Algorithms for Inverse Reinforcement Learning

Andrew Y. Ng ANG@CS.BERKELEY.EDU
Stuart Russell RUSSELL@CS.BERKELEY.EDU
CS Division, U.C. Berkeley, Berkeley, CA 94720 USA

Abstract

This paper addresses the problem of *inverse reinforcement learning* (IRL) in Markov decision processes, that is, the problem of extracting a reward function given observed, optimal behaviour. IRL may be useful for apprenticeship learning to acquire skilled behaviour, and for ascertaining the reward function being optimized by a natural system. We first characterize the set of all reward functions for which a given policy is optimal. We then derive three algorithms for IRL. The first two deal with the case where the entire policy is known; we handle tabulated reward functions on a finite state space and linear functional approximation of the reward function over a potentially infinite state space. The third algorithm deals with the more realistic case in which the policy is known only through a finite set of observed trajectories. In all cases, a key issue is *degeneracy*—the existence of a large set of reward functions for which the observed policy is optimal. To remove degeneracy, we suggest some natural heuristics that attempt to pick a reward function that maximally differentiates the observed policy from other, sub-optimal policies. This results in an efficiently solvable linear programming formulation of the IRL problem. We demonstrate our algorithms on simple discrete/finite and continuous/infinite state problems.

1. Introduction

The inverse reinforcement learning (IRL) problem can be characterized informally as follows (Russell, 1998):

Given 1) measurements of an agent's behaviour over time, in a variety of circumstances, 2) if needed, measurements of the sensory inputs to that agent; 3) if available, a model of the environment.

Determine the reward function being optimized.

We can identify two sources of motivation for this problem. The first arises from the potential use of reinforcement learning and related methods as computational models for animal and human learning (Watkins, 1989; Schmajuk & Zanutto, 1997; Touretzky & Saksida, 1997). Such models are supported both by behavioural studies and by neurophysiological evidence that reinforcement learning occurs in bee foraging (Montague et al., 1995) and in songbird vocalization (Doya & Sejnowski, 1995). This literature assumes, however, that the reward function is fixed and known—for example, models of bee foraging assume that the reward at each flower is a simple saturating function of nectar content. Yet it seems clear that *in examining animal and human behaviour we must consider the reward function as an unknown to be ascertained through empirical investigation.* This is particularly true of *multiattribute* reward functions. Consider, for example, that the bee might weigh nectar ingestion against flight distance, time, and risk from wind and predators. It is hard to see how one could determine the relative weights of these terms *a priori.* Similar considerations apply to human economic behaviour, for example. Hence, inverse reinforcement learning is a fundamental problem of theoretical biology, econometrics, and other fields.

A second motivation arises from the task of constructing an intelligent agent that can behave successfully in a particular domain. An agent designer (or indeed the agent itself) may have only a very rough idea of the reward function whose optimization would generate "desirable" behaviour, so straightforward reinforcement learning may not be usable. (Consider, for example, the task of "driving well.") One source of information for learning is the behaviour of other "expert" agents, as used in *imitation learning* and *apprenticeship learning.* In this setting, it is commonly assumed that the purpose of observation is to learn a *policy,* i.e., a direct representation of a mapping from states to actions. We propose instead to recover the expert's reward function and to use this to generate desirable behaviour. We suggest that the reward function often provides a much more parsimonious description of behaviour. After all, the entire field of reinforcement learning is founded on the presupposition that the reward function, rather than the policy, is the most succinct, robust, and transferrable definition of the task. Hence, it seems likely that inverse reinforcement learning may, in some domains, provide an effective form of apprenticeship learning.

To our knowledge, this computational task has not been well-studied in computer science, control theory, psychology, or biology. The closest work is in economics, where the task of multiattribute utility assessment has been studied in depth—that is, how does a person actually combine the various attributes of each available choice when making a decision. The theory is well-developed (Keeney & Raiffa, 1976), and the applications numerous. However, this field studies only *one-shot* decisions where a single action is taken and the outcome is immediate. The sequential case was first considered by Sargent (1978), who tried to ascertain the effective hiring cost for labor by examining a firm's hiring behaviour over time, assuming it to be rational. In the last decade, the area of structural estimation of Markov decision processes in econometrics has grown rapidly (Rust, 1994). Some of the basic ideas carry over to our setting. IRL also appeared briefly in control theory: in the early 1960s, Kalman posed the problem of recovering the objective function for a deterministic linear system with quadratic costs. It was recently solved as a semidefinite program (Boyd et al., 1994).

In this paper, we address the IRL problem in settings more familiar to the machine learning community, beginning with finite Markov decision processes (MDPs). Section 2 gives formal definitions of MDPs and the IRL problem; we focus initially on the setting in which the model is known and the complete policy is given. Section 3 characterizes the set of all reward functions for which a given policy is optimal. We demonstrate that the set contains many degenerate solutions, including, for example, the reward function that is identically zero everywhere. We resolve this difficulty via heuristics that attempt to identify a reward function that maximally differentiates between the observed policy and other, sub-optimal policies. This can be done efficiently in the discrete case using linear programming. Section 4 deals with the case of large or infinite state spaces, for which an explicit, tabular representation of the reward function would be infeasible. We show that if the fitted reward function is represented as a linear combination of arbitrary, fixed basis functions, then the IRL problem remains in the class of linear programs and can again be solved efficiently. Section 5 deals with the more realistic case in which the policy is known only through a finite set of observed trajectories; for this, we present a simple iterative algorithm. The three algorithms we develop are then applied, in Section 6, to some simple examples including both discrete and continuous stochastic navigation problems, and the "mountain–car" problem. In all cases, we are able to recover a reward function that "explains" the observed behavior fairly well. Finally, Section 7 summarizes our findings and describes directions for future work.

2. Notation and Problem Formulation

In this section, we introduce some notation, definitions, and basic theorems for Markov decision processes. We then define the version of the IRL problem that we will address.

2.1 Markov Decision Processes

A (finite) MDP is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where

- S is a finite set of N **states**.
- $A = \{a_1, \dots, a_k\}$ is a set of k **actions**.
- $P_{sa}(\cdot)$ are the state **transition probabilities** upon taking action a in state s.
- $\gamma \in [0, 1)$ is the **discount factor**.
- $R : S \mapsto \mathbb{R}$ is the **reinforcement function**, bounded in absolute value by $R_{\max}$.

For simplicity in exposition, we have written rewards as $R(s)$ rather than $R(s, a)$; the extension is trivial.

A **policy** is defined as any map $\pi : S \mapsto A$, and the **value function** for a policy π, evaluated at any state s_1 is given by

$$V^\pi(s_1) = \mathrm{E}\left[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + \cdots |\pi\right]$$

where the expectation is over the distribution of the state sequence $(s_1, s_2, \dots)$ we pass through when we execute the policy π starting from s_1. We also define the **Q-function** according to

$$Q^\pi(s, a) = R(s) + \gamma \mathrm{E}_{s' \sim P_{sa}(\cdot)}\left[V^\pi(s')\right]$$

(where the notation $s' \sim P_{sa}(\cdot)$ means the expectation is with respect to s' distributed according to $P_{sa}(\cdot)$). The **optimal value function** is $V^*(s) = \sup_\pi V^\pi(s)$ and the **optimal Q-function** is $Q^*(s, a) = \sup_\pi Q^\pi(s, a)$.

For discrete, finite spaces, all these functions can be represented as vectors indexed by state, for which we adopt boldface notation. More precisely, fix some enumeration from 1 to N of the finite state space S. The rewards may then be written as an N-dimensional vector $\boldsymbol{R}$, whose ith element is the reward at the ith state of the MDP. Similarly, $\boldsymbol{V}^\pi$ is a vector whose ith element is the value function for π evaluated at state i. For each action a, we also let $\boldsymbol{P}_a$ be the N-by-N matrix such that its (i,j) element gives the probability of transitioning to state j upon taking action a in state i. Finally, we let the symbols $\prec$ and $\preceq$ denote strict and non-strict vectorial inequality—i.e., $\boldsymbol{x} \prec \boldsymbol{y}$ if and only if $\forall i \ \boldsymbol{x}_i < \boldsymbol{y}_i$.

The goal of standard reinforcement learning is to find a policy π such that $V^\pi(s)$ is maximized. And indeed, it can be shown (see, e.g. (Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996)) that there does exist at least one **optimal policy** π^* such that $V^\pi(s)$ is simultaneously maximized for all $s \in S$ by $\pi = \pi^*$.

2.2 Basic Properties of MDPs

For our solution to the IRL problem, we will need two of the classical results concerning MDPs (Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996).

Theorem 1 (Bellman Equations) *Let an MDP $M = (S, A, \{P_{sa}\}, \gamma, R)$ and a policy $\pi : S \mapsto A$ be given. Then, for all $s \in S, a \in A$, V^π and Q^π satisfy*

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s') V^\pi(s') \qquad (1)$$

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s') V^\pi(s') \qquad (2)$$

Theorem 2 (Bellman Optimality) *Let an MDP $M = (S, A, \{P_{sa}\}, \gamma, R)$ and a policy $\pi : S \mapsto A$ be given. Then π is an optimal policy for M if and only if, for all $s \in S$,*

$$\pi(s) \in \arg\max_{a \in A} Q^\pi(s,a) \qquad (3)$$

2.3 Inverse Reinforcement Learning

The inverse reinforcement learning problem is to find a reward function that can explain observed behaviour. We begin with the simple case where the state and action spaces are finite, the model is known, and the complete policy is observed. More precisely, then, we are given a finite state space S, a set of k actions $A = \{a_1, \ldots, a_k\}$, transition probabilities $\{P_{sa}\}$, a discount factor γ, and a policy π; the problem is to find the set of possible reward functions R such that π is an optimal policy in $(S, A, \{P_{sa}\}, \gamma, R)$. (We may then wish to identify functions within this set satisfying additional criteria.) By renaming actions if necessary, we will assume without loss of generality that $\pi(s) \equiv a_1$. This trick is used only to simplify our notation.

3. IRL in Finite State Spaces

In this section, we give a simple characterization of the set of all reward functions for which a given policy is optimal. We then show that the set contains many degenerate solutions and propose a simple heuristic for removing this degeneracy, resulting in a linear programming solution to the IRL problem.

3.1 Characterization of the Solution Set

Our main result characterizing the set of solutions is the following:

Theorem 3 *Let a finite state space S, a set of actions $A = \{a_1, \ldots, a_k\}$, transition probability matrices $\{\boldsymbol{P}_a\}$, and a discount factor $\gamma \in (0,1)$ be given. Then the policy π given by $\pi(s) \equiv a_1$ is optimal if and only if, for all $a = a_2, \ldots, a_k$, the reward $\boldsymbol{R}$ satisfies*

$$(\boldsymbol{P}_{a_1} - \boldsymbol{P}_a)(\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \succeq 0 \qquad (4)$$

Proof. Since $\pi(s) \equiv a_1$, Equation (1) may be written $\boldsymbol{V}^\pi = \boldsymbol{R} + \gamma \boldsymbol{P}_{a_1} \boldsymbol{V}^\pi$. Thus,[1]

$$\boldsymbol{V}^\pi = (\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \qquad (5)$$

Substituting Equation (2) into (3) from Theorem 2, we see that $\pi \equiv a_1$ is optimal if and only if

$$\begin{aligned}
& a_1 \equiv \pi(s) \in \arg\max_{a \in A} \sum_{s'} P_{sa}(s') V^\pi(s') \quad \forall\, s \in S \\
\Leftrightarrow\ & \sum_{s'} P_{sa_1}(s') V^\pi(s') \\
& \qquad \geq \sum_{s'} P_{sa}(s') V^\pi(s') \quad \forall\, s \in S, a \in A \\
\Leftrightarrow\ & \boldsymbol{P}_{a_1} \boldsymbol{V}^\pi \succeq \boldsymbol{P}_a \boldsymbol{V}^\pi \qquad \forall\, a \in A \setminus a_1 \\
\Leftrightarrow\ & \boldsymbol{P}_{a_1} (\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \\
& \qquad \succeq \boldsymbol{P}_a (\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \quad \forall\, a \in A \setminus a_1
\end{aligned}$$

where the last implication in this derivation used Equation (5). This completes the proof. □

Remark. Using a very similar argument, it is easy to show (essentially by replacing all inequalities in the proof above with strict inequalities) that the condition $(\boldsymbol{P}_{a_1} - \boldsymbol{P}_a)(\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \succ 0$ is necessary and sufficient for $\pi \equiv a_1$ to be the *unique* optimal policy.

For finite-state MDPs, this result characterizes the set of all reinforcement functions that are solutions to the inverse reinforcement learning problem. However, we immediately see two problems: First, $\boldsymbol{R} = 0$ (and indeed any other constant vector) is always a solution—if

[1] Here, $\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1}$ is always invertible. To see this, first note that $\boldsymbol{P}_{a_1}$, being a transition matrix, has all eigenvalues in the unit circle in the complex plane. Since $\gamma < 1$, this implies the matrix $\gamma \boldsymbol{P}_{a_1}$ has all eigenvalues in the interior of the unit circle (and in particular that 1 is not an eigenvalue). This means $\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1}$ has no zero eigenvalues, and is thus not singular.

the reward is the same no matter what action we take, then any policy, including $\pi \equiv a_1$, is optimal. Demanding that π be the unique optimal policy would alleviate this problem, but is not entirely satisfying since usually some reward vectors arbitrarily close to 0 would still be solutions. Second, for most MDPs, it also seems likely that there are many choices of $\boldsymbol{R}$ that meet the criteria (4). How do we decide which one of these many reinforcement functions to choose? The answer to these problems is not to be found in the original statement of the IRL problem; but in the next section, we describe some natural criteria that will suggest solutions to both of these problems.

3.2 LP Formulation and Penalty Terms

Clearly, linear programming can be used to find a feasible point of the constraints in Equation (4). But as discussed in the previous section, some of these points may be less "meaningful" than others, and we desire to find some way to choose between solutions satisfying Equation (4). The proposals outlined in this section were to a large extent chosen because they can be incorporated into a linear program, but nonetheless should seem fairly natural.

One natural way to choose R is to first demand that it makes π optimal (and hence solves the IRL problem), and moreover to favor solutions that make any single-step deviation from π as costly as possible. Thus, of all the functions R satisfying (4) (and, $|R(s)| \leq R_{\max}\ \forall s$), we might choose one so as to maximize

$$\sum_{s \in S} \left(Q^\pi(s, a_1) - \max_{a \in A \setminus a_1} Q^\pi(s, a) \right) \qquad (6)$$

In other words, we seek to maximize the sum of the differences between the quality of the optimal action and the quality of the next-best action. (Other criteria, such as $\sum_{s \in S} \sum_{a \in A \setminus a_1} Q^\pi(s, a_1) - Q^\pi(s, a)$ are also possible, but for the sake of concreteness, let us remain with (6) for now.)

In addition, if we believe that, all other things being equal, solutions with mainly small rewards are "simpler" and therefore preferable, we may optionally add to the objective function a weight decay-like penalty term such as $-\lambda||R||_1$, where λ is an adjustable penalty coefficient that balances between the twin goals of having small reinforcements, and of maximizing (6). A side-effect of using such an ℓ_1-penalty term is that, for sufficiently large λ, R will often be nonzero in only a few states, consistent with our idea of a "simple" reward function. Moreover, while it is common practice in many applications to hand-tune penalty coefficients, it can also be shown (assuming the solution is not already degenerate at $\lambda = 0$) that as λ is increased, there will be a phase transition at some point λ_0, such that the optimal R is bounded away from 0 for $\lambda < \lambda_0$, and $R = 0$ for $\lambda > \lambda_0$. Thus, if we wanted to choose λ automatically, $\lambda = \lambda_0^-$ (a value just before the phase transition, perhaps found via binary search on λ) would be an appealing choice, since it is gives the "simplest" R (largest penalty coefficient) such that R is not zero everywhere (and in particular so that R does at least partially "explain" why π is optimal).

Putting it all together, our optimization problem is:

$$\begin{aligned}
\text{maximize} \quad & \textstyle\sum_{i=1}^N \min_{a \in \{a_2, \ldots, a_k\}} \{ (\boldsymbol{P}_{a_1}(i) - \boldsymbol{P}_a(i)) \\
& \qquad (\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \} - \lambda ||\boldsymbol{R}||_1 \\
\text{s.t.} \quad & (\boldsymbol{P}_{a_1} - \boldsymbol{P}_a)(\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \succeq 0 \\
& \qquad\qquad\qquad\qquad \forall a \in A \setminus a_1 \\
& |\boldsymbol{R}_i| \leq R_{\max}, \;\; i = 1, \ldots, N
\end{aligned}$$

where $\boldsymbol{P}_a(i)$ denotes the ith row of $\boldsymbol{P}_a$. Clearly, this may easily be formulated as a linear program and solved efficiently. Section 6 reports on simple experiments using this algorithm.

4. Linear Function Approximation in Large State Spaces

We now consider the case of infinite state spaces. Apart from some measure-theoretic assumptions and minor regularity conditions (which we will ignore in this paper), infinite-state MDPs may be defined in much the same way as finite-state MDPs were in Section 2. For the sake of concreteness, we restrict ourselves to the case of $S = \mathbb{R}^n$. We will assume the availability of a subroutine for approximating the value of a policy, V^π, for any particular MDP.

In this setting, the reward function R is now a function from $S = \mathbb{R}^n$ into the reals, and a general solution to inverse reinforcement learning would require working with this space of all functions $\mathbb{R}^n \mapsto \mathbb{R}$. While the calculus of variations does give us some tools for optimizing over this space, it is often difficult to work with algorithmically. Hence, we choose instead to use a linear approximation for the reward function, expressing R according to

$$R(s) = \alpha_1 \phi_1(s) + \alpha_2 \phi_2(s) + \cdots + \alpha_d \phi_d(s) \qquad (8)$$

where $\phi_1, \ldots, \phi_d$ are d fixed, known, bounded basis functions mapping from S into $\mathbb{R}$, and the α_i are the unknown parameters that we want to "fit."

Since R is again linear in the variables being optimized, it is no surprise that a linear programming formulation applies here as well. Let V_i^π denote the value function of the policy π in the MDP when the reward function is $R = \phi_i$. By the linearity of expectations, the value function when the reward function R is given by

Equation (8) is therefore

$$V^\pi = \alpha_1 V_1^\pi + \cdots + \alpha_d V_d^\pi . \tag{9}$$

Using this fact and Theorem 2, the reader may easily verify (using essentially the argument in Theorem 3's proof,) that for R to make the policy $\pi(s) \equiv a_1$ optimal, the appropriate generalization of (4) is the condition that

$$\mathrm{E}_{s' \sim P_{sa_1}}\left[V^\pi(s')\right] \geq \mathrm{E}_{s' \sim P_{sa}}\left[V^\pi(s')\right] \tag{10}$$

for all states s and all actions $a \in A \setminus a_1$. From Equation (9), we know $V^\pi(s)$ is linear in the coefficients α_i. Hence, we have a set of linear constraints on the α_is.

There are however two problems with the current formulation. The first is that, for infinite state spaces, there are infinitely many constraints of the form in Equation (10), making it hard or impossible to check them all. Algorithmically, we circumvent this problem by sampling only a large but finite subset S_0 of the states, and using this constraint only at those states $s \in S_0$. The second problem, which is a more subtle one, is that since we have restricted ourselves to use the linear function approximator in Equation (8) to express R, we may no longer be able to express any reward function (other than the trivial $R = 0$) for which π is optimal. Nevertheless, even in this case, we would like to do as well as we can using the linear function approximator class, and so as a compromise, we may be willing to relax some of the constraints (10), paying a penalty when they are violated.

Our final linear programming formulation is then:

$$\begin{aligned} \text{maximize } & \textstyle\sum_{s \in S_0} \min_{a \in \{a_2, \ldots, a_k\}} \{ \\ & \quad p(\mathrm{E}_{s' \sim P_{sa_1}}\left[V^\pi(s')\right] - \mathrm{E}_{s' \sim P_{sa}}\left[V^\pi(s')\right])\} \\ \text{s.t. } & |\alpha_i| \leq 1, \;\; i = 1, \ldots, d \end{aligned}$$

where we remind the reader that V^π is an implicit function of the α_is as given by Equation (9), and S_0 is the subsample of states. Here, p is given by $p(x) = x$ if $x \geq 0$, $p(x) = 2x$ otherwise, and penalizes violations of the constraints (10) (where 2 is penalty weight that was heuristically chosen; this was a parameter to which our results did not seem very sensitive, with moderately larger values usually giving quite similar results).

5. IRL from Sampled Trajectories

This section addresses the IRL problem for the more realistic case where we have access to the policy π only through a set of actual trajectories in the state space. For this, we also do not require an explicit model of the MDP, though we do assume the ability to find an optimal policy under any reward of our choice.

We fix some initial state distribution D, and assume that for the (unknown) policy π, our goal is to find R such that π maximizes $\mathrm{E}_{s_0 \sim D}[V^\pi(s_0)]$ (where the subscript of the expectation denotes that the expectation is with respect to s_0 drawn according to D). To simplify notation, we'll assume that there is only one fixed start state s_0. (This is in fact w.l.o.g., since s_0 can be a "dummy" state whose next-state distribution under any action is D.) As with the previous algorithm for infinite state spaces, we assume R will be expressed using a linear function-approximator class.

We assume that we have the ability to simulate trajectories in the MDP (from the initial state s_0) under the optimal policy, or under any policy of our choice. For each policy we will consider (including the optimal one), we will need a way of estimating $V^\pi(s_0)$ for any setting of the α_is. To do this, we first execute m Monte Carlo trajectories under π. Then, for each $i = 1, \ldots, d$, define $\hat{V}_i^\pi(s_0)$ to be what the average empirical return would have been on these m trajectories if the reward had been $R = \phi_i$. For example, if we take only $m = 1$ trajectories, and if that trajectory visited the sequence of states $(s_0, s_1, \ldots)$, then we have:

$$\hat{V}_i^\pi(s_0) = \phi_i(s_0) + \gamma\phi_i(s_1) + \gamma^2\phi_i(s_2) + \cdots$$

In general, $\hat{V}_i^\pi(s_0)$ would be the average over the empirical returns of m such trajectories.[2] Then, for any setting of the α_is, a natural estimate of $V_i^\pi(s_0)$ is:

$$\hat{V}^\pi(s_0) = \alpha_1 \hat{V}_1^\pi(s_0) + \cdots + \alpha_d \hat{V}_d^\pi(s_0) \tag{11}$$

As in the previous algorithm's derivation, this is justified by the fact that $V^\pi(s_0) = \alpha_1 V_1^\pi(s_0) + \cdots + \alpha_d V_d^\pi(s_0)$. We now describe the algorithm.

To start off the algorithm, we first find value estimates as described above for π^*, and for the "base case" policy π_1, which is in our case a randomly chosen policy.

The "inductive step" of the algorithm is as follows: We have some set of policies $\{\pi_1, \ldots, \pi_k\}$, and want to find a setting of the α_is so that the resulting reward function (hopefully) satisfies

$$V^{\pi^*}(s_0) \geq V^{\pi_i}(s_0), \;\; i = 1, \ldots, k \tag{12}$$

As in the previous algorithm, we modify the objective

[2]In practice, we also truncate the trajectories after a large but finite number H of steps. Because of discounting, this introduces only a small error into the approximation; for example, if $H = H_\epsilon = \log_\gamma(\epsilon(1-\gamma)/R_{\max})$, the ϵ-horizon time, then this truncation introduces at most ϵ error into the estimates. If one is unhappy with this approximation, there is also a way to execute only a finite-length trajectory of expected length $O(H_\epsilon)$, but so that we still obtain an unbiased estimate of the true infinite-horizon reward (Kearns et al., 1999); that method can also be used here.

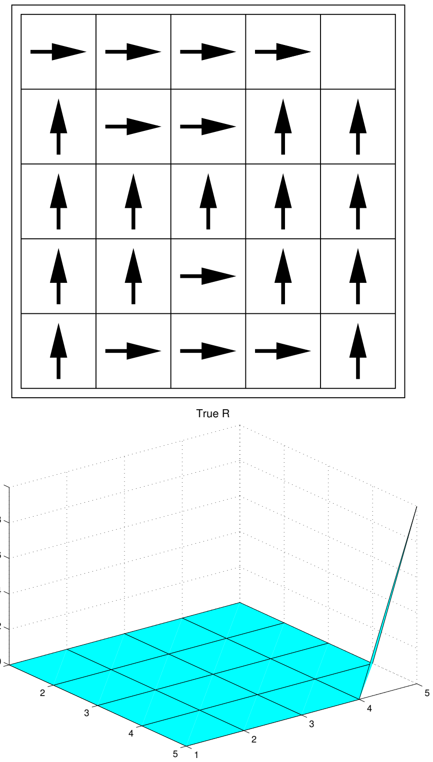

Figure 1. Top: 5x5 grid world with optimal policy. *Bottom*: True reward function.

slightly, so that the optimization becomes:

$$\begin{aligned} \text{maximize} \qquad & \sum_{i=1}^{k} p\left(\hat{V}^{\pi^*}(s_0) - \hat{V}^{\pi_i}(s_0)\right) \\ & \text{s.t. } |\alpha_i| \leq 1, \quad i = 1, \ldots, d \end{aligned}$$

where, as before, $p(x) = x$ if $x \geq 0$, and $p(x) = 2x$ if $x < 0$, so that violations of the constraints (12) are penalized (where 2 is, once more, a heuristically chosen parameter, to which our results again did not seem extremely sensitive). Note that $\hat{V}^{\pi_i}(s_0)$ and $\hat{V}^{\pi^*}(s_0)$ above are just (implicit) linear functions of the α_is as given in Equation (11), and hence this problem is easily solved via linear programming.

The above optimization gives a new setting of the α_is and hence a new reward function $R = \alpha_1\phi_1 + \cdots \alpha_d\phi_d$. We then find a policy π_{k+1} that maximizes $V^{\pi}(s_0)$ under R, add π_{k+1} to the current set of policies, and continue (for some large number of iterations, until we find an R with which we are "satisfied").

6. Experiments

In our first experiment, we used a 5×5 grid world where the agent starts from the lower-left grid square, and has to make its way to the (absorbing) upper-right grid square, whereupon it receives a reward of 1. The actions correspond to trying to move in the four compass directions, but are noisy and have a 30% chance of resulting in moving in a random direction instead. An optimal policy is shown at top in Figure 1, together with the true reward function. The inverse reinforcement problem is that of recovering the reward structure given the policy and problem dynamics.

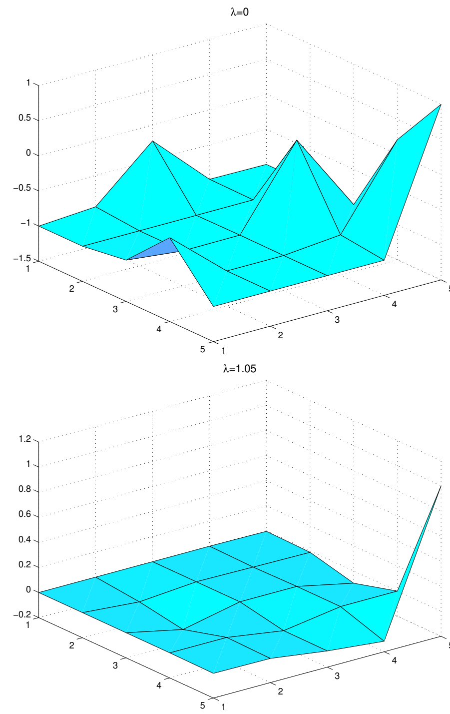

Figure 2. Inverse RL on the 5×5 grid. *Top*: $\lambda = 0$. *Bottom*: $\lambda = 1.05$.

Running the algorithm described in Section 3.2 with no penalty term, we obtain the reward function shown in Figure 2 (top). While it has clearly recovered most of the reward structure, it is still slightly "bumpy." Some of this bumpiness is hard to avoid, and comes from arbitrary symmetry-breaking in the chosen policy. However, with the penalty coefficient λ set to a value just below the phase transition as discussed earlier, we obtain the second reward function in Figure 2, which is very close to the true reward.[3]

Our next experiment was run on the well-known "mountain–car" task, a cartoon of which is shown in Figure 3. The true, undiscounted, reward is -1 per-step until we reach the goal at the top of the hill, and the state is the car's x-position and velocity. Since the state space is continuous, we used the version of our algorithm described in Section 4. We chose the function approximator class for the reward to be functions

[3] Interestingly, intermediate values of λ such as 0.5 did not give "smooth" looking functions at all. In retrospect, this is not too surprising: small λ results in many values near ± 1; large λ results in many values near 0; and intermediate λ has a mix of the two.

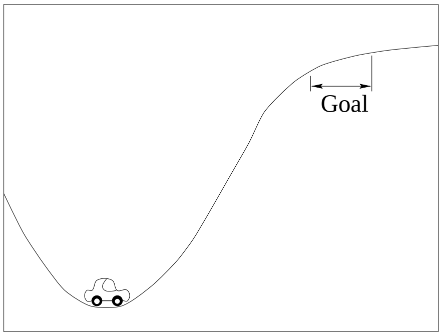

Figure 3. Cartoon of the mountain–car problem. (Not to scale.)

of the car x-position only, with the class consisting of all linear combinations of 26 evenly spaced Gaussian-shaped basis functions. Giving the optimal policy[4] to our algorithm, a typical reward function found by it is shown in Figure 4a. (Note the scale on the y-axis.) Clearly, the solution has nearly perfectly captured the $R = -c$ structure of the reward.

For a more challenging problem, we also reran the experiment with the true reward changed to be 1 in an interval [-0.72, -0.32] centered around the bottom of the hill and 0 everywhere else, and $\gamma = 0.99$. In this problem, the optimal policy is to go as quickly as possible to the bottom of the hill and park there. (This is not always possible because if, for example, we are near the top of the hill on the right and moving too quickly, then we may shoot off the right end of the hill and enter the absorbing state no matter how hard we braked.) Running our algorithm on this new problem, a typical solution is shown in Figure 4b. By and large, it has successfully recovered the main structure of the reward being large and positive around the specified interval; it also has an artifact on the right side, we believe from the effect of unavoidably "shooting off" the right end sometimes. Nevertheless, we think the solution shown is a fairly good one for the problem.

Our final experiment applied the sample-based algorithm to a *continuous* version of the 5×5 grid world. More precisely, the state was $[0,1] \times [0,1]$, and the effect of each of the four compass-direction actions is to move the agent 0.2 in the intended direction, after which uniform noise in $[-0.1, 0.1]$ is added to each coordinate, and the state is finally truncated if necessary to keep it within the unit square. The true reward was 1 in the (non-absorbing) square $[0.8, 1] \times [0.8, 1]$, and 0 everywhere else, and $\gamma = 0.9$. The function approx-

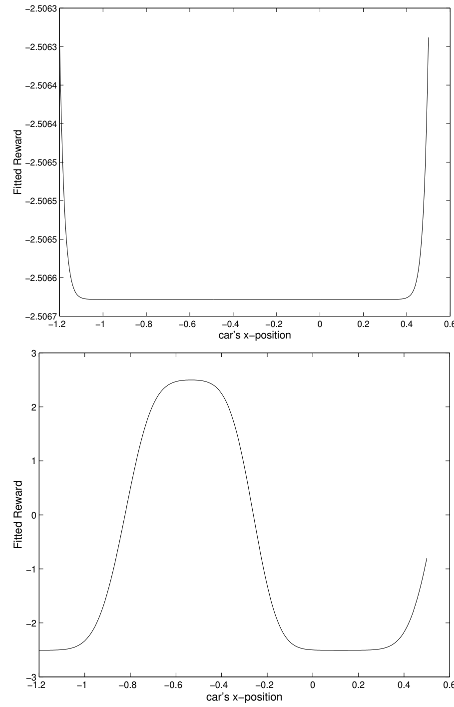

Figure 4. Typical solutions found by IRL for the mountain–car. *Top*: Original problem (note scale on y-axis). *Bottom*: Problem of parking at bottom of hill.

imator class consisted of all linear combinations of a 15×15 array of two-dimensional Gaussian basis functions. The initial state distribution D was uniform over the state space, and our algorithm was run using $m = 5000$ trajectories, each of 30 steps, to evaluate each policy. When needed (such as to find the "optimal" policy for comparison), the MDP was solved based on a 50×50 discretization of the state space.

Running this experiment, the solution found by our algorithm was usually already reasonable after just 1 iteration, and by about 15 iterations, the algorithm had usually settled on fairly good solutions. We compared the fitted reward's optimal policy with the true optimal policy, calculating the fraction of the state space on which their action choices disagree (Figure 5, top). We found discrepancies typically between 3% and 10%; with many distinct near-optimal policies, such variation is to be expected. Perhaps a more appropiate measure of our algorithm's performance is to compare the *quality* of the fitted reward's optimal policy with the *quality* of the true optimal policy. (Quality is of course measured using the *true* reward function!) Usually by about 15 iterations of the algorithm, our evaluations (which used 50000 Monte Carlo trials of 50 steps each) were unable to detect a statistically significant difference between the value of the true "optimal policy" (about 6.65) and the value of the fitted reward's optimal policy (Figure 5, bottom).

[4] This is as determined by a fine 120x120 discretization of the state space. The functions V_i^π, needed by the algorithm, were also found this way. To run the algorithm, we used a sample of states of size $|S_0| = 5000$, not counting states that did not give nontrivial constraints.

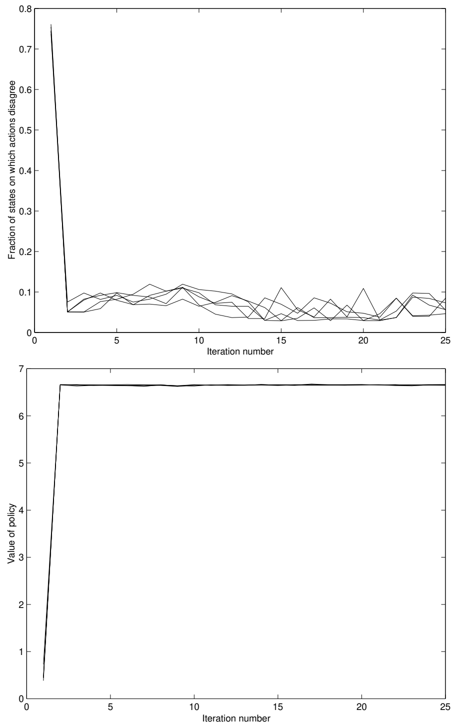

Figure 5. Results on the continuous grid world, for 5 runs. *Top*: Fraction of states on which the fitted reward's optimal policy disagrees with the true optimal policy, plotted against iteration number. *Bottom*: The value of the fitted reward's optimal policy. (Estimates are from 50000 Monte Carlo trials of length 50 each; negligible error-bars).

7. Conclusions and Future work

Our results show that the inverse reinforcement learning problem is soluble, at least for moderate-sized discrete and continuous domains. A number of open questions remain to be addressed:

- Potential-based shaping rewards (Ng et al., 1999) can produce reward functions that make it dramatically easier to learn a solution to an MDP, without affecting optimality. Can we design IRL algorithms that recover "easy" reward functions?

- In real-world empirical applications of IRL, there may be substantial noise in the *observer's* measurements of the *agent's* sensor inputs and actions; moreover, the agent's own action selection process may be noisy and/or suboptimal. Finally, there may be many optimal policies, of which only a few are observed. What are appropriate metrics for fitting such data?

- If behaviour is strongly inconsistent with optimality, can we identify "locally consistent" reward functions for specific regions in state space?

- How can experiments be designed to maximize the identifiability of the reward function?

- How well does our algorithmic approach carry to the case of partially observable environments?

Acknowledgments

A. Ng is supported by a Berkeley fellowship. This work was also supported by NSF ECS-9873474.

References

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

Boyd, S., Ghaoui, L. E., Feron, E., & Balakrishnan, V. (1994). *Linear matrix inequalities in system and control theory*. SIAM.

Doya, K., & Sejnowski, T. (1995). A novel reinforcement model of birdsong vocalization learning. *Advances in Neural Information Processing Systems 7* (pp. 101–108). Denver, CO: MIT Press.

Kearns, M., Mansour, Y., & Ng, A. Y. (1999). Approximate planning in large POMDPs via reusable trajectories. (extended version).

Keeney, R. L., & Raiffa, H. (1976). *Decisions with multiple objectives: Preferences and value tradeoffs*. New York: Wiley.

Montague, P. R., Dayan, P., Person, C., & Sejnowski, T. J. (1995). Bee foraging in uncertain environments using predictive hebbian learning. *Nature*, *377*, 725–728.

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

Russell, S. (1998). Learning agents for uncertain environments (extended abstract). *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*. ACM Press.

Rust, J. (1994). Do people behave according to Bellman's principal of optimality? Submitted to Journal of Economic Perspectives.

Sargent, T. J. (1978). Estimation of dynamic labor demand schedules under rational expectations. *Journal of Political Economy*, *86*, 1009–1044.

Schmajuk, N. A., & Zanutto, B. S. (1997). Escape, avoidance, and imitation: a neural network approach. *Adaptive Behavior*, *6*, 63–129.

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

Touretzky, D. S., & Saksida, L. M. (1997). Operant conditioning in Skinnerbots. *Adaptive Behavior*, *5*, 219–47.

Watkins, C. J. (1989). *Models of delayed reinforcement learning*. Doctoral dissertation, Psychology Department, Cambridge University, Cambridge, United Kingdom.