Active Learning for Reward Estimation in Inverse Reinforcement Learning*

Manuel Lopes[1], Francisco Melo[2], and Luis Montesano[3]

[1] Instituto de Sistemas e Robótica - Instituto Superior Técnico
Lisboa, Portugal
macl@isr.ist.utl.pt
[2] Carnegie Mellon University
Pittsburgh, PA, USA
fmelo@cs.cmu.edu
[3] Universidad de Zaragoza
Zaragoza, Spain
lmontesa@unizar.es

Abstract. Inverse reinforcement learning addresses the general problem of recovering a reward function from samples of a policy provided by an expert/demonstrator. In this paper, we introduce *active learning* for inverse reinforcement learning. We propose an algorithm that allows the agent to *query* the demonstrator for samples at specific states, instead of relying only on samples provided at "arbitrary" states. The purpose of our algorithm is to estimate the reward function with similar accuracy as other methods from the literature while reducing the amount of policy samples required from the expert. We also discuss the use of our algorithm in higher dimensional problems, using both Monte Carlo and gradient methods. We present illustrative results of our algorithm in several simulated examples of different complexities.

1 Introduction

We address the general problem of *learning from demonstration*. In this class of problems, an agent is given a set of sample situation-action pairs by a demonstrator, from which it must recover the overall demonstrated behavior and/or corresponding task description. In this paper we are particularly interested in recovering the task description. In other words, the agent infers the underlying task that the demonstrator is trying to solve. From this task description, the agent can then construct its own policy to solve the recovered task. One interesting aspect of this approach is that it can accommodate for differences between the demonstrator and the learner [1]. The learner is not just replicating the observed trajectory, but is inferring the "reason" behind such behavior.

* Work partially supported by the ICTI and FCT, under the CMU-Portugal Program, the (POS_C) program that includes FEDER funds and the projects PTDC/EEA-ACR/70174/2006, (FP6-IST-004370) RobotCub and (FP7-231640) Handle.

We formalize our problem using Markov decision processes (MDP). Within this formalism, the demonstration consists of a set of state-action pairs and the compact task representation takes the form of a *reward function*. Learning from demonstration in MDPs has been explored in different ways in the literature [2–4], and is usually known as *inverse reinforcement learning*. The seminal paper [3] gives the first formal treatment of inverse reinforcement learning as well as several algorithms to compute a reward description from a demonstration. This problem has since been addressed in several other works [4–8].

The general IRL problem poses several interesting challenges to be dealt with. On one hand, the process of searching for the "right" reward function typically requires the underlying MDP to be solved multiple times, making this process potentially computationally expensive in large problems. Furthermore, it is unreasonable to assume that the desired policy is completely specified, as this is impractical in problems with more than a few dozen states, or that there is no noise in the demonstration. Finally, the IRL problem is *ill-posed*, in the sense that there is not a single reward function that renders a given policy optimal and also there are usually multiple optimal policies for the same reward function [9]. This means that, even if the desired policy is completely specified to the learner, the problem remains ill-posed, as additional criteria are necessary to disambiguate between multiple rewards yielding the same optimal policy.

Probabilistic sample-based approaches to the IRL problem [4–6] partly address these issues, alleviating the requirement for complete and correct demonstration while restricting the set of possible solution rewards. These approaches allow the solution to IRL to be "better conditioned" by increasing the size of the demonstration and are robust to suboptimal actions in the demonstration.[4] However, this will typically require a large amount of data (samples) for a good estimate of the reward function to be recovered.

In this paper we propose the use of *active learning* to partly mitigate the need for large amounts of data during learning. We adopt a Bayesian approach to IRL, following [6]. The idea behind active learning is to reduce the data requirements of learning algorithms by actively selecting potentially informative samples, in contrast with random sampling from a predefined distribution [10]. In our case we use this idea to reduce the number of samples required from the expert, and only ask the expert to demonstrate the desired behavior at the most informative states. We compute the posterior distribution over possible reward functions and use this information to *actively select* the states whose action the expert should provide. Experimental results show that our approach generally reduces the amount of data required to learn the reward function. Also, it is more adequate in terms of interaction with the expert, as it requires the expert to illustrate the desired behavior on fewer instances.

[4] By considering demonstrations in which suboptimal actions can also be sampled, the learner is provided with a *ranking* of actions instead of just an indication of the optimal actions. Demonstrations are thus "more informative", enforcing a more constrained set of possible reward functions than in the general case where only optimal policies are provided. This, in turn, simplifies the search problem.

2 Background

In this section we review some background material on MDPs and inverse reinforcement learning.

2.1 Markov Decision Processes

A *Markov decision process* (MDP) is a tuple $(\mathcal{X}, \mathcal{A}, \mathsf{P}, r, \gamma)$, where $\mathcal{X}$ represents the finite state-space, $\mathcal{A}$ the finite action space, P the transition probabilities, r the reward function and is γ a discount factor. $\mathsf{P}_a(x, y)$ denotes the probability of transitioning from state x to state y when action a is taken. The purpose of the agent is to choose the action sequence $\{A_t\}$ maximizing

$$V(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(X_t, A_t) \mid X_0 = x\right].$$

A *policy* is a mapping $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$, where $\pi(x, a)$ is the probability of choosing action $a \in \mathcal{A}$ in state $x \in \mathcal{X}$. Associated with any such policy there is a *value-function* V^π, $V^\pi(x) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(X_t, A_t) \mid X_0 = x\right]$,where the expectation is now taken with respect to policy π. For any given MDP there exists at least one policy π^* such that

$$V^{\pi^*}(x) \geq V^\pi(x).$$

Any such policy is an *optimal policy* for that MDP and the corresponding value function is denoted by V^*.

Given any policy π, the following recursion holds

$$V^\pi(x) = r_\pi(x) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}_\pi(x, y) V^\pi(y)$$

where $\mathsf{P}_\pi(x, y) = \sum_{a \in \mathcal{A}} \pi(x, a) \mathsf{P}_a(x, y)$ and $r_\pi(x) = \sum_{a \in \mathcal{A}} \pi(x, a) r(x, a)$. For the particular case of the optimal policy π^*, the above recursion becomes

$$V^*(x) = \max_{a \in \mathcal{A}} \left[r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}_a(x, y) V^*(y)\right].$$

We also define the Q-function associated with a policy π as

$$Q^\pi(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}_a(x, y) V^\pi(y)$$

Sometimes, it will be convenient to write the above expressions using vector notation, leading to the expressions

$$\mathbf{V}^\pi = \mathbf{r}_\pi + \gamma \mathsf{P}_\pi \mathbf{V}^\pi \qquad \mathbf{Q}_a^\pi = \mathbf{r}_a + \gamma \mathsf{P}_a \mathbf{V}^\pi \tag{1a}$$

$$\mathbf{V}^* = \max_{a \in \mathcal{A}} [\mathbf{r}_a + \gamma \mathsf{P}_a \mathbf{V}^*] \qquad \mathbf{Q}_a^* = \mathbf{r}_a + \gamma \mathsf{P}_a \mathbf{V}^*, \tag{1b}$$

where $\mathbf{r}_a$ and $\mathbf{Q}_a$ denote the ath columns of matrices $\mathbf{r}$ and $\mathbf{Q}$, respectively.

2.2 Bayesian Inverse Reinforcement Learning

As seen above, an MDP describes a sequential decision making problem in which an agent must choose its actions so as to maximize the total discounted reward. In this sense, the reward function in an MDP encodes the *task* of the agent.

Inverse reinforcement learning (IRL) deals with the problem of recovering the task representation (*i.e.*, the reward function) given a demonstration of the task to be performed (*i.e.*, the desired policy). In this paper, similarly to [6], IRL is cast as an *inference problem*, in which the agent is provided with a noisy sample of the desired policy from which it must estimate a reward function explaining the policy.

Our working assumption is that there is one reward function, r_{target}, that the demonstrator wants the agent to maximize. We denote the corresponding optimal Q-function by Q^*_{target}. Given this reward function, the demonstrator will choose an action $a \in \mathcal{A}$ in state $x \in \mathcal{X}$ with probability

$$\mathbb{P}\left[A_{\text{demo}} = a \mid X_{\text{demo}} = x, r_{\text{target}}\right] = \frac{e^{\eta Q^*_{\text{target}}(x,a)}}{\sum_{b \in \mathcal{A}} e^{\eta Q^*_{\text{target}}(x,b)}}, \tag{2}$$

where η is a non-negative constant.

We consider the demonstration as a sequence $\mathcal{D}$ of state-action pairs,

$$\mathcal{D} = \left\{(x_1, a_1), (x_2, a_2), \ldots, (x_n, a_n)\right\},$$

From (2), for any given r-function, the *likelihood* of a pair $(x, a) \in \mathcal{X} \times \mathcal{A}$ is given by

$$L_r(x, a) = \mathbb{P}\left[(x, a) \mid r\right] = \frac{e^{\eta Q^*_r(x,a)}}{\sum_{b \in A} e^{\eta Q^*_r(x,b)}},$$

where we denoted by $Q^*_r(x, a)$ the optimal Q-function associated with reward r. The constant η can be seen as a *confidence parameter* that translates the confidence of the agent on the demonstration. Note that, according to the above likelihood model, evaluating the likelihood of a state-action pair given a reward r requires the computation of Q^*_r. This can be done, for example, using dynamic programming, which requires knowledge of the transition probabilities P. In the remainder of the paper, we assume these transtion probabilities are known.

Assuming independence between the state-action pairs in the demonstration, the likelihood of the demonstration $\mathcal{D}$ is

$$L_r(\mathcal{D}) = \prod_{(x_i, a_i) \in \mathcal{D}} L_r(x_i, a_i).$$

Given a prior distribution $\mathbb{P}[r]$ over the space of possible reward functions, we have

$$\mathbb{P}\left[r \mid \mathcal{D}\right] \propto L_r(\mathcal{D})\mathbb{P}\left[r\right].$$

The posterior distribution takes into account the demonstration and prior information and will provide the information used to actively select samples to be included in the demonstration. From this distribution we can extract several policies and rewards, for instance the mean policy $\pi_{\mathcal{D}}$:

$$\pi_{\mathcal{D}}(x,a) = \int \pi_r(x,a)\mathbb{P}\left[r \mid \mathcal{D}\right] dr, \tag{3}$$

or the *maximum a posteriori*

$$r^* = \max_r \mathbb{P}\left[r \mid \mathcal{D}\right], \tag{4}$$

We conclude this section by describing two methods used to address the IRL problem within this Bayesian framework.

2.3 Two Methods for Bayesian IRL

So far, we cast the IRL problem as an *inference problem*. In the continuation we describe two methods to address this inference problem, one that directly approximates the maximum given in (4) and another that estimates the complete posterior distribution $\mathbb{P}\left[r \mid \mathcal{D}\right]$.

Gradient-based IRL The first approach considers a *uniform prior* over the space of possible rewards. This means that maximizing $\mathbb{P}\left[r \mid \mathcal{D}\right]$ is equivalent to maximizing the likelihood $L_r(\mathcal{D})$. To this purpose, we implement a gradient-ascent algorithm on the space of rewards, taking advantage of the structure of the underlying MDP. A similar approach has been adopted in [4].

We start by writing the log-likelihood of the demonstration given r:

$$\Lambda_r(\mathcal{D}) = \sum_{(x_i,a_i)\in\mathcal{D}} \log(L_r(x_i,a_i)). \tag{5}$$

We can now write

$$\left[\nabla_r \Lambda_r(\mathcal{D})\right]_{xa} = \sum_{(x_i,a_i)\in\mathcal{D}} \frac{1}{L_r(x_i,a_i)} \frac{\partial L_r(x_i,a_i)}{\partial r_{xa}}. \tag{6}$$

To compute $\nabla_r L_r(x,a)$, we observe that

$$\nabla_r L_r(x,a) = \frac{dL_r}{dQ^*}(x,a)\frac{dQ^*}{dr}(x,a). \tag{7}$$

Computing the derivative of L_r with respect to each component of Q^* yields

$$\frac{dL_r}{dQ^*_{yb}}(x,a) = \eta L_r(x,a)\big(\delta_{yb}(x,a) - L_r(y,b)\delta_y(x)\big),$$

with $x, y \in \mathcal{X}$ and $a, b \in \mathcal{A}$. In the above expression, $\delta_u(v)$ denotes the Kronecker delta function.

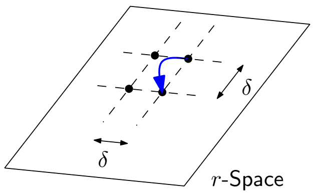

Fig. 1. Representation of the PolicyWalk variation of MCMC. We refer to [6] for details.

To compute $\frac{dQ^*}{dr}$, we recall that $\mathbf{Q}^*_a = \mathbf{r}_a + \gamma \mathsf{P}_a(\mathbf{I} - \gamma \mathsf{P}_{\pi^*})^{-1}\mathbf{r}_{\pi^*}$. We also note that, except for those points in reward space where the policy is not differentiable with respect to r — corresponding to situations in which a small change in a particular component of the reward function induces a change in a component of the policy, — *the policy remains unchanged under a small variation in the reward function*. We thus consider the approximation $\frac{dQ^*}{dr_{zu}}(x,a) \approx \frac{\partial Q^*}{\partial r_{zu}}(x,a)$ that ignores the dependence of the policy on r. The gradient estimate thus obtained corresponds to the actual gradient except near those reward functions on which the policy is not differentiable.[5] Considering the above approximation, and letting $\mathbf{T} = \mathbf{I} - \gamma \mathsf{P}_{\pi^*}$, we have

$$\frac{\partial Q^*}{\partial r_{zu}}(x,a) = \delta_{zu}(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}_a(x,y)\mathbf{T}^{-1}(y,z)\pi^*(z,u), \tag{8}$$

with $x, y, z \in \mathcal{X}$ and $a, u \in \mathcal{A}$.

Putting everything together, the method essentially proceeds by considering some initial estimate r_0 and then use the gradient computation outlined above to perform the update

$$\mathbf{r}_{t+1} = \mathbf{r}_t + \alpha_t \nabla_r \Lambda_{r_t}(\mathcal{D})$$

MCMC IRL The second approach, proposed in [6], uses the Monte-Carlo Markov chain (MCMC) algorithm to approximate the posterior $\mathbb{P}[r \mid \mathcal{D}]$. The MCMC algorithm thus generates a set of sample reward functions, $\{r_1, \ldots, r_N\}$, distributed according to the target distribution, $\mathbb{P}[r \mid \mathcal{D}]$. Then,

$$\mathbb{P}[r \mid \mathcal{D}] \approx \frac{1}{N}\sum_{i=1}^{N} \delta(r, r_i). \tag{9}$$

In the MCMC algorithm, these samples correspond to a sample trajectory of a Markov chain designed so that its invariant distribution matches the target distribution, $\mathbb{P}[r \mid \mathcal{D}]$ [11].

We implement PolicyWalk, an MCMC algorithm described in [6]. In this particular variation, the reward space is discretized into a uniform grid and

[5] It is noteworthy that Rademacher's theorem guarantees that the set of such reward functions is null-measured. We refer to [4] for further details.

the MCMC samples jump between neighboring nodes in this grid (see Fig. 1). In other words, a new sample is obtained from the current sample to one of the neighboring nodes in the grid. The new sample is accepted according to the ratio between the posterior probabilities of the current and new samples. Reward functions with higher (posterior) probability are thus selected more often than those with lower probability, and the method is guaranteed to sample according to the true posterior distribution.

A problem with this method is that, for large-dimensional problems, it generally requires a large number of sample rewards to ensure that the estimate of $\mathbb{P}[r \mid \mathcal{D}]$ is accurately represented by the sample set. We refer to the result in [6], in which the number of samples N required to ensure an estimation error bounded by ε must be $O(M^2 \log(1/\varepsilon))$, where M is the dimension of the reward function. This means that the number of samples grows (roughly) quadratically with the dimension of the reward space. Furthermore, this result assumes that $\|r\|_\infty \leq 1/M$. As noted in [6], this condition on r can be ensured by rescaling the reward function, which does not affect the optimal policy. It does, however, affect the likelihood function and, consequently, $\mathbb{P}[r \mid \mathcal{D}]$. Whenever such rescaling is not possible or convenient and the rewards can only be bounded by some value C, the previous lower-bound on the sample size roughly deteriorates to $O(M^6 e^{2M} \log(1/\varepsilon))$, which quickly becomes prohibitive for large M.

3 Active Learning for Reward Estimation

In the previous section we discussed two possible (Bayesian) approaches to the IRL problem. In these approaches, the agent is provided with a demonstration $\mathcal{D}$, consisting of pairs (x_i, a_i) of states and corresponding actions. From this demonstration the agent must identify the underlying target task.

In the active learning setting, we now assume that, after some initial batch of data $\mathcal{D}$, the agent has the possibility to *query* the expert for the action at particular states chosen by the agent. In this section we propose a criterion to select such states and discuss how this can be used within the IRL framework. We also discuss on-line versions of the methods in the previous section that are able to cope with the successive additional data provided by the expert as a result of the agent's queries.

3.1 Active Sampling

The active learning strategies presented below rely on the uncertainty about the parameter to be estimated to select new data points. As discussed in Section 2, the parameter to be estimated in the IRL setting is the task description, *i.e.*, the reward function. Unfortunately, the relation between rewards and policies is not one-to-one, making active learning in this setting more complex than in other settings.

This is easily seen by thinking of a scenario in which *all* possible reward functions give rise to the same policy in a given state x (this can be the case if there is only one action available in state x). This means that a large uncertainty in the reward function does not necessarily translate into uncertainty in terms of which action to choose in state x. Therefore, in some scenarios it may not be possible to completely disambiguate the reward function behind the demonstration.

In our setting, we have access to an estimate of the posterior distribution over the space of possible reward functions, $\mathbb{P}[r \mid \mathcal{D}]$. From this distribution, the agent should be able to choose a state and query the expert about the corresponding action in such a way that the additional sample is as useful/informative as possible. Notice that the posterior distribution, $\mathbb{P}[r \mid \mathcal{D}]$, does not differentiate between states (it is a distribution over functions) and, as such, a standard variance criterion cannot be used directly.

We are interested in finding a criterion to choose the states to query the demonstrator so as to recover the correct reward (or, at least, the optimal target behavior) while requiring significantly less data than if the agent was provided with randomly chosen state-action pairs. To this purpose, we define the set $\mathcal{R}_{xa}(p)$ as the set of reward functions r such that $\pi_r(x,a) = p$. Now for each pair $(x,a) \in \mathcal{X} \times \mathcal{A}$, the distribution $\mathbb{P}[r \mid \mathcal{D}]$ in turn induces a distribution over the possible values p for $\pi(x,a)$. This distribution can be characterized by means of the following density

$$\bar{\mu}_{xa}(p) = \mathbb{P}[\pi(x,a) = p \mid \mathcal{D}] = \mathbb{P}[r \in \mathcal{R}_{xa}(p) \mid \mathcal{D}]. \tag{10}$$

Using the above distribution, the agent can now query the demonstrator about the correct action in states where the uncertainty on the policy is larger, *i.e.*, in states where $\bar{\mu}_{xa}$ exhibits larger "spread".

One possibility is to rely on some measure of *entropy* associated with $\bar{\mu}_{xa}$. Given that $\bar{\mu}_{xa}$ corresponds to a continuous distribution, the appropriate concept is that of *differential entropy.* Unfortunately, as is well-known, differential entropy as a measure of uncertainty does not exhibit the same appealing properties as its discrete counterpart. To tackle this difficulty, we simply consider a partition of the interval $I = [0,1]$ into K subintervals I_k, with $I_k = (\frac{k}{K}, \frac{k+1}{K}]$, $k = 0, \ldots, K-1$, and $I_0 = [0, 1/K]$. We can now define a new *discrete* probability distribution

$$\mu_{xa}(k) = \mathbb{P}[\pi(x,a) \in I_k \mid \mathcal{D}] = \int_{I_k} \bar{\mu}_{xa}(p)dp, \quad k = 1, \ldots, K.$$

The distribution μ_{xa} thus defined is a discretized version of the density in (10), for which we can compute the associated (Shannon) entropy, $H(\mu_{xa})$. As such, for each state $x \in \mathcal{X}$, we define the *mean entropy* as

$$\bar{H}(x) = \frac{1}{|\mathcal{A}|}\sum_a H(\mu_{xa}) = -\frac{1}{|\mathcal{A}|}\sum_{a,k} \mu_{xa}(k)\log \mu_{xa}(k)$$

and let the agent query the expert about the action to be taken at the state x^* given by

$$x^* = \operatorname*{arg\,max}_{x \in \mathcal{X}} \bar{H}(x),$$

with ties broken arbitrarily. Given the estimate (9) for $\mathbb{P}[r \mid \mathcal{D}]$, this yields

$$\mu_{xa}(k) \approx \frac{1}{N}\sum_i \mathbb{I}_{I_k}(\pi_i(x,a)), \tag{11}$$

where π_i is the policy associated with the ith reward sampled in the MC method and $\mathbb{I}_{I_k}$ is the indicator function for the set I_k. This finally yields

$$\bar{H}(x) \approx -\frac{1}{|\mathcal{A}|N}\sum_{i,a,k}\mathbb{I}_{I_k}(\pi_i(x,a))\log\frac{\sum_i \mathbb{I}_{I_k}(\pi_i(x,a))}{N}.$$

It is worth mentioning that, in the context of IRL, there are two main sources of uncertainty in recovering the reward function. One depends on the natural ambiguity of the problem: for any particular policy, there are typically multiple reward functions that solve the IRL problem. This type of ambiguity appear even with perfect knowledge of the policy, and is therefore independent of the particular process by which states are sampled. The other source of uncertainty arises from the fact that the policy is not accurately specified in certain states. This class of ambiguity can be addressed by sampling these states until the policy is properly specified. Our entropy-based criterion does precisely this.

3.2 Active IRL

We conclude this section by describing how the active sampling strategy above can be combined with the IRL methods in Section 2.3.

Algorithm 1 General active IRL algorithm.

```
Require: Initial demo $\mathcal{D}$
1: Estimate $\mathbb{P}[r \mid \mathcal{D}]$ using general MC algorithm
2: for all $x \in \mathcal{X}$ do
3:     Compute $\bar{H}(x)$
4: end for
5: Query action for $x^* = \arg\max_x \bar{H}(x)$
6: Add new sample to $\mathcal{D}$
7: Return to 1
```

The fundamental idea is simply to use the data from an initial demonstration to compute a first posterior $\mathbb{P}[r \mid \mathcal{D}]$, use this distribution to query further states, recompute $\mathbb{P}[r \mid \mathcal{D}]$, and so on. This yields the general algorithm summarized in Algorithm 1. We note that running MCMC and recompute $\mathbb{P}[r \mid \mathcal{D}]$ at each iteration is very time consuming, even more so in large-dimensional problems. For efficiency, step 1 can take advantage of several optimizations of MCMC such as sequential and hybrid monte-carlo.

In very large dimensional spaces, however, the MC-based approach becomes computationally too expensive. We thus propose an approximation to the general Algorithm 1 that uses the gradient-based algorithm in Section 2.3. The idea

Algorithm 2 Active gradient-based IRL algorithm.

```
Require: Initial demo $\mathcal{D}$
1: Compute $r^*$ as in (4)
2: Estimate $\mathbb{P}[r \mid \mathcal{D}]$ in a neighborhood of $r^*$
3: for all $x \in \mathcal{X}$ do
4:     Compute $\bar{H}(x)$
5: end for
6: Query action for $x^* = \arg\max_x \bar{H}(x)$
7: Add new sample to $\mathcal{D}$
8: Return to 1
```

behind this method is to replace step 1 in Algorithm 1 by two steps, as seen in Algorithm 2. The algorithm thus proceeds by computing the maximum-likelihood estimate r^* as described in Section 2.3. It then uses Monte-Carlo sampling to approximate $\mathbb{P}[r \mid \mathcal{D}]$ *in a neighborhood* $B_\varepsilon(r^*)$ *of* r^* and uses this estimate to compute $H(x)$ as in Algorithm 1. The principle behind this second approach is that the policy π_{r^*} should provide a reasonable approximation to the target policy. Therefore, the algorithm can focus on estimating $\mathbb{P}[r \mid \mathcal{D}]$ only in a neighborhood of r^*. As expected, this significantly reduces the computational requirements of the algorithm.

It is worth mentioning that the first method, relying on standard MC sampling, eventually converges to the true posterior distribution as the number of samples goes to infinity. The second method first reaches a local maximum of the posterior and then only estimates the posterior around that point. If the posterior is unimodal, it is expectable that the second method brings significant advantages in computational terms; however, if the posterior is multimodal, this local approach might not be able properly represent the posterior distribution. In any case, as discussed in Section 2.3, in high-dimensional problems, the MCMC method requires a prohibitive amount of samples to provide accurate estimates, rendering such approach inviable.

4 Simulations

We now illustrate the application of the proposed algorithms in several problems of varying complexity.

4.1 Finding the Maximum of a Quadratic Function

We start by a simple problem of finding the maximum of a quadratic function. Such a problem can be described by the MDP in Fig. 2, where each state corresponds to a discrete value between -1 and 1. The state-space thus consists of 21 states and 2 actions that we denote a_l and a_r. Each action moves the agent deterministically to the contiguous state in the corresponding direction (a_l to the left, a_r to the right). For simplicity, we consider a reward function parameterized

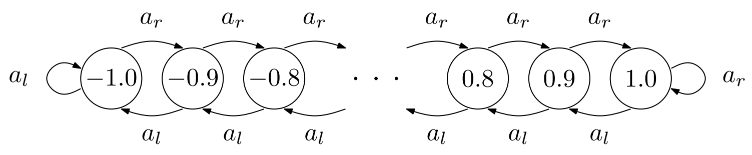

Fig. 2. Simple MDP where the agent must find the maximum of a function.

using a two-dimensional parameter vector $\boldsymbol{\theta}$, yielding

$$r(x) = \theta_1(x - \theta_2)^2,$$

corresponding to a quadratic function with a (double) zero at θ_2 and concavity given by θ_1. For the MDP thus defined, the optimal policy either moves the agent toward the state in which the maximum is attained (if $\theta_1 < 0$) or toward one of the states ± 1 (if $\theta_1 > 0$).

For our IRL problem, we consider the reward function, $r(x) = -(x - 0.15)^2$, for which the agent should learn the parameter $\boldsymbol{\theta}$ from a demonstration. The initial demonstration consisted on the optimal actions for the extreme states:

$$\mathcal{D} = \{(-1.0, a_r), (-0.9, a_r), (-0.8, a_r), (0.8, a_l), (0.9, a_l), (1.0, a_l)\}$$

and immediately establishes that $\theta_1 < 0$.

Figure 3 presents the results obtained using Algorithm 1, with the confidence parameter in the likelyhood function set to $\eta = 500$ and $N = 400$ in the MCMC estimation. The plots on the left represent the reward functions sampled in the MCMC step of the algorithm and the plots on the right the corresponding average policy $\pi_{\mathcal{D}}$. In the depicted run, the queried states were $x = 0$ at iteration 1, $x = 0.3$ at iteration 2, $x = 0.1$ at iteration 3, and $x = 0.2$ at iteration 4. It is clear from the first iteration that the initial demonstration only allows the agent to place θ_2 somewhere in the interval $[-0.8, 0.8]$ (note the spread in the sampled reward functions). Subsequent iterations show the distribution to concentrate on the true value of θ_2 (visible in the fact that the sampled rewards all exhibit a peak around the true value). Also, the policy clearly converges to the optimal policy in iteration 5 and the corresponding variance decreases to 0.

We conclude by noting that our algorithm is roughly implementing the *bisection method*, known to be an efficient method to determine the maximum of a function. This toy example provides a first illustration of our active IRL algorithm at work and the evolution of the posterior distributions over r along the iterations of the algorithm.

4.2 Puddle World

We now illustrate the application of our algorithm in a more complex problem. This problem is known in the reinforcement learning literature as the *puddle world*. The puddle world consists in a continuous-state MDP in which an agent

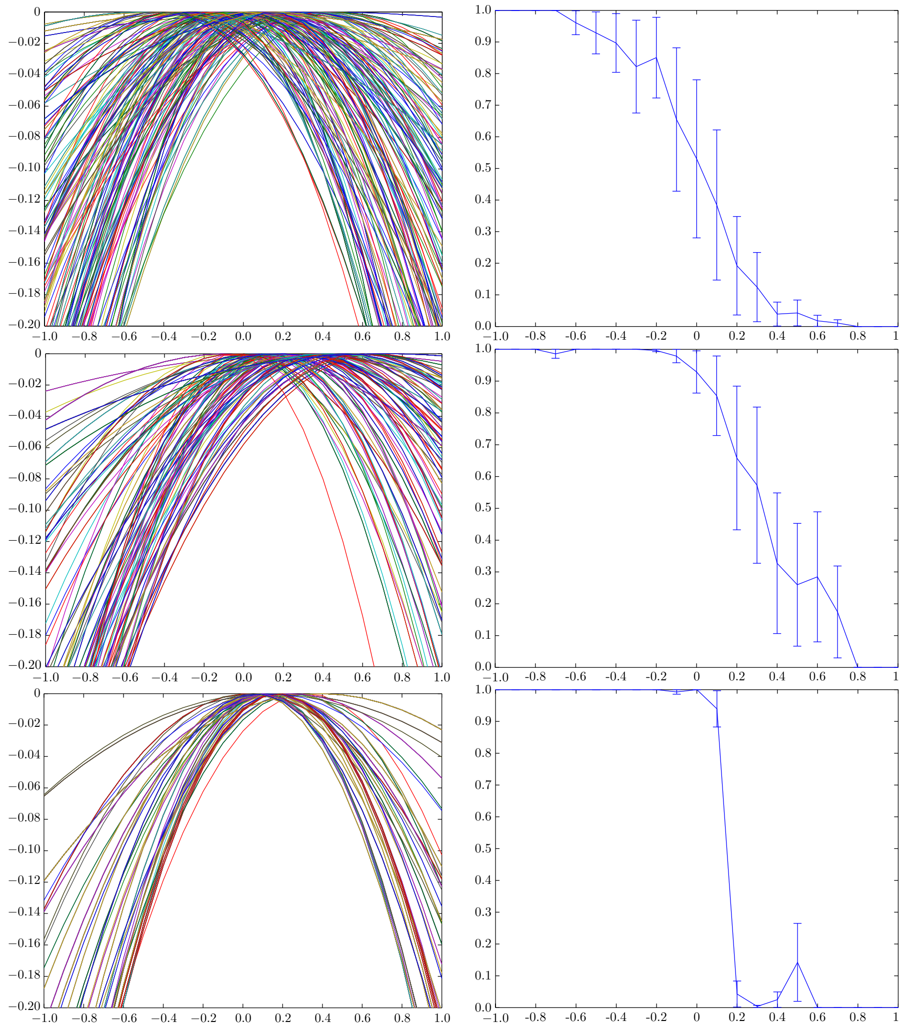

Fig. 3. Sample iterations (1st, 2nd and 5th) of Algorithm 1 in the problem of Fig. 2. On the left are samples obtained from $\mathbb{P}[r \mid \mathcal{D}]$ and on the right the corresponding π_D, showing the mean the and variance of the optimal action for each state (1- move right, 0- move left).

must reach a goal region while avoiding a penalty region (the "puddle"), as depicted in Fig. 4. This example illustrates our active IRL algorithm at work in a more complex problem that can still visualized.

The MDP has a continuous state-space, consisting of the unit square, and a discrete action-space that includes the four actions N (north), S (south), E (east), and W (west). Each action moves the agent 0.05 in the corresponding direction. Since the MDP has a continuous state-space, exact solution methods

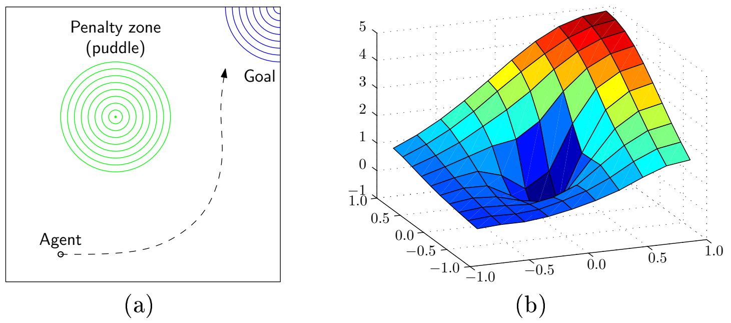

Fig. 4. Representation of the puddle world and a possible value function.

are not available. We adopt a batch approximate RL method known as *fitted Q-iteration* that essentially samples the underlying MDP and uses regression to approximate the optimal Q-function [12]. The fact that we must resort to function approximation implies that the exact optimal policy cannot be recovered but only an approximation thereof. This will somewhat impact the ability of our algorithm to properly estimate the posterior $\mathbb{P}[r \mid \mathcal{D}]$.

In the puddle world, the reward function can be represented as

$$r(x) = r_{\text{goal}} \exp\big((\mathbf{x} - \boldsymbol{\mu}_{\text{goal}})^2/\alpha\big) + r_{\text{puddle}} \exp\big((\mathbf{x} - \boldsymbol{\mu}_{\text{puddle}})^2/\alpha\big),$$

where r_{goal} and r_{puddle} represent the reward and maximum penalty received in the goal position and in the center of the puddle, respectively. The parameters $\boldsymbol{\mu}_{\text{goal}}$ and $\boldsymbol{\mu}_{\text{puddle}}$ define the location of the goal and puddle, respectively. The parameter α is fixed *a priori* and roughly defines the width of both regions. For our IRL problem, the agent should learn the parameters $\boldsymbol{\mu}_{\text{goal}}, \boldsymbol{\mu}_{\text{puddle}}, r_{\text{goal}}$, and r_{puddle} from a demonstration.

Figure 5 presents two sample iterations of Algorithm 1. To solve the MDP we ran fitted Q-iteration with a batch of $3,200$ sample transitions. We ran MCMC with $N = 800$. Notice that after the first iteration (using the initial demonstration), the MCMC samples are already spread around the true parameters. At each iteration, the algorithm is allowed to query the expert in 10 states. In the depicted run, the algorithm queried states around the goal region — to pinpoint the goal region — and around the puddle — to pinpoint the puddle region.

4.3 Random Scenarios

We now illustrate the application of our approach in random scenarios with different complexity. We also discuss the scalability of our algorithm and statistical significance of the results.

These general MDPs in this section consist of squared grid-worlds with varying number of states. At each state, the agent has 4 actions available (N, S, E, W), that moves the agent in the corresponding direction. We divide our results in two classes, corresponding to *parameterized rewards* and general rewards.

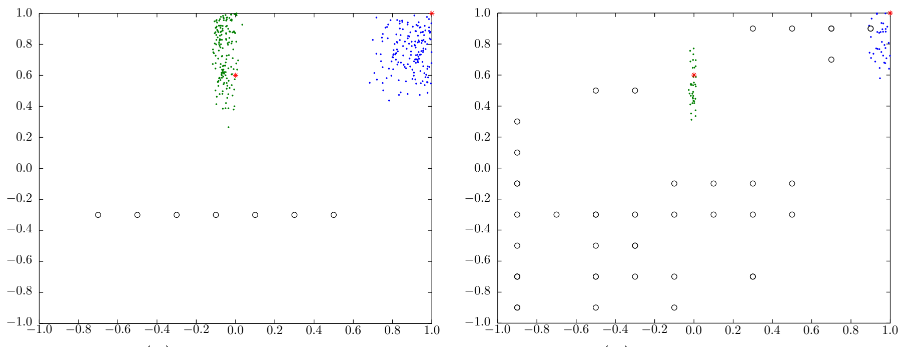
(a) Iteration 1. (b) Iteration 3.

Fig. 5. Two sample iterations of Algorithm 1. The red stars ($*$) represent the target values for the parameters $\boldsymbol{\mu}_{\text{goal}}$ and $\boldsymbol{\mu}_{\text{puddle}}$. The green and blue dots ($\cdot$) represents the sampled posterior distribution over possible value of these parameters. The circles ($\circ$) denotes the states included in the demonstration.

Parameterized Rewards We start by considering a simple parameterization of the reward function of the form $\delta_{x^*}(x)$. Therefore, the only parameter to be learnt is the position of the goal state x^* in the grid.

We applied Algorithm 2 to a 15×15 grid-world. The estimation in step 2 of the algorithm uses $N = 15$. At each iteration, the agent is allowed to query the expert in 10 states. Figure 6(b) shows the error between the estimated policy and the target policy as a function of the size of the demonstration, averaged over 50 independent trials. Our approach clearly outperforms random sampling, attaining the same error while requiring about 1/3 of the samples.

We conclude by noting that we chose to run Algorithm 2 in this scenarion since (as discussed in Section 2.3, the MCMC component in Algorithm 1 does not scale well with the number of states. Indeed, for a similar scenario with 100 states, the MCMC-based algorithm required around 12,000 MC samples, for each of which an MDP must be solved. In that same 100-state scenarion, Algorithm 2 required around 50 gradient steps and then 20 MC samples to compute the local approximation of the posterior, thus requiring a total of 70 MDPs to be solved.

Non-parameterized reward We now consider a more general situation, in which the reward function is a vector $\mathbf{r}$ in the $|\mathcal{X}|$-dimensional unit square. In this case, the reward value is merely a real-valued function $r : \mathcal{X} \rightarrow [0; 1]$, and the problem is significantly more complex than in the previous case.

We applied Algorithm 2 to a 10×10 grid-world. The estimation in step 2 of the algorithm uses $N = 40$. At each iteration, the agent is allowed to query the expert in 2 states. Figure 6(a) shows the error between the estimated policy and the target policy as a function of the size of the demonstration, averaged over 50 independent trials. In this case, it is clear that there is no apparent advantage

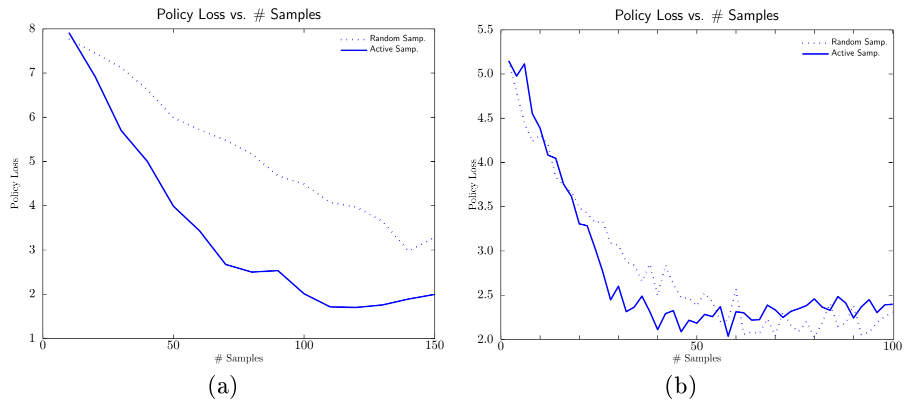

Fig. 6. Performance of Algorithm 2 comparing active sampling vs. random sampling as a function of the demonstration size. (a) Results with parameterized rewards in a 15×15 grid-world. (b) Results with general (non-parameterized) rewards in a 10×10 grid-world.

in using the active learning approach. Whatever small advantage there may be is clearly outweighed by the added computational cost.

These results illustrate, in a sense, some of the issues already discussed in Section 3.1. When considering a non-parameterized form for the reward function and a prior over possible rewards that is state-wise independent, there is not enough structure in the problem to generalize the observed policy from observed states to non-observed states. In fact, the space of general (non-parameterized) reward functions has enough degrees of freedom to yield any possible policy. In this case, any sampling criterion will, at best, provide only a mild advantage over uniform sampling. On the other hand, when using parameterized rewards or a prior that weights positively ties between the reward in different states (*e.g.*, an Ising prior [6]), the policy in some states restricts the possible policies on other states. In this case, sampling certain states can certainly contribute to disambiguate the policy in other states, bringing significant advantages to an active sampling approach over a uniform sampling approach.

5 Conclusions

In this paper we introduced the first active learning algorithm explicitly designed to estimate rewards from a noisy and sampled demonstration of an unknown optimal policy. We used a full Bayesian approach and estimate the posterior probability of each action in each state, given the demonstration. By measuring the state-wise entropy in this distribution, the algorithm is able to select the potentially most informative state to be queried to the expert. This is particularly important when the cost of providing a demonstration is high.

As discussed in Section 4, our results indicate that the effectiveness of active learning in the described IRL setting may greatly depend on the prior knowledge about the reward function or the policy. In particular, when considering parameterized policies or priors that introduce relations (in terms of rewards) between different states, our approach seems to lead to encouraging results. In the general (non-parameterized) case, or when the prior "decorrelates" the reward in different states, we do not expect active learning to bring a significant advantage. We are currently conducting further experiments to gain a clearer understanding on this particular issue.

We conclude by noting that active learning has been widely applied to numerous settings distinct from IRL. In some of these settings there are even theoretical results that state the improvements or lack thereof arising from considering active sampling instead of random sampling [10]. To the extent of our knowledge, ours is the first paper in which active learning is applied within the context of IRL. As such, many new avenues of research naturally appear. In particular, even if the ambiguities inherent to IRL problems make it somewhat distinct from other settings, we believe that it should be possible (at least in some problems) to theoretically asses the usefulness of active learning in IRL.

References

1. Lopes, M., Melo, F., Kenward, B., Santos-Victor, J.: A computational model of social-learning mechanisms. Adaptive Behavior (2009, to appear)
2. Melo, F., Lopes, M., Santos-Victor, J., Ribeiro, M.: A unified framework for imitation-like behaviors. In: Proc. 4th Int. Symp. Imitation in Animals and Artifacts. (2007)
3. Ng, A., Russell, S.: Algorithms for inverse reinforcement learning. In: Proc. 17th Int. Conf. Machine Learning. (2000) 663–670
4. Neu, G., Szepesvári, C.: Apprenticeship learning using inverse reinforcement learning and gradient methods. In: Proc. 23rd Conf. Uncertainty in Artificial Intelligence. (2007) 295–302
5. Abbeel, P., Ng, A.: Apprenticeship learning via inverse reinforcement learning. In: Proc. 21st Int. Conf. Machine Learning. (2004) 1–8
6. Ramachandran, D., Amir, E.: Bayesian inverse reinforcement learning. In: Proc. 20th Int. Joint Conf. Artificial Intelligence. (2007) 2586–2591
7. Syed, U., Schapire, R., Bowling, M.: Apprenticeship learning using linear programming. In: Proc. 25th Int. Conf. Machine Learning. (2008) 1032–1039
8. Ziebart, B., Maas, A., Bagnell, J., Dey, A.: Maximum entropy inverse reinforcement learning. In: Proc. 23rd AAAI Conf. Artificial Intelligence. (2008) 1433–1438
9. Ng, A.Y., Harada, D., Russell, S.: Policy invariance under reward transformations: Theory and application to reward shaping. In: Proc. 16th Int. Conf. Machine Learning. (1999) 278–287
10. Settles, B.: Active learning literature survey. CS Tech. Rep. 1648, Univ. Wisconsin-Madison (2009)
11. Andrieu, C., de Freitas, N., Doucet, A., Jordan, M.: An introduction to MCMC for machine learning. Machine Learning **50** (2003) 5–43
12. Timmer, S., Riedmiller, M.: Fitted Q-iteration with CMACs. In: Int. Symp. Approximate Dynamic Programming and Reinforcement Learning. (2007)