augmented with a potential-based shaping reward will also be optimal for the unaugmented MDP.

4. Potential-based Advice

Although potential-based shaping is an elegant tool for giving guidance to a reinforcement learner, it is not general enough to represent any type of advice. Potential-based shaping can give the agent a hint on whether a particular state is good or bad, but it cannot provide the same sort of advice about various actions.

We extend potential-based shaping to the case of a potential function defined over both states and actions. We define *potential-based advice* as a supplemental reward determined by the states the agent visits and the actions the agent chooses.

One of the consequences of this extension is that the modification to the MDP cannot be described as the addition of a shaping function. A shaping function's parameters are the current state, the action chosen, and the resulting state. This is the same information that determines the reward function. The advice function requires an additional parameter related to the policy the agent is currently evaluating. Note that if the policy being evaluated is static, this parameter is effectively constant, and therefore the advice may be represented as a shaping function.

We propose two methods for implementing potential-based advice. The first method, which we call *look-ahead advice*, is a direct extension of potential-based shaping. A second method, called *look-back advice*, is also described. This method provides an alternative when the agent's policy cannot be directly manipulated or when Q-value generalization may make look-ahead advice unattractive.

4.1. Look-Ahead Advice

In look-ahead advice, the augmented reward received for taking action a in state s, resulting in a transition to s' is defined as

$$F(s,a,s',a') = \gamma\Phi(s',a') - \Phi(s,a),$$

Where a' is defined as in the learning rule. We refer to the advice component of the reward given the the agent at time t as f_t.

We analyze how look-ahead advice changes the Q-values of the optimal policy in the original MDP. Call the optimal Q-value for some state and action in the original MDP $Q^*(s,a)$. We know that this value is equal to the expected reward for following the optimal policy $\pi^*()$:

$$Q^*(s,a) = \mathrm{E}\Big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi = \pi^*\Big]$$

When this policy is held constant and its Q-values are evaluated in the MDP with the addition of advice rewards, the Q-values differ from their true value by the potential function:

$$\begin{aligned} Q^*(s,a) &= E\Big[\sum_{t=0}^{\infty} \gamma^t (r_t + f_t)\Big] \\ &= E\Big[\sum_{t=0}^{\infty} \gamma^t (r_t + \gamma\Phi(s_{t+1}, a_{t+1}) \\ &\qquad -\Phi(s_t, a_t))\Big] \\ &= E\Big[\sum_{t=0}^{\infty} \gamma^t (r_t)\Big] \\ &\quad +E\Big[\sum_{t=1}^{\infty} \gamma^t \Phi(s_t, a_t)\Big] \\ &\quad -E\Big[\sum_{t=0}^{\infty} \gamma^t \Phi(s_t, a_t)\Big] \\ &= E\Big[\sum_{t=0}^{\infty} \gamma^t r_t\Big] - \Phi(s,a) \end{aligned}$$

In order to recover the optimal policy in a MDP augmented with look-ahead advice rewards, the action with the highest Q-value plus potential must be chosen. We call this policy *biased greedy*. It is formally defined as

$$\pi^b(s) = \operatorname*{argmax}_a \Big(Q(s,a) + \Phi(s,a)\Big).$$

Notice that when the Q-values are initialized are zero, the biased greedy policy chooses actions with the highest value in the potential function, encouraging exploration of the highly advised actions first. Any policy can be made biased by adding the potential function to the current Q-value estimates for the purpose of choosing an action.

4.1.1. Learnability of the Optimal Policy

Although we can recover the optimal policy using the biased greedy policy, we still need to determine whether the optimal policy is learnable. While we cannot make a claim on the learnability of the optimal policy under any learning scheme, we can make claims on its learnability when the state and action space are finite. In this case, the learning dynamics for the agent using look-ahead advice and a biased policy are essentially the same as an unbiased agent whose Q-values were initialized to the potential function.

We define two reinforcement learners, L and L', that will experience the same changes in Q-values throughout learning. Let the initial values of L's Q-table be $Q(s,a) = Q_0(s,a)$. Look-ahead advice $F()$, based upon the potential function $\Phi()$ will be applied during learning. The other learner, L', will have a Q-table initialized to $Q'_0(s,a) = Q_0(s,a) + \Phi(s,a)$. This learner will not receive advice rewards.

Both learners' Q-values are updated based on an experience using the standard reinforcement learning update rule described previously:

$$\begin{aligned} Q(s,a) &\leftarrow Q(s,a) + \alpha\big(\\ &\qquad \underbrace{r + F() + \gamma Q(s',a') - Q(s,a)}_{\delta Q(s,a)}\big), \\ Q'(s,a) &\leftarrow Q'(s,a) + \alpha\big(\\ &\qquad \underbrace{r + \gamma Q'(s',a') - Q'(s,a)}_{\delta Q'(s,a)}\big). \end{aligned}$$

One can think of the above equations as updating the Q-values with an error term scaled by α, the learning rate. We refer to the error terms as $\delta Q(s,a)$ and $\delta Q'(s,a)$. We also track the total change in $Q(\cdot)$ and $Q'(\cdot)$ during learning. The difference between the original and current values in $Q(\cdot)$ and $Q'(\cdot)$ are referred to as $\Delta Q(\cdot)$ and $\Delta Q'(\cdot)$, respectively. The Q-values for the learners can be represented as their initial values plus the change in those values that resulted from the updates:

$$\begin{aligned} Q(s,a) &= Q_0(s,a) + \Delta Q(s,a) \\ Q'(s,a) &= Q_0(s,a) + \Phi(s,a) + \Delta Q'(s,a). \end{aligned}$$

Theorem 1 *Given the same sequence of experiences during learning,* $\Delta Q(\cdot)$ *always equals* $\Delta Q'(\cdot)$.

Proof: Proof by induction. The base case is when the Q-table entries for s and s' are still their initial values. The theorem holds for this case, because the entries in $\Delta Q(\cdot)$ and $\Delta Q'(\cdot)$ are both uniformly zero.

For the inductive case, assume that the entries $\Delta Q(s,a) = \Delta Q'(s,a)$ for all s and a. We show that in response to experience $\langle s, a, r, s' \rangle$, the error terms $\delta Q(s,a)$ and $\delta Q'(s,a)$ are equal.

First we examine the update performed on $Q(s,a)$ in the presence of the advice:

$$\begin{aligned}
\delta Q(s,a) &= r + F() + \gamma Q(s',a') - Q(s,a) \\
&= r + \gamma\Phi(s',a') - \Phi(s,a) \\
&\quad +\gamma\big(Q_0(s',a') + \Delta Q(s',a')\big) \\
&\quad -Q_0(s,a) - \Delta Q(s,a)
\end{aligned}$$

Now we examine the update performed on $Q'(s,a)$:

$$\begin{aligned}
\delta Q'(s,a) &= r + \gamma Q'(s',a') - Q'(s,a) \\
&= r + \gamma\big(Q_0(s',a') + \Phi(s',a') \\
&\quad +\Delta Q(s',a')\big) \\
&\quad -Q_0(s,a) - \Phi(s,a) - \Delta Q(s,a) \\
&= r + \gamma\Phi(s',a') - \Phi(s,a) \\
&\quad +\gamma\big(Q_0(s',a') + \Delta Q(s',a')\big) \\
&\quad -Q_0(s,a) - \Delta Q(s,a) \\
&= \delta Q(s,a)
\end{aligned}$$

Both Q-tables are updated by the same value, and thus $\Delta Q(\cdot)$ and $\Delta Q'(\cdot)$ are still equal. □

Because the Q-values of these two agents change the same amount given the same experiences, they will always differ by the amount they differed in their initialization. This amount is exactly the potential function.

Corollary 1 *After learning on the same experiences using standard reinforcement learning update rules, the biased policy for an agent receiving look-ahead advice is identical to the unbiased policy of an agent with Q-values initialized to the potential function.*

This immediately follows from the proof. The implication of this is that any theoretical results for the convergence of a learner's greedy policy to the optimal policy will hold for the biased greedy policy of an agent receiving look-ahead advice.

If the potential function makes finer distinctions in the state space than the agent's Q-value approximator, the agent may perceive the potential function as stochastic. Because the difference in Q-values between the optimal action and a sub-optimal action can be arbitrarily close, any amount of perceived randomness in the potential function may cause the biased greedy policy to choose a suboptimal action[1]. This remains true after any amount of learning.

[1]Generalized Q-values are usually not capable of representing the optimal policy. State distinctions made by the potential function may allow the agent to learn a better policy than its Q-value approximation would ordinarily allow.

5. Look-Back Advice

So far we have assumed that the potential function is deterministic and stable throughout the lifetime of the agent, and that we can manipulate the agent's policy. If either of these conditions is violated, look-ahead advice may not be desirable.

An alternate approach to potential-based biasing examines the difference in the potential function of the current and previous situations an agent experienced. The advice received by choosing action a_t in state s_t, after being in state s_{t-1} and choosing a_{t-1} in the previous time step is

$$F(s_t,a_t,s_{t-1},a_{t-1}) = \Phi(s_t,a_t) - \gamma^{-1}\Phi(s_{t-1},a_{t-1}).$$

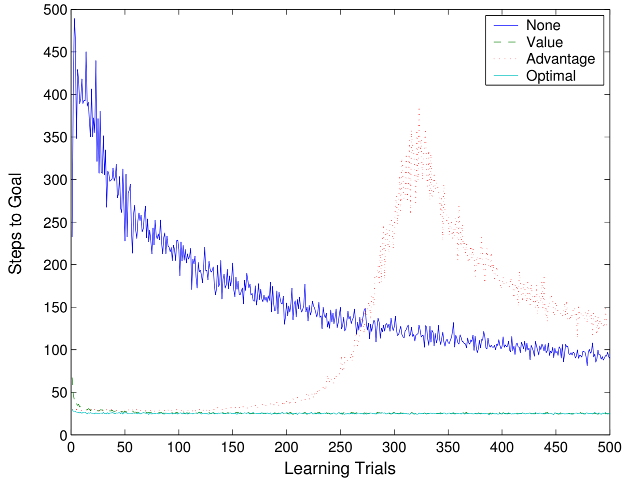
When the agent starts a trial, the potential of the previous state and action is set to 0.

Let's examine what the Q-values are expected to converge to while evaluating a stationary policy π and receiving look-back advice:

$$\begin{aligned}
Q^*(s,a) &= E\Big[\sum_{t=0}^{\infty}\gamma^t(r_t+f_t)\Big] \\
&= E\Big[\sum_{t=0}^{\infty}\gamma^t\big(r_t+\Phi(s_t,a_t) \\
&\qquad -\gamma^{-1}\Phi(s_{t-1},a_{t-1})\big)\Big] \\
&= E\Big[\sum_{t=0}^{\infty}\gamma^t\big(r_t\big)\Big] \\
&\qquad +E\Big[\sum_{t=0}^{\infty}\gamma^t\Phi(s_t,a_t)\Big] \\
&\qquad -E\Big[\sum_{t=-1}^{\infty}\gamma^t\Phi(s_t,a_t)\Big] \\
&= E\Big[\sum_{t=0}^{\infty}\gamma^t r_t\Big] - \gamma^{-1}E\big[\Phi(s_{-1},a_{-1})\big]
\end{aligned}$$

Here $E[\Phi(s_{-1},a_{-1})]$ is the expected value of the potential function of the previous state, given π. Because the agent's exploration history factors into the advice, only on-policy learning rules such as Sarsa should be used with this method.

The correct Q-values for all actions in a given state differ from their value in the absence of look-back advice by the same amount. This means that we can use most



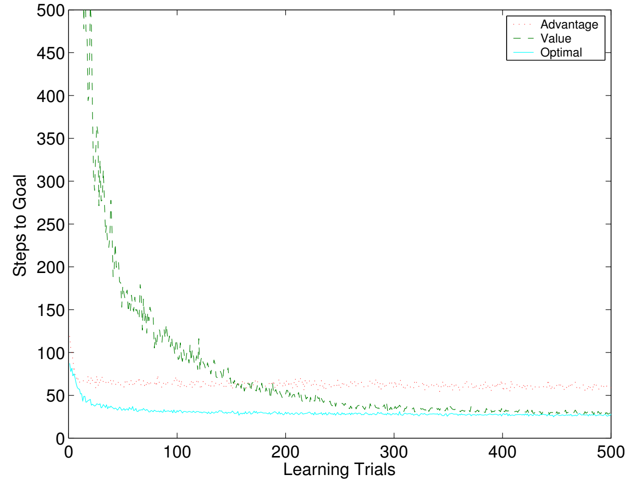


Figure 2. Experiments with different types of bias in a gridworld. On the left look-ahead advice is used, and on the right look-back advice is used. Both methods seem specialized for one type of advice

policies with the assurance that the advised agent will behave similarly to a learner without advice after both have learned sufficiently. The policies where this holds true share the property that they are invariant to a constant addition to all the Q-values in a given state. Some examples of such policies are greedy, ϵ-greedy, and (perhaps surprisingly) softmax. Because we do not have to manipulate the agent's policy to preserve optimality, this advising method can also be used in conjunction with an actor-critic learning architecture.

This analysis also suggests that look-back advice is less sensitive to perceived randomness in the potential function. Learning with this form of advice already faces randomness in approximation the value of the potential function of the state and action previous to the current choice. Extra randomness in the potential function would be indistinguishable from other sources of randomness in the agent's experience. This robustness to noise will likely come at a cost of learning time, however.

At this point we do not have a proof that an agent's Q-values will converge to the the expected values derived above. All experiments in tabular environments support this claim, however.

6. Experiments

We have tested our advice-giving algorithms in a stochastic gridworld to gain some insight into the algorithms' behavior. Our gridworld experiments replicate the methodology found in Ng et al. (1999). We use a 10×10 gridworld with a single start state in the lower left corner. A reward of -1 is given for each action the agent takes, except that the agent receives a reward of 0 for transitioning to the upper right corner. When the agent reaches the upper right corner, the trial ends and the agent is placed back at the start state. Agents choose from four actions, representing an intention to move in one of the four cardinal directions. An action moves the agent the intended direction with probability 0.8, and a random direction otherwise. Any movement that would move the agent off the grid instead leaves the agent in its current position. All learners use one-step Sarsa learning with a learning rate of 0.02, a tabular Q-table initialized uniformly 0, and follow a policy where the greedy action is taken with probability 0.9, and a random action is taken otherwise.

Under our framework, advice is interpreted by the agent as a hint on the Q-values. This advice may take two qualitatively different forms. State-value advice provides an estimate of the value of a state while following the agent's objective policy. This is the only type of advice potential-based shaping can use. The potential function used in state-value advice is equal to the minimum steps from the current state to the goal, divided by the probability an action will move the agent in the intended diretion.

Advantage advice provides an estimate of the relative advantage of different actions in a given state. In many situations, advantage advice is much simpler and more readily available than state-value advice. In our domain, advantage advice has a potential function equal to -1 for the move down or move left actions, and a value of 0 for the other two actions. This is approximately the difference between the true Q-values of the sub-optimal actions and the preferable move up or left actions.

It is also possible to receive combinations of the two forms of advice. In this case, the potential functions are added together. We define optimal advice as the sum of the previously mentioned state-value and advantage advice. This advice is very close to the true Q-values for all states and actions, making it nearly optimal in terms of reducing learning time.

Figure 2 shows the results of experiments with different types of advice using both of the algorithms. For the look-ahead advice algorithm, advice on the value of states appears more useful than the advantage of different actions. This is due to the large discrepancy between the agent's initial Q-values and the values they converge to. The average value of a state in the environment is -12. Without advice on the magnitude of the Q-values, a good deal of exploration is required before the agent learns a good approximation of the value of states.

When advice only consists of the advantage of different actions, an interesting behavior emerges. The agent begins learning following an optimal policy. However, later during learning the agent abandons the optimal policy. Because the agent has explored the optimal actions more than others, the agent learns a better approximation for their Q-values, which are negative. The suboptimal actions are explored less, and thus have values closer to zero.

The look-back advising algorithm shows the opposite result. Value advice starts off very bad. The reason for this behavior can be explained by examining how the advice reward function interacts with the potential function. When the agent transitions from a state with low potential to one with a higher potential, it will receive a positive reward. Unfortunately, the agent receives this reward only after it has taken another action. Thus, if the agent takes a step towards the goal, and then immediately steps away from the goal, the Q-value for stepping away from the goal will be encouraged by the advice reward.

The look-back advice algorithm performed much better with advantage advice. The agent immediately finds a reasonable policy, and maintains a good level of performance throughout learning. The agent using look-back advice is able to maintain good performance because it follows the policy suggested by the bias less diligently than the look-ahead agent. Thus, the look-back agent can pace exploration more evenly as learning progresses.

These experiments shed light on when one of these

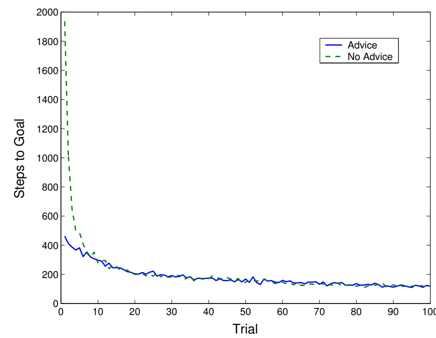


Figure 3. Results for the mountain car problem averaged over 20 runs. The baseline learning algorithm partitions the state space using CMACS, and uses Sarsa(λ) as its learning rule. The advice is given using look-back advice. The advice is based on a policy that aimed to increase a the car's mechanical energy.

methods should be preferred over the other. When advice on state values prevails, look-ahead advice should be used. When the advice comes in the form of a preference for action selection, look-back advice should be given. When both types of advice are present, both algorithms do very well.

6.1. Mountain-Car Problem

Our second experiment examines how simple advice can improve learning performance in a continuous-state control task. The problem we examine is the mountain car problem, a classic testbed for RL algorithms. The task is to get a car to the top of the mountain on the right side of the environment. The mountain is too steep for the car to drive up directly. Instead the car must first back up in order to gain enough momentum to reach the summit. Like the grid-world environment, the agent receives a -1 penalty for every step the agent takes until the goal condition is met.

We took existing code that solves the mountain car problem written by Sridhar Mahadevan[2]. By testing on existing code, we show that our algorithm can treat the agent as a black box, and that advice can improve the performance of agents who can already solve the problem efficiently.

In order to improve learning, We use a potential function that encourages the agent to increase its total me-

[2]http://www-anw.cs.umass.edu/rlr/distcode/mcar.tar The code was modified so that the agent starts a new trial in the center of the valley with zero velocity. The modified code is available on request.

chanical energy. This is accomplished by setting the potential to -1 for choosing an action that accelerates the car in the direction opposite its current velocity, and 0 otherwise. Following this strategy will cause the agent to make consistently faster swings through the valley, reaching higher positions on slopes before the car's momentum is expended. Note that this potential function depends upon the agent's actions and the car's current velocity, but ignores the agent's position in the world. Also, this strategy is not the fastest way for the agent to reach the goal.

Because the advice makes suggestions on appropriate actions but not states, we used the look-back algorithm to add advice. The agent's learning algorithm, Q-value representation and policy remain unaltered with the incorporation of the advice.

As can be seen in figure 3, the advice reduces early learning time by 75%. We show results with the eligibility trace decay rate, λ, individually optimized for each method. The remaining parameters are left at their original value. Without advice, a $\lambda = 0.9$ yields good results. With advice, however, $\lambda = 0.2$ provided the best performance. When lambda is set near one, the influence of the advice tends to be cancelled by future experiences, leaving little improvement over no advice. This effect can be mitigated by scaling the advice by $1/(1-\lambda)$ if this parameter value is available to the advisor.

7. Discussion

We have presented a method for incorporating advice into an arbitrary reinforcement learner in a way that preserves the value of any policy in terms of the original MDP. Although the advice in itself does not alter the agent's ability to learn a good policy, the learning algorithm and state representation the agent uses must be capable of representing a good policy to begin with. Also, it should be stressed that our method does not act as a replacement for the original rewards in the MDP. Without environmental reinforcement, an agent recieving advice will eventually learn flat Q-values for every state.

Although we have assumed no inherent structures in the reinforcement learning agent, our advising method has analogues in many specific learning architectures. We have already mentioned the connection between look-ahead advice and Q-value initialization. If the agent uses a linear combination of features to represent its Q-values, the potential function can be incorporated into the feature set. If the potential feature had a fixed weight of 1 throughout learning, it would be an exact emulation of look-ahead advice. Schemes where advice is built directly into the agent's Q-value approximator have been proposed in Bertsekas and Tsitsiklis (1996); Maclin and Shavlik (1996).

Acknowledgements

We would like to acknowledge support for this research from Matsushita Electric Industrial Co., Ltd. and helpful comments from GURU.

References

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Athena Scientific.

Hailu, G., & Sommer, G. (1999). On amount and quality of bias in reinforcement learning. *IEEE International Conference on Systems, Man and Cybernetics*.

Maclin, R., & Shavlik, J. W. (1996). Creating advice-taking reinforcement learners. *Machine Learning*, *22*, 251–281.

Malak, R. J., & Kholsa, P. K. (2001). A framework for the adaptive transfer of robot skill knowledge among reinforcement learning agents. *Robotic Automation, IEEE International Conference*.

Mataric, M. J. (1994). Reward functions for accelerated learning. *Machine Learning, Proceedings of the Ninth International Conference*. Morgan Kaufmann.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: theory and application to reward shaping. *Machine Learning, Proceedings of the Sixteenth International Conference*. Bled, Slovenia: Morgan Kaufmann.

Perkins, T., & Barto, A. (2001). Lyapunov design for safe reinforcement learning control. *Machine Learning, Proceedings of the Sixteenth International Conference*. Morgan Kaufmann.

Price, B., & Boutilier, C. (1999). Implicit imitation in multiagent reinforcement learning. *Machine Learning, Proceedings of the Sixteenth International Conference*. Bled, Slovenia: Morgan Kaufmann.

Randløv, J., & Alsrøm, P. (1998). Learning to ride a bicycle using reinforcement learning and shaping. *Machine Learning, Proceedings of the Fifteenth International Conference*. Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. The MIT Press.