Active Reward Learning from Critiques

Yuchen Cui
Department of Computer Science
University of Texas at Austin, Austin, TX 78712
yuchencui@utexas.edu

Scott Niekum
Department of Computer Science
University of Texas at Austin, Austin, TX 78712
sniekum@cs.utexas.edu

***Abstract*—Learning from demonstration algorithms, such as *Inverse Reinforcement Learning*, aim to provide a natural mechanism for programming robots, but can often require a prohibitive number of demonstrations to capture important subtleties of a task. Rather than requesting additional demonstrations blindly, active learning methods leverage uncertainty to query the user for action labels at states with high expected information gain. However, this approach can still require a large number of labels to adequately reduce uncertainty and may also be unintuitive, as users are not accustomed to determining optimal actions in a single out-of-context state. To address these shortcomings, we propose a novel trajectory-based active Bayesian inverse reinforcement learning algorithm that 1) queries the user for *critiques* of automatically generated trajectories, rather than asking for demonstrations or action labels, 2) utilizes *trajectory segmentation* to expedite the critique / labeling process, and 3) predicts the user's critiques to generate the most highly informative trajectory queries. We evaluated our algorithm in simulated domains, finding it to compare favorably to prior work and a randomized baseline.**

I. Introduction

Robots have successfully been used to automate tasks such as manufacturing, in which the environment is strictly controlled. However, as automation begins to expand to homes and less structured workplaces, it is not feasible for robotics engineers to program general-purpose robots to perform highly variable tasks in many different environments. In response to this challenge, recent learning from demonstrations (LfD) [4] algorithms aim to provide an intuitive interface that allows end-users to program robots without the use of code or expert knowledge. *Inverse Reinforcement Learning* (IRL) is a form of LfD that focuses on recovering the underlying reward function that generates an expert's demonstrations [1]. Using reinforcement learning in conjunction with this learned reward function typically provides superior generalization compared to supervised policy learning, as it captures the underlying *intention* of the demonstrator, which transfers well to unseen states and environments.

A significant problem for existing IRL algorithms is that they often require a large number of demonstrations to be robust, while at the same time, it is difficult for users to determine what types of demonstrations are most informative to show the robot. *Active learning* is a framework in which the learning agent selects its own input data, leveraging uncertainty and expected information gain. A closely related active IRL technique [15] allows a robot to query the user

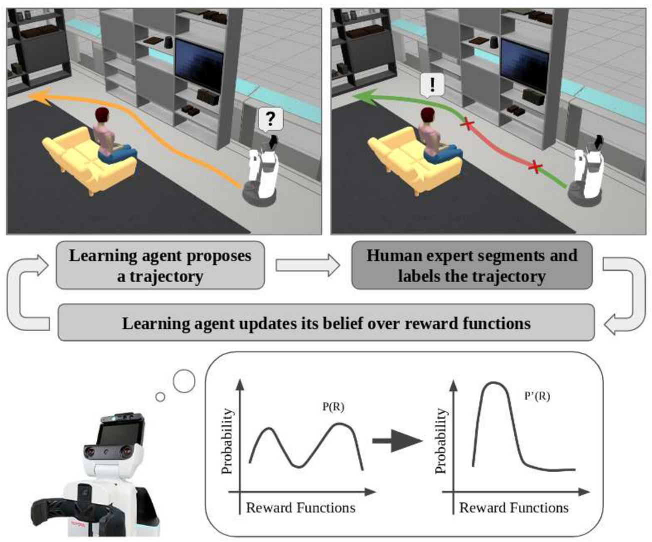

Learning agent proposes a trajectory
Human expert segments and labels the trajectory
Learning agent updates its belief over reward functions
P(R)
P'(R)
Probability
Reward Functions

Fig. 1: Proposed active learning process with an example showing a Toyota Human Support Robot (HSR) interacting with a human user by proposing a trajectory and updating its belief over reward functions, leveraging the labeled segments.

for demonstrations at states with highest uncertainty over actions, which can still require substantial human effort and is unintuitive for certain types of problems where it is hard to determine the optimal action at a single out-of-context state. It is also desirable to have a method that can work when the user does not understand the action space of the robot, or is not physically present to provide additional demonstrations—for example, a robot encountering an error while deployed may contact a remote call center for assistance. Finally, the algorithm should minimize the amount of human feedback required, in order to reduce user burden. To address these shortcomings, we propose a novel trajectory-based algorithm, Active Reward learning from Critiques (ARC).

As shown in Figure1, the proposed method automatically generates a trajectory per interaction with an user, which the user is then asked to *critique* by marking segmentation points along the trajectory and labeling the resulted segments. Instead of asking the user to critique the trajectory as a whole, our method allows the user to segment the trajectory into *good* and *bad* sections, since trajectories are rarely purely optimal or pessimal. This approach enables users to understand actions in the context of a trajectory, while also allowing for the collection of many state-action labels from a small number of segmentation points. By proposing trajectories and being able to leverage both positive and negative critiques, ARC also allows undesired behaviors to be addressed explicitly during learning. To generate trajectories that are likely to result in high information gain, ARC builds on Bayesian Inverse Reinforcement Learning (BIRL) [18], which samples from a belief distribution over possible reward functions under a given set of demonstrations. These reward samples can be used to predict an expected segmentation of any given trajectory, and in turn, predict the expected change in the reward function distribution, and thus the information gain that will result from the query. While exactly finding an information-maximizing trajectory is infeasible in practice, we present an approximate algorithm for doing so, and show that it compares favorably to prior work and a random baseline in terms of policy loss, data efficiency, and required labeling effort.

II. Related Work

Inverse Reinforcement Learning (IRL), a subtype of Learning from Demonstration [4], is the process of inferring a reward function from observed behavior [1]. In contrast to approaches that aim to directly mimic the expert's behavior, such as max margin planning [19], IRL algorithms learn a reward function that describes the task and therefore is often transferable to new environments. However, recovering the exact reward function is an ill-posed problem since many reward functions generate the same optimal policy, including a reward function that is zero at every state. Abbeel and Ng [1] use a max-margin algorithm to match the feature counts between the expert's policy and the learning agent's policy. Given samples from the expert's policy, there are many policies matching the feature counts. To address the ambiguity in choosing policies, Ziebart et al. [22] employed the principle of maximum entropy to find a reward function that maximizes the entropy of the probability distribution over paths. However, these feature-count based techniques do not directly address the potential sub-optimality of demonstrations and cannot work with partial trajectories.

Bayesian Inverse Reinforcement Learning (BIRL) was first proposed by Ramachandran & Amir [18] as a principled way of approaching IRL, casting the ill-posed problem into Bayesian framework. In BIRL, demonstrated state-action pairs are each used as independent evidence to update a posterior distribution over the space of reward functions. Choi & Kim [7] suggested to use a Maximum a Posteriori (MAP) estimation instead of taking the mean of the reward distribution as a more accurate estimate of the reward function. Our proposed method transforms BIRL into a *learning from critiques* (LfC) method and also leverages both positive and negative samples to address data-efficiency.

Lopes et al. [15] built an active sampling (AS) algorithm based on BIRL that enables the robot to query the expert for demonstrations at states where the entropy over the distribution of action probabilities is high. Cohn et al. [9] proposed to use myopic expected value of information as the measure for selecting action queries instead of policy entropy. Our work is closely related to that of Lopes et al. [15] and Cohn et al. [9]. However, comparing to the two, our method is more data-efficient with the same amount of human effort since it generates a sequence of actions as its query instead of asking for individual demonstrations at out-of-context states. Additionally, ARC reasons about information gain directly over the space of possible reward functions instead of policy representations or expected values. While many reward functions will generate the same optimal policy, the size of the set of all possible reward functions will not increase as more evidence (demonstrations) are provided. Therefore, we expect the entropy of the probability distribution over all possible reward functions to decrease as more informative demonstrations are provided.

There also exists work on reducing teaching burden by leveraging human feedback outside the context of IRL. Cakmak and Thomaz [5] studied how to design effective robot learners from a human-robot interaction perspective. Their results in part support our design choice of using label queries instead of asking for demonstrations in an effort to reduce teaching burden. Argall et al. [3] presented an approach to incorporate human critiques at the policy level into an 1-Nearest-Neighbor-based LfD algorithm. In the context of Reinforcement Learning (RL), Judah et al. [11] enabled an agent to learn a parameterized policy from expert's critiques by encoding critiques into reward values. The TAMER framework by Knox et al. [12] provides a way of interactive policy shaping by explicitly addressing the credit assignment problem. Preference-based learning has also been widely used to reduce teaching burden. Christiano et al. [8] trained a deep network to predict rewards using feedback of human's preference over pairs of trajectory segments. Sadigh et al. [10] proposed a way to learn reward functions that encode a human's preferences for the behavior of a dynamical system by generating pairs of candidate trajectories using different feature weights. However, both trajectories may contain different sub-optimal segments, making it difficult to compare them as whole trajectories.

III. Background

A. Markov Decision Processes

In general, a Markov Decision Process (MDP) is a tuple $(S, A, T, R, d_0, \gamma)$, where: S is a set of states; A is a set of actions; $T : S \times A \times S \to [0, 1]$ is a transition probability function; $R : S \to \mathbb{R}$ is a reward function, with absolute value bounded by R_{max}; d_0 is a starting state distribution and $\gamma \in [0, 1)$ is the discount factor.

A deterministic policy is a mapping from state to action $\pi : S \to A$. The value of a state given a policy is calculated by:

$$V^{\pi}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|s_0 = s, \pi] \qquad (1)$$

The Q-function is defined to describe values of state-action pairs according to some policy:

$$Q^\pi(s,a) = R(s) + \gamma \mathbb{E}_{s' \sim T(s,a,*)}[V^\pi(s')] \tag{2}$$

Bellman equations are used to describe a recursive relationship between values of neighboring states and state-action pairs:

$$V^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^\pi(s') \tag{3}$$

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s'} T(s, a, s') V^\pi(s') \tag{4}$$

A policy π is optimal if and only if:

$$\forall s \in S, \pi(s) \in \arg\max_{a \in A} Q^\pi(s,a) \tag{5}$$

B. Bayesian Inverse Reinforcement Learning

In the formulation of BIRL by Ramachandran & Amir [18], we consider a Markov Decision Process without reward function, denoted as MDP/R, (S, A, T, d_0, γ) and an expert χ operating in the MDP. The expert χ is assumed to be attempting to maximize the total accumulated reward according to a reward function R, using some stationary policy. The IRL agent receives a set of demonstrations $D = \{(s_0, a_0), (s_1, a_1) ... (s_k, a_k)\}$. Since the policy used by χ is assumed to be stationary, we can make the independence assumption that:

$$Pr(D|R) = \prod_{i=0}^{k} Pr((s_i, a_i|R)) \tag{6}$$

According to equation (5), the reward-maximizing actions are equivalent to the actions with highest Q-values. Therefore, the likelihood of an action (s_i, a_i) given a reward function R can be modeled as:

$$Pr((s_i, a_i)|R) = \frac{1}{Z_i} e^{\alpha Q(s_i, a_i, R)} \tag{7}$$

where α is a parameter representing the degree of confidence we have in χ's ability to choose the optimal actions [18]. Therefore, the likelihood of the entire evidence can be expressed as:

$$Pr(D|R) = \frac{1}{Z} e^{\alpha \sum_i Q(s_i, a_i, R)} \tag{8}$$

With Bayes theorem, the posterior probability of reward function R is:

$$Pr(R|D) = \frac{Pr(D|R)Pr(R)}{Pr(D)} = \frac{1}{Z'} e^{\alpha \sum_i Q(s_i, a_i, R)} Pr(R) \tag{9}$$

While the normalizing constant Z' is hard to compute, the Markov Chain Monte Carlo (MCMC) sampling algorithm only needs the ratios of probability densities. Therefore, BIRL outputs an unnormalized probability distribution of reward functions, from which we can extract a MAP estimate of the reward function R or the mean policy $\bar{\pi}$.

IV. Methodology

First, ARC proposes a trajectory to a human expert, who will segment the trajectory into *good* and *bad* contiguous segments. The expected information content of the trajectory will be judged by examining the expected change in the agent's belief over reward functions. The belief distribution is approximated using the MCMC sampling algorithm in BIRL. BIRL considers each state-action pair separately so it is able to learn from partial trajectories and segments, unlike feature-count based methods [1][22] that require full trajectories.

A. Learning from Negative Examples using BIRL

Given that the trajectories will be segmented into *good* and *bad* parts, we modified BIRL so that it can leverage both positive and negative samples of the expert's policy. In an MDP, for a particular state s_i, an action (s_i, a_i) is either optimal or not, though multiple actions can be optimal. We assume the expert has been instructed to label the optimal actions as *good* and sub-optimal actions as *bad* but these labels may be corrupted with noise. As implied by the Bellman equations, the set of optimal actions $O(s)$ at each state s is:

$$O(s) = \arg\max_{a \in A} Q^\pi(s,a) \tag{10}$$

Following the original BIRL formulation [18], given a reward function R, the probability that the action belongs to $O(s)$ is exponentially higher if it has a larger $Q(s,a)$ value. We assume that the expert's policy is stationary and optimal under some reward function, so that demonstrations labeled as *good* all belong to $O(s)$, and those labeled as *bad* do not. Therefore, probabilities that a state-action pair is *good* or *bad* under some reward function R can be formulated as:

$$Pr(a_i \in O(s_i) \mid R) = \frac{1}{Z_i} e^{\alpha Q(s_i, a_i, R)} \tag{11}$$

$$Pr(a_i \notin O(s_i) \mid R) = 1 - \frac{1}{Z_i} e^{\alpha Q(s_i, a_i, R)} \tag{12}$$

The value of parameter α quantifies the degree of confidence or importance of a particular state-action pair (s_i, a_i) being optimal or not. This value of α can be approximated—for example, with expectation maximization [21]—but this is not the focus of this work. We denote the set of *good* trajectory segments as D^+ and the set of *bad* trajectory segments as D^-. The likelihood of the entire evidence is then expressed as:

$$Pr(D^+, D^- \mid R) = \prod_{(s_i, a_i) \in D^+} Pr(a_i \in O(s_i) \mid R) \prod_{(s_j, a_j) \in D^-} Pr(a_j \notin O(s_j) \mid R) \tag{13}$$

The algorithm we use to generate samples of reward functions from the posterior using D^+ and D^- is GenerateSamples as shown in **Algorithm 1**, which is modified from the PolicyWalk algorithm of Ramachandran & Amir [18]. It takes the likelihood function, an MDP/R, sets of positive and negative evidence, a desired chain length and a step size for modifying

Algorithm 1 GenerateSamples(P, mdp, D^+,D^-,l,ϵ)

```
1: Randomly initialize reward vector $R \in \mathbb{R}^{||S||}$
2: $R\_chain[0] = R$
3: $\pi$ :=PolicyIteration$(mdp, R)$
4: $i := 1$
5: while $i < l$ do
6:     Randomly perturb $R$ by step size $\epsilon$ and get $R'$
7:     Compute $Q^\pi(s, a, R')$ for all $(s, a) \in D^+ \cup D^-$
8:     $\pi'$ := PolicyIteration$(mdp, R', \pi)$
9:     if $rand(0, 1) < min\{1, \frac{P(R',\pi',D^+,D^-)}{P(R,\pi,D^+,D^-)}\}$ then
10:        $R\_chain[i] = R'$
11:        $R = R'$
12:        $i := i + 1$
13:    end if
14: end while
15: return $R\_Chain$
```

the reward functions as input, and outputs an array of sampled reward functions. To reduce the autocorrelation in between the samples obtained, only every 20th sample is used in practice.

B. Calculating Expected Information Gain

In order to compare sate-action pairs in terms of their information gain, we need a measure for estimating how much information the agent can obtain by updating some state-action pair to be good or bad. It is desirable to have a measure that captures the differences in belief distributions before and after updating the optimality of a candidate state-action pair.

The Kullback-Leibler (KL) divergence [14] between two distributions is widely used as a measure for information gain in information theory [2]. The equation for computing the KL divergence for two discrete distributions is:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{14}$$

KL divergence is asymmetric, since $D_{KL}(P||Q)$ and $D_{KL}(Q||P)$ are not the same. While it is desirable to use a symmetric measure as a distance metric, it is known that the asymmetry of KL divergence helps to avoid local optima during active learning processes [13]. Many other measures of distance between two probability distributions [6] may also be used here.

Since GenerateSamples only outputs a set of samples in the MCMC chain, the KL divergence between the underlying densities p and q on $\mathbb{R}^d$ is then estimated from the two sets of samples $\{X_1, ..., X_n\}$ and $\{Y_1, ..., Y_m\}$ by a method based on k-Nearest-Neighbor distances [20] using the following equation:

$$\hat{D}_{n,m}(p||q) = \frac{d}{n} \sum_i \log \frac{\nu_k(i)}{\rho_k(i)} + \log \frac{m}{n-1} \tag{15}$$

where $\rho_k(i)$ and $\nu_k(i)$ denotes the distance from X_i to its k-NN in $\{X_j\}_{j \neq i}$ and $\{Y_j\}$ respectively.

Divergence between the updated and the original distribution cannot be directly used as the expected information gain. For instance, updating some state-action pair, which is believed with 0.99 probability to be *good*, as *bad* will certainly shift the distribution by a lot, while there is little chance to actually update it to *bad*. Therefore, the algorithm also weights these distances by the probabilities of the state-action pair being optimal or not based on current belief.

Given the IRL agent's current belief of reward functions $Be(R)$, the probability of a state-action pair to be labeled as *good* or *bad* is calculated using:

$$Pr(a_i \in O(s_i) \mid Be(R)) = \sum_{R_k} Pr(R_k \mid Be(R)) Pr(a_i \in O(s_i) \mid R_k) \tag{16}$$

$$Pr(a_i \notin O(s_i) \mid Be(R)) = \sum_{R_k} Pr(R_k \mid Be(R)) Pr(a_i \notin O(s_i) \mid R_k) \tag{17}$$

The expected information gain by updating one state-action pair to be *good* or *bad* is then expressed as:

$$\begin{aligned} G^+(s_i, a_i) &= G(D^+ \cup (s_i, a_i) \mid Be(R)) \\ &= Pr(a_i \in O(s_i) \mid Be(R)) D(Be'(R)||Be(R)) \end{aligned} \tag{18}$$

$$\begin{aligned} G^-(s_i, a_i) &= G(D^- \cup (s_i, a_i) \mid Be(R)) \\ &= Pr(a_i \notin O(s_i) \mid Be(R)) D(Be'(R)||Be(R)) \end{aligned} \tag{19}$$

Algorithm 2 GetInfoGain((s, a), P, mdp, D^+, D^-, l, ϵ)

```
1: $D^+_{tmp} := D^+ \cup (s, a)$
2: $D^-_{tmp} := D^- \cup (s, a)$
3: $Rwd$ := GenerateSamples$(P, mdp, D^+, D^-, l, \epsilon)$
4: $Rwd^+$ := GenerateSamples$(P, mdp, D^+_{tmp}, D^-, l, \epsilon)$
5: $Rwd^-$ := GenerateSamples$(P, mdp, D^+, D^-_{tmp}, l, \epsilon)$
6: $Rwd\_total := Rwd \cup Rwd^+ \cup Rwd^-$
7: Initialize belief arrays $Be, Be_+, Be_-$
8: Initialize probabilities $P^+ := 0$, $P^- := 0$
9: for each $r \in Rwd\_total$ do
10:    $\pi$ :=PolicyIteration$(mdp, r)$
11:    Compute $Q^\pi(s, a, r)$ for all $(s, a) \in D^+ \cup D^-$
12:    $Be(r) := P(r, \pi, D^+, D^-)$
13:    $Be^+(r) := P(r, \pi, D^+_{tmp}, D^-)$
14:    $Be^-(r) := P(r, \pi, D^+, D^-_{tmp})$
15:    Initialize normalizing factor $Z := 0$
16:    for $a_i \in A$ do
17:        $Z := Z + e^{\alpha Q(s, a_i)}$
18:    end for
19:    $P^+ := P^+ + Be^+(r) \frac{e^{\alpha Q^\pi(s,a,r)}}{Z}$
20:    $P^- := P^- + Be^-(r)(1 - \frac{e^{\alpha Q^\pi(s,a,r)}}{Z})$
21: end for
22: Normalize $Be, Be_+, Be_-$
23: $Gain\_total = D(Be_+||Be)P_+ + D(Be_-||Be)P_-$
24: return $Gain\_total$
```

GetInfoGain, as presented in **Algorithm 2** [1], returns the estimated information gain of a specific state-action pair, which is calculated as the sum of the weighted KL divergences.

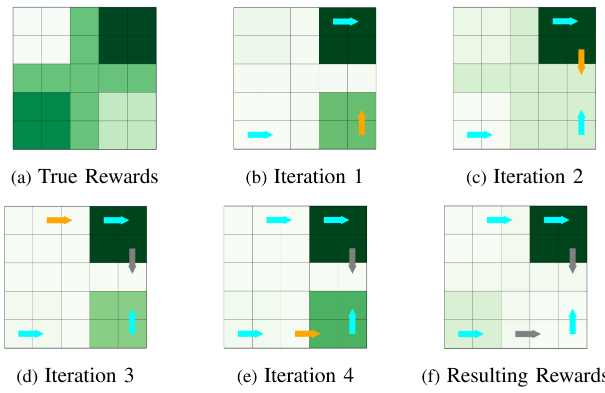


Fig. 2: An illustrative example in a 5×5 gridworld demonstrating actions with maximum expected information gain explore unseen features. Each grid cell has only one of the 5 features. (green: average rewards - darker is larger; cyan: known good actions; gray: known bad actions; orange: actions with max expected info gain)

Iteration	Expected Information Gain	Entropy	Policy Loss
0	-	-	60%
1	4.2753338603	231.58	32%
2	4.2614594772	159.88	28%
3	4.9553412646	151.70	24%
4	5.2887902710	150.42	0%

TABLE I: Info Gain for Action Queries in Fig. 2

Figure 2(a) shows an example of 5×5 gridworld with a simple layout of rewards. The process started with two initial demonstrations and Figures 2(b)(c)(d) and (e) present the recovered mean rewards and the corresponding action with the maximized expected information gain in 4 consecutive iterations running the algorithm. The selected actions tend to explore a variety of states with different rewards, which agrees with our intuition for selecting informative actions. Table I shows the entropy of the distribution is decreasing over iterations and the policy loss inferred by the mean rewards also decreases over iterations.

C. Generating Informative Trajectories

By leveraging the above technique for computing the information gain of single state-action pairs, trajectories of length N can be constructed in various ways. To find the trajectory with the maximum expected information gain among all trajectories of length N starting at some state-action (s_0, a_0), we need to evaluate $O(A^N)$ trajectories, where A is the number of possible actions at every time step, which will quickly become intractable as the dimensionality of problem space increases.

A greedy approach will be more efficient than brute force search. However, trajectories generated by selecting actions greedily without any constraint can be arbitrarily shaped (such as oscillating between states) and unnatural for humans to effectively evaluate. Therefore, instead of generating a trajectory action by action, we sample candidate trajectories from the optimal policies of sampled reward functions. By doing so, actions in a trajectory are all generated from a single, consistent reward function, so that they will be more interpretable to a human in terms of features. The expected information gain of each of these trajectories is estimated by iteratively labeling each action with their expected label and the trajectory with maximum expected information gain is selected as the query to the expert for critiquing. In order to estimate the information gain of a state-action pair that is not the first one in the trajectory, all prior state-action pairs are added to the demonstration sets with their expected label given current belief. Given a trajectory p of length k, its total information gain is estimated using:

$$G(p) = \sum_{(s_i,a_i)\in p} \beta^i G(s_i, a_i) \tag{20}$$

where $\beta \in [0, 1)$ is the discount factor for a bias toward higher information gain at the beginning of the trajectory since the later state-action pairs' information gains are estimated with accumulated assumptions. β is set to be 0.9 in all our experiments.

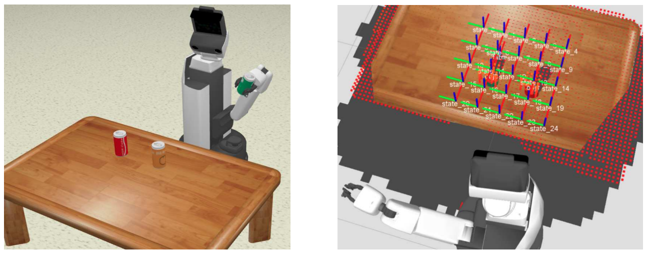
V. Experiment Setup

We perform experiments in two different tasks: Navigate-In-Gridworld task and Place-An-Object task. In the first task, grid cells are initialized with different numbers of features with randomized weights so that we can easily test the algorithm with problems of varying complexity. In the second task, the problem domain is structured in a way that all the state features are interpretable so that we can examine the trajectories generated by the algorithm qualitatively.

In the Navigate-In-Gridworld task, 8×8 gridworlds with different number of features are used. For ground-truth rewards, each grid cell is randomly assigned a feature vector with binary values that indicate which features are present in this cell. The reward is calculated as a linear combination of features, as assumed by prior work [1], of which the weights are randomly generated as well. The sizes of the feature vector used are 8, 16, 32 and 48. The MDP/R problem we formulated is (S, A, T, d_0, γ) with states $||S|| = 64$ (each cell is a unique state), actions $A = \{Up, Down, Left, Right\}$, T is a deterministic transition matrix, and γ is set to be 0.95 favoring potential future rewards.

In the Place-An-Object task, we assume that the robot is learning how to efficiently place an object relative to two objects currently on the table. As shown in Figure 3, possible placement locations around the two objects are detected as different states that the robot's end-effector can move to. 32 binary features are predefined to describe the position of a state and its distance relative to the two existing objects (e.g. 1-step left to object A). The robot can move its end-effector to any start state, from which it will be able to move to

[1]In our implementation, samples for the base distribution are only obtained once before calling the GetInfoGain function for a specific state-action pair.



(a) Simulated Scene (b) Detected States

Fig. 3: Setup for Place-An-Object task in simulation

any adjacent states (in eight discrete directions) or stay at the current state at each step. We defined four different ground-truth rewards (shown in Figure 5(a)(b)(c)(d)) to see how the queries generated by the system would differ. We designed one of the ground truth reward functions 5(d) so that there are equally optimal actions in several states.

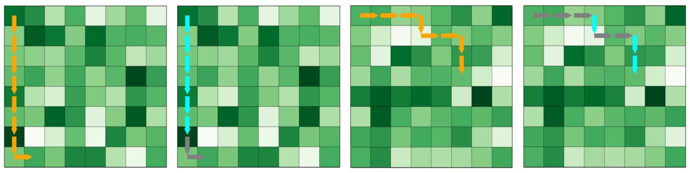
For both tasks, we used the optimal policy under the ground-truth rewards as the expert segmenting and labeling the generated trajectories. 100 different experiments were conducted for each feature size in the Navigate-In-Gridworld task, and 100 runs of the same experiment setup were performed for each of the four ground truth reward for Place-An-Object task. The convergence rate of BIRL is sensitive to the confidence factor α in equations (7),(8) and (9). Since we are using the optimal oracle, we set α to be relatively high at 200 so that the BIRL processes with full information for a 8×8 gridworld converge quickly before 12000 samples. In more realistic cases with noise, the value of α can be adjusted for each state-action pair by measuring its consistency with other demonstrations.

To test the performance of the algorithm, we compare ARC with a baseline uniform sampling algorithm (*Random*) and an active sampling algorithm (*AS*) by Lopes et al. [15], which asks for demonstrations at states with high entropy over actions. ARC and Random each proposes a trajectory of length at most N per iteration. Since the active sampling (AS) algorithm is not designed to generate trajectories, we instead for each iteration select N states with maximum entropies and ask the expert for demonstrations at those states.

In most domains, there are usually more sub-optimal actions than optimal ones in any given state, so that demonstrations, or positive labels, provide more valuable information than negative labels. Therefore, we expect the AS algorithm, which asks for a demonstration at selected states, to perform better than ARC. Moreover, the way our method estimates information gain is based on the system's current belief, therefore, the estimations will only become more accurate as more evidence is provided. Hence, our hypothesis is that, in terms of policy loss per iteration, ARC will reach a lower policy loss faster than uniform sampling and will catch up with the performance of the active sampling algorithm given enough evidence. However, if we consider labeling a segmentation point and providing a demonstration cost the same amount of labeling effort, ARC will outperform AS in terms of policy loss per labeling effort.

VI. Results

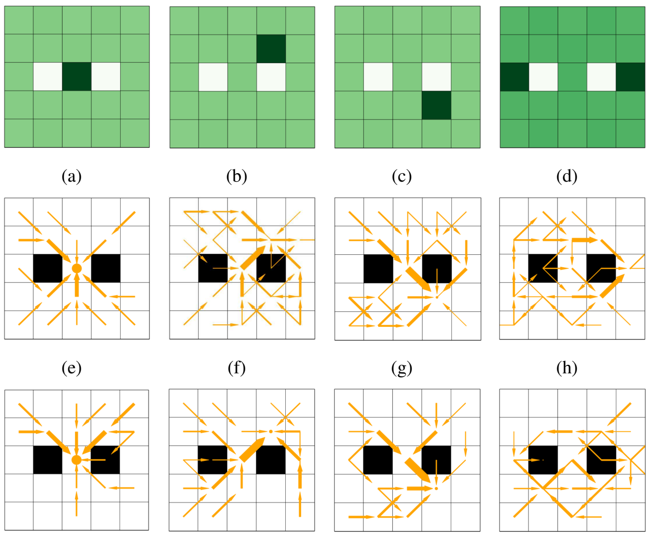
A. *Qualitative Analysis*



(a) Query 1 (b) Labeled 1 (c) Query 2 (d) Labeled 2

Fig. 4: Example of two trajectory queries and their feedback in 8×8 gridworlds with randomly generated features (green: true rewards; cyan: *good* actions; gray: *bad* actions; orange: trajectory query)

Figure 4 shows two examples of generated trajectory queries from ARC in the Navigate-In-Gridworld task and the corresponding true labels for the generated trajectories. For these queries, the user only needs to provide a few segmentation points, one and three respectively, and the algorithm obtains eight labels in total per iteration. Among all the experiments for the Navigate-In-Gridworld task, the average segmentation points for trajectories of length 8 is 2. In practice, task domains normally have more structured layout than randomly generated gridworlds, so it is reasonable to expect that trajectories generated by the algorithm won't consist of many small fragments. In the cases where we have to address the issue of fragmentation, it would be straightforward to introduce a penalty term when computing expected information gain, where an action with a different expected label than the previous action will produce a cost.



(a) (b) (c) (d)

(e) (f) (g) (h)

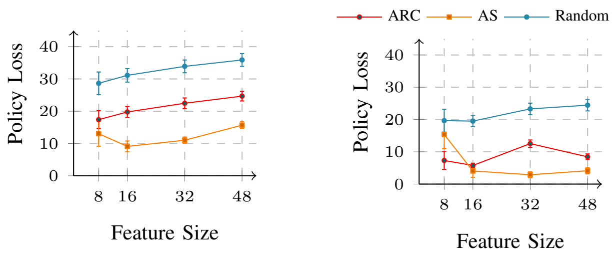
(i) (j) (k) (l)

Fig. 5: (a)(b)(c)(d) are selected ground truth rewards for Place-An-Object task; (e)(f)(g)(h) and (i)(j)(k)(l) are their corresponding 2nd and 3rd ARC query distributions (green: reward - darker is larger; orange: actions of the query - the size of the arrow is proportional to its relative frequency among all trajectories across 100 experiments)

Figure 5 presents the four selected ground truth rewards and their corresponding second and third query distributions on the discretized 2D map for the Place-An-Object task.

Without initial demonstrations, first queries are random, and therefore not shown. Once some state-action pairs are labeled accordingly, the system can then make intelligent queries. As Figure 5 shows, the second and third queries by ARC are mostly concentrated around the area with high rewards. By exploring around states with high rewards, the algorithm can then quickly learn which features should be weighted higher than others.

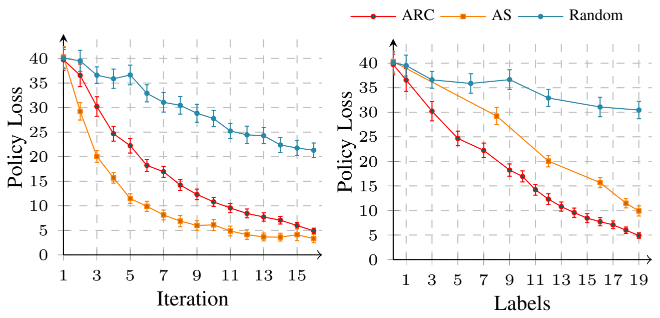
B. *Performance Evaluation*



(a) After 4 Iterations (b) After 12 Iterations

Fig. 6: Averaged policy losses (with 95% confidence bars) after 4 (left graph) and 12 (right graph) iterations over 100 different 8×8 gridworlds

Figure 6 shows the averaged policy loss after 4 and 12 iterations of the three algorithms in 8×8 gridworlds with different number of features. The larger the feature size is, the more complex the randomly generated domain becomes. The width of the gap between averaged policy losses of ARC and AS after 4 iterations is similar to that of ARC and Random, however, after 12 iterations, the averaged policy losses of ARC are very close to that of the AS algorithm and are lower in domains with smaller features. Since the information gain estimate of ARC only becomes accurate if the current belief distribution is somewhat accurate, ARC therefore performs better than AS in simpler domains.

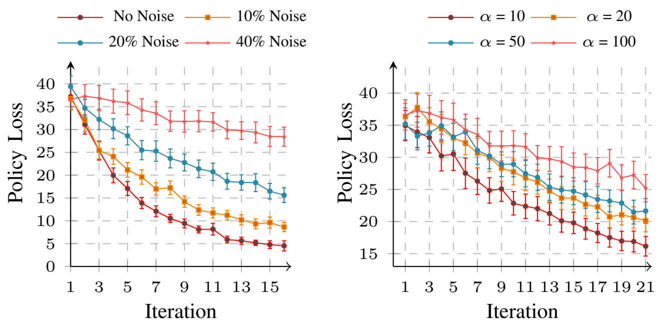


(a) Policy Loss per Iteration (b) Policy Loss per Label

Fig. 7: Averaged policy losses in 100 different 8×8 gridworlds with 48 features and queries of size 8

Figure 7 specifically presents performances of the three algorithms in 8×8 gridworlds with 48 features. Figure 7(a) shows that, per interaction with the expert, ARC outperforms uniform sampling by a large margin and it slowly catches up with the performance of the AS algorithm. Figure 7(b) shows the performance of the three algorithms in terms of per labeling effort [2] and under this criteria ARC outperforms the other two algorithms since ARC's path queries on average each contains only 2 to 3 segments. Therefore, we believe that ARC is more efficient in terms of reducing teaching burden.

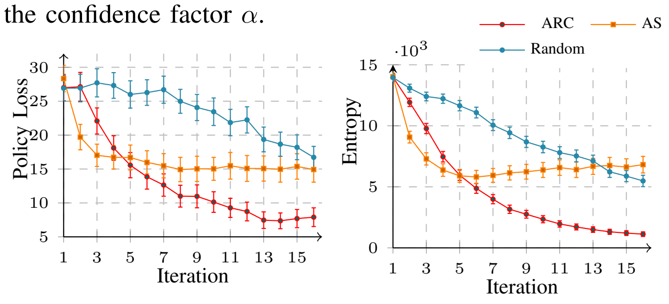
The above results agree with our hypothesis that ARC, with less teaching effort, will achieve the performance of the active sampling algorithm. At the same time, the more accurate the current belief model becomes, the more accurate the expected information gains are predicted by ARC.



(a) Policy loss under varying noise ratio with α = 100 (b) Policy loss under varying α value with 40% noise ratio

Fig. 8: Averaged policy losses of ARC under different noise ratios and α vlaues in 100 different 8×8 gridworlds with 16 features and queries of size 8

Figure 8 shows how noise could affect the performance of ARC and the performance of ARC can improve by lowering the confidence factor α.



(a) Policy Loss (b) Entropy over Rewards

Fig. 9: Average Performance on Place-An-Object Task

Figure 9 shows the policy losses and the entropy over reward functions for all three algorithms in Place-An-Object task. In this domain, we see that ARC outperforms the other two algorithms in terms of efficiency in reducing policy loss and reducing the entropy over reward functions after five iterations, with very high confidence. This is mainly because the domain is relatively small so that just a few labels can lead to a roughly accurate model for estimating information gain. Besides, one of the ground truth reward functions leads to multiple equally optimal actions in certain states, where the AS algorithm won't be able to sample all optimal actions [3] but ARC can.

[2]Here we consider providing a segmentation point, a demonstration or a label as a unit of labeling effort. Providing a demonstration at a previously demonstrated state, or a label to a previously labeled action, is considered a tenth of the original labeling effort.

[3]This is also the reason why the entropy over reward functions for AS went higher after 7 iterations.

VII. Conclusion

In this paper, we presented the ARC algorithm and discussed the major advantages of our proposed method comparing to prior work, including data efficiency, reducing human effort, enabling remote learning and allowing explicit exploration of bad behaviors. ARC uses Bayesian inverse reinforcement learning to intelligently generate trajectories with maximum information gain, asks a user to segment the trajectory into *good* and *bad* fragments and leverages these labeled state-action pairs to update its belief over reward functions. Experiments have shown that ARC can actively reduce uncertainty in Bayesian IRL, leading to reward functions that produce better policies. We analyzed the results qualitatively and quantitatively. Our results imply that directly reasoning with the belief over reward functions is a good measure of information gain, which allows the system to quickly not only reduce its policy loss but also increase its confidence over the reward function distribution. It is also shown that using our trajectory-based active learning algorithm, an agent learns more efficiently than uniform sampling and can achieve the performance of the alternative active sampling algorithm with much less labeling effort.

VIII. Future Work

One practical concern with ARC, as with all BIRL-based algorithms, is the computational cost. ARC requires running two MCMC processes per state-action pair from all the candidate queries and each and every step of the MCMC process is solving an MDP. Therefore, we are exploring different methods for efficient MCMC sampling such as using Hamiltonian dynamics [16] and for fast approximation of an MDP's value function such as Non-Parametric ALP [17]. At the same time, to better model human's capability in critiquing robot trajectories, we also plan to conduct an in-depth user study where human users will interact with a mobile robotic platform and teach it to perform various tasks by segmenting and labeling trajectories. In order to conduct such experiments with human teachers, we will also need to design an interface for efficient communication between human and the robot.

Acknowledgments

This work has taken place in the Personal Autonomous Robotics Lab (PeARL) at The University of Texas at Austin. PeARL research is supported in part by the NSF (IIS-1638107, IIS-1617639, IIS-1724157).

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Yuichiro Anzai. *Pattern recognition and machine learning*. Elsevier, 2012.

[3] Brenna Argall, Brett Browning, and Manuela Veloso. Learning by demonstration with critique from a human teacher. In *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, pages 57–64. IEEE, 2007.

[4] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[5] Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 17–24. ACM, 2012.

[6] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[7] Jaedeug Choi and Kee-Eung Kim. Map inference for bayesian inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1989–1997, 2011.

[8] Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017.

[9] Robert Cohn, Edmund Durfee, and Satinder Singh. Comparing action-query strategies in semi-autonomous agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1287–1288. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[10] Anca D Dragan Dorsa Sadigh, Shankar Sastry, and Sanjit A Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems (RSS)*, 2017.

[11] Kshitij Judah, Saikat Roy, Alan Fern, and Thomas G Dietterich. Reinforcement learning via practice and critique advice. In *AAAI*, 2010.

[12] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Teaching agents with human feedback: a demonstration of the tamer framework. In *Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion*, pages 65–66. ACM, 2013.

[13] Johannes Kulick, Robert Lieck, and Marc Toussaint. The advantage of cross entropy over entropy in iterative information gathering. *arXiv preprint arXiv:1409.7552*, 2014.

[14] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[15] Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

[16] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011.

[17] Jason Pazis and Ronald Parr. Non-parametric approximate linear programming for mdps. 2011.

[18] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51(61801):1–4, 2007.

[19] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.

[20] Qing Wang, Sanjeev R Kulkarni, and Sergio Verdú. Divergence estimation for multidimensional densities via k-nearest-neighbor distances. *IEEE Transactions on Information Theory*, 55(5):2392–2405, 2009.

[21] Jiangchuan Zheng, Siyuan Liu, and Lionel M Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *AAAI*, pages 2198–2205, 2014.

[22] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.