Active Inverse Reward Design

Sören Mindermann [1] **Rohin Shah** [1] **Adam Gleave** [1] **Dylan Hadfield-Menell** [1]

Abstract

Reward design, the problem of selecting an appropriate reward function for an AI system, is both critically important, as it encodes the task the system should perform, and challenging, as it requires reasoning about and understanding the agent's environment in detail. AI practitioners often iterate on the reward function for their systems in a trial-and-error process to get their desired behavior. Inverse reward design (IRD) is a preference inference method that infers a true reward function from an observed, possibly misspecified, proxy reward function. This allows the system to determine when it should trust its observed reward function and respond appropriately. This has been shown to avoid problems in reward design such as negative side-effects (omitting a seemingly irrelevant but important aspect of the task) and reward hacking (learning to exploit unanticipated loopholes). In this paper, we actively select the *set of proxy reward functions* available to the designer. This improves the quality of inference and simplifies the associated reward design problem. We present two types of queries: discrete queries, where the system designer chooses from a discrete set of reward functions, and feature queries, where the system queries the designer for weights on a small set of features. We evaluate this approach with experiments in a personal shopping assistant domain and a 2D navigation domain. We find that our approach leads to reduced regret at test time compared with vanilla IRD. Our results indicate that actively selecting the set of available reward functions is a promising direction to improve the efficiency and effectiveness of reward design.

[*]Equal contribution [1]Department of EECS, University of California, Berkeley, USA. Correspondence to: Sören Mindermann <soeren.mindermann@gmail.com>.

Accepted at the 1st Workshop on Goal Specifications for Reinforcement Learning, FAIM 2018, Stockholm, Sweden, 2018.

1. Introduction

We typically design AI systems in two steps: 1) determine a reward function that ranks the desirability of different states of the world; and 2) write an algorithm, e.g. planning or reinforcement learning, to optimize that reward function. Reward design, the problem of selecting an appropriate reward function, is both critically important, as it encodes the task the system should perform, and challenging, as it requires reasoning about and understanding the agent's environment in detail. For example, in the game CoastRunners, Amodei and Clark (2016) trained a reinforcement learning (RL) agent to maximize the score so it would learn to win races. Instead, it learned to drive in circles to collect points—a behavior quite distinct from racing to win. When we deploy algorithms to directly optimize a hand specified reward function, we implicitly rely on the system designer to anticipate all possible behaviors and determine appropriate incentives or penalties.

Consider the personal shopping assistant example in Fig. 1. Alice wants her robot to buy healthy foods from the supermarket. She designs a set of features that capture the ingredients and nutrients of the available products. She iterates through trial-and-error to design a reward function over these features that leads to good purchases. She rewards vitamin A, so her robot gets carrots. However, unbeknownst to Alice, the supermarket introduces a new product: energy bars, which contain vitamin A, but also the rare unhealthy ingredient *Maltodextrin*. Alice forgot to penalize *Maltodextrin* because the store originally contained no products with both *Maltodextrin* and vitamin A. She got good behavior in the training environment, even though her reward function did not generalize.

Preference inference methods infer reward functions from human-created data and can mitigate issues with reward misspecification. Inverse reinforcement learning (IRL) (Ng and Russell, 2000; Abbeel and Ng, 2004; Ziebart et al., 2008) infers a reward function from expert demonstrations. Another common approach is to learn directly from preferences between states, actions or trajectories (Fürnkranz et al., 2012; Wirth et al., 2016; 2017; Christiano et al., 2017). Inverse reward design (IRD) (Hadfield-Menell et al., 2017) is a recent preference inference method that learns directly from an observed, possibly misspecified, proxy reward function.

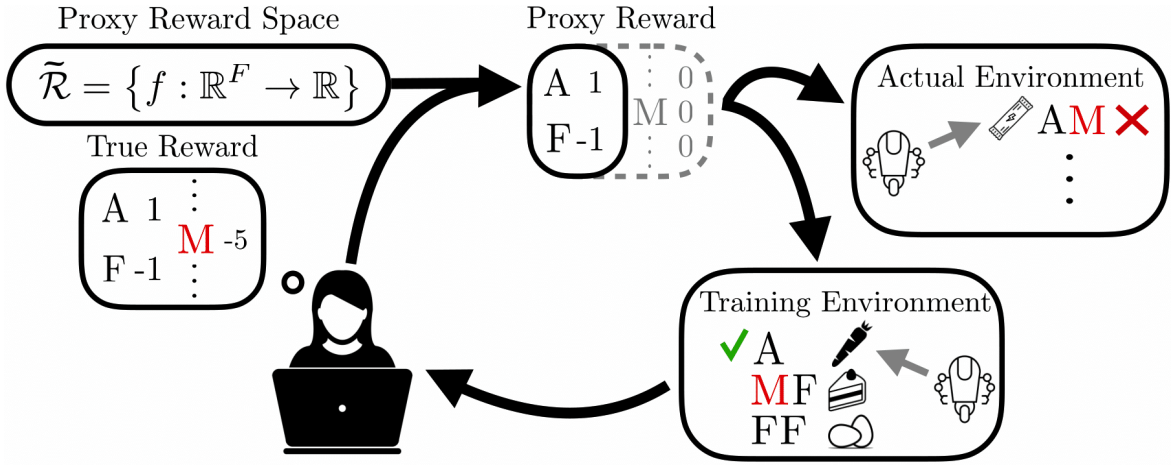

Figure 1. Illustration of failed generalization. Alice designs a reward function so that her robot buys healthy food. She tunes the reward by trial-and-error, rewarding vitamin A, and then penalizing fat (F) so that her robot buys carrots (A) but not eggs (FF). She forgets to penalize the unhealthy ingredient *Maltodextrin* (M) because the robot now already buys healthy carrots and avoids cake (MF). When deployed in a different store, the robot maximizing this *proxy* reward function buys an unhealthy energy bar which has both vitamin A and *Maltodextrin*. In this work, we show how to learn the true reward function in such cases.

The probabilistic model of IRD assumes that the designed reward function leads to approximately optimal behavior specifically *in the training environment*.

In this paper, we consider an active learning approach to IRD. We ask the designer to solve a sequence of reward design problems that we actively select to maximize information gain about the true reward. The core of our approach is that we choose the *set of proxy reward functions* available to the designer.

A natural way to design queries is to optimize over a discrete set of reward functions. We present a greedy approach to optimize this set and verify that it matches the performance of a large random search. Drawing on recent work which shows that determining the relevant features in a user's preference leads to more efficient learning (Basu et al., 2018), we consider queries where the designer specifies weights for a small set of features. We design approaches to select which features to query the designer on and the weights for the remaining features.

Our contributions are as follows: we 1) present an active learning approach to choose the set of proxy reward functions available to the system designer; 2) present an algorithm that actively selects between the *features* to query the designer on, as opposed to directly selecting between reward functions; and 3) evaluate this approach with experiments in a personal shopping assistant and a 2D navigation domain. We find that active selection leads to reduced regret at test time compared with vanilla IRD - often fully recovering the true reward function - and that binary feature-based queries, which only ask the designer to specify two numbers, lead to behavior equivalent to selecting between 10 discrete reward functions. Our results indicate that actively selecting the set of available reward functions is a promising direction for increasing the efficiency and effectiveness of reward design. In future work, we plan to apply this method to larger domains and to verify that smaller queries reduce the load on reward designers with a human subjects study.

2. Background

2.1. Inverse Reward Design

In *inverse reward design* (IRD) (Hadfield-Menell et al., 2017) the goal is to infer the true reward function the designer wants to optimize, given an observed, but possibly misspecified proxy reward function. This allows the agent to maintain a posterior over the true reward function and, e.g., avoid unintended side effects of an incorrect proxy reward function.

The key idea behind the approach is the assumption it makes about how the proxy reward function relates to the true reward function. Reward design is an inherently complex and hard-to-model process. IRD assumes that the reward designer iterates on the proxy reward function, e.g., through trial and error, until it incentivizes the correct behavior in a training environment. This implies the core assumption that IRD formalizes: *proxy reward functions are likely to the extent that they lead to high **true** utility behavior in the **training** environment.*

We will represent the decision problem faced by our agent as a Markov decision process (MDP). To simplify notation, we write reward functions $r(\xi;\, w) = w^\top \phi(\xi)$ as linear in weights w and features $\phi(\xi)$ of a trajectory ξ. We note,

however, that none of our techniques rely on this linearity assumption; they can be applied fully to arbitrary reward functions and learned features. We assume that the system designer selects a proxy reward function from a set of options $\widetilde{r} \in \widetilde{\mathcal{R}}$ so that the agent's behavior optimizes the true reward function r^* in the training environment $\widetilde{M}$. We use $\pi(\cdot|\widetilde{r}, \widetilde{M})$ to represent the designer's model of the agent's behavior given a proxy reward function and an environment. This defines the *reward design problem* (Singh et al., 2010).

In the inverse reward design problem, the agent observes $\widetilde{r}$, $\widetilde{\mathcal{R}}$, $\widetilde{M}$, and $\pi(\cdot|\widetilde{r}, \widetilde{M})$ but crucially does not observe $r^* \in \mathcal{R}$ directly, where $\mathcal{R}$ is the space of possible true reward functions. The agent must infer r^* from this information, under the assumption that $\widetilde{r}$ incentivizes approximately optimal behavior. In the next section, we describe how this assumption is formalized into an observation model.

2.2. Observation Model

We assume a fixed training MDP $\widetilde{M}$ and a training agent $\pi(\xi|\widetilde{w}, \widetilde{M})$ which defines a distribution over trajectories ξ given proxy reward $\widetilde{w}$. The key assumption of IRD is that proxy reward functions are likely to the extent that they incentivize high utility behavior in the training MDP. An optimal designer chooses $\widetilde{w}$ so that the expected true value $\mathbb{E}[{w^*}^\top \phi(\xi)|\xi \sim \pi(\xi|\widetilde{w}, \widetilde{M})]$ is maximal. We model the approximately optimal designer:

$$P(\widetilde{w}|w^*, \widetilde{M}) \propto \exp\left(\beta\, \mathbb{E}\left[{w^*}^\top \phi(\xi) \mid \xi \sim \pi(\xi|\widetilde{w}, \widetilde{M})\right]\right) \tag{1}$$

where β controls how close to optimal the designer is assumed to be.

Cost of inference. Computing the normalization constant $\widetilde{Z}(w^*)$ for the likelihood (1) is the primary difficulty as it involves integrating over all possible proxy rewards and solving a planning problem for each. The resulting feature expectations $\tilde{\phi} = \mathbb{E}[\phi(\xi)|\xi \sim \pi(\xi|\widetilde{w}, \widetilde{M})]$ are memoized. Hadfield-Menell et al. (2017) explores methods to approximate $\widetilde{Z}(w^*)$ by sampling.

Conversely, the normalization constant $Z(\widetilde{w})$ for the *posterior* $P(w^*|\widetilde{w}, \widetilde{M}) \propto P(\widetilde{w}|w^*, \widetilde{M})P(w^*)$ integrates over $w \in \mathcal{R}$, and is cheap to compute once the feature expectations $\tilde{\phi}$ have been memoized. The primary cost is to evaluate the average reward $w^T\tilde{\phi}$ on memoized feature expectations for each w. This does not involve planning and can be computed in a single matrix multiplication for finite $\mathcal{R}$ and $\widetilde{\mathcal{R}}$. Even for nonlinear reward functions, $Z(\widetilde{w})$ is fairly cheap to compute. Approximate inference methods such as MCMC do not need to compute this normalizer at all.

2.3. Related work

A variety of approaches for learning reward functions have been proposed. In *inverse reinforcement learning* (IRL) (Ng et al., 2000; Ziebart et al., 2008), the agent observes demonstrations of (approximately) optimal behavior, and infers the reward function that would produce that behavior. Reward functions have also been learned from *expert ratings* (Daniel et al., 2014) and *human reinforcement* (Knox and Stone, 2009; Warnell et al., 2017).

Methods that learn reward functions from preferences, surveyed in Wirth et al. (2017), are particularly relevant to our work. Christiano et al. (2017) learn a reward function from preferences over pairs of trajectories, by searching over trajectories sampled from the learned policy and querying the user about pairs on which an ensemble of reward predictors disagrees. A similar setup is used in Wirth et al. (2016) based around Relative Entropy Policy Search (REPS) instead of deep RL. It is also possible to learn reward functions from preferences on actions (Fürnkranz et al., 2012) and states (Runarsson and Lucas, 2014). APRIL (Akrour et al., 2012) chooses new trajectories to compare to the current best, based on the *approximate expected utility of selection*.

Our work is most similar to Sadigh et al. (2017), which finds queries through gradient-based optimization directly in trajectory space in a continuous environment. Their objective is expected volume removed from the hypothesis space by the query, which has a similar effect as our method of optimizing for information gain. We differ from prior work by learning from preferences over reward functions in a particular environment, rather than trajectories, states or actions. Most of our queries are not binary, some taking the form of infinite sets, but these can still be seen as preference queries. Of course, a preference over reward functions is a preference over the corresponding trajectories, but by working directly with reward functions, we can create more targeted queries, optimizing directly in the relatively small, structured space of reward functions, instead of the much larger space of trajectories. This lets us recover the true reward function, leading to good generalization to new environments. To our knowledge, no other method has demonstrated this property.

3. Active Selection of Queries

In vanilla IRD, the designer selects an approximately optimal proxy $\widetilde{w}$ from a proxy space $\widetilde{\mathcal{R}}$ which is large – possibly equal to the whole reward space $\mathcal{R}$. In this work, they select the reward function from small sets $\widetilde{\mathcal{R}}_t \subset \widetilde{\mathcal{R}}$ instead. We call these sets *queries*. Reducing the size of the set simplifies the choice for the designer and, as we will see, also

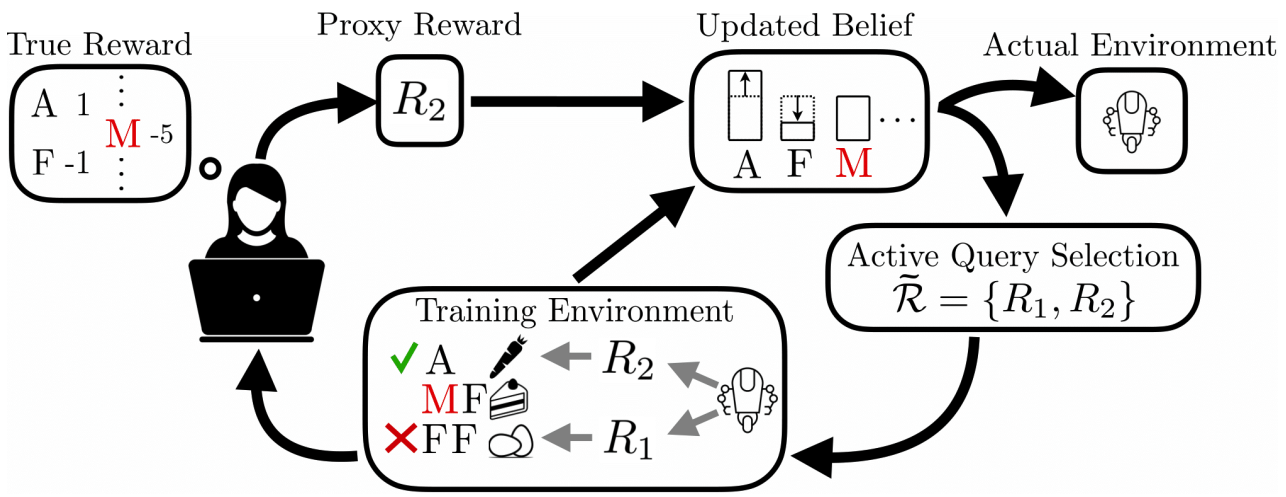

Figure 2. Illustration of active IRD. Active query selection: The large proxy space in figure 1 is replaced with a small query (in this case of size 2 for illustration) whose answer is most informative given the *training environment*. The reward functions lead to different product choices (policies) and the designer chooses the best one according to the true reward. Alice returns the reward function which leads to her preferred behavior. The belief is updated based on the range of behaviors implied by the reward functions. After a number of iterations, the robot is safely deployed in the actual environment.

improves regret in test environments while saving computation. We introduce discrete queries, small unstructured sets of reward functions, and feature queries, affine subspaces of $\widetilde{\mathcal{R}}$ that differ only in the weight given to particular features.

Active selection criterion. How do we select queries given the current belief $P(w^*)$? An active learning criterion, or *acquisition function*, should return a high value for queries that are likely to convey useful information about w^*.

A common class of information-theoretic acquisition functions used in active learning picks queries on which the current model is uncertain. Uncertainty is often expressed as the predictive entropy:

$$\mathcal{H}[\widetilde{w}|\widetilde{\mathcal{R}}_t, \mathcal{D}_t] = -\sum_{\widetilde{w}_t \in \widetilde{\mathcal{R}}_t} P(\widetilde{w}_t|\mathcal{D}_t) \log P(\widetilde{w}_t|\mathcal{D}_t), \quad (2)$$

where $\mathcal{D}_t = \{\widetilde{w}_{1:t-1}, \widetilde{\mathcal{R}}_{1:t-1}\}$ is the set of previously answered queries and $P(\widetilde{w}_t|\mathcal{D}_t)$ is the predictive probability that the user picks $\widetilde{w}_t \in \widetilde{\mathcal{R}}_t$.

Uncertainty maximization can be sufficient for active supervised learning, where the user is queried to label a single example, e.g. in (Gal et al., 2017). However, when the query answers are noisy, uncertainty maximization can lead to queries that are optimized to be close to the decision boundary, i.e. close calls. This happens easily when queries are *sets* of options, e.g. the algorithm could choose to present a choice between reward functions that lead to identical behavior.

Therefore, we also want to minimize a second term, the expected conditional entropy $\mathbb{E}_{P(w^*|\mathcal{D}_t)}\mathcal{H}[\widetilde{w}|\widetilde{\mathcal{R}}_t, w^*]$. This ensures that the user can decide the answer well in expectation. Substracting this from (2) gives the mutual information $\mathcal{I}(w^*, \widetilde{w}|\widetilde{\mathcal{R}}_t, \mathcal{D}_t)$, also known as *expected information gain*. We estimate this measure by sampling.

3.1. Discrete queries

A natural way to design queries is to optimize over a discrete set of reward functions. A discrete query is of the form $\widetilde{\mathcal{R}}_t = \{\widetilde{w}_1, \ldots, \widetilde{w}_k\}$ for some k. As in regular reward design, the user is assumed to choose proxies by observing the effects the reward functions have on the agent's behavior in the training environment. The features therefore do not have to be interpretable.

3.1.1. Computational efficiency

Discrete queries allow us to maximize the information learned about the true reward function while minimizing the number of reward functions for which we have to plan (or learn) a policy. We can learn more about the true reward than IRD does, given the same number of plans computed.

To this end, we initially sample a proxy space $\widetilde{\mathcal{R}}_{pool} \subset \mathcal{R}$ of size $k_{pool} \ll |\mathcal{R}|$, a hyperparameter that trades off between computation (planning calls) and the sample efficiency of user answers. We actively select the query from $\widetilde{\mathcal{R}}_{pool}$ since query selection is expensive, but when we get the user answer, we compute the belief update over all of $\mathcal{R}$, so we can still find the optimal reward function. For every proxy re-

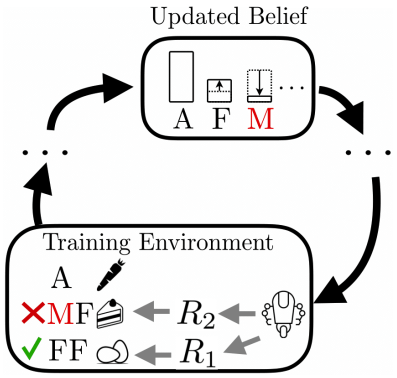

Figure 3. Continuation of the example in figure 2 - after the second query, we infer that Maltodextrin (M) is worse than fat (F) by comparing between two *suboptimal* choices - cake and eggs. Full IRD cannot infer this because it only observes that the human most prefers carrots.

ward function $\widetilde{w}$, to compute its likelihood (Equation 1) we need its implied feature expectations $\phi(\xi), \xi \sim \pi(\xi|\widetilde{w}, \widetilde{M})$. Since feature expectations are computed through (expensive) planning, we refer to them as *plans*. Before user interaction, we calculate and memoize plans for all proxies in $\widetilde{\mathcal{R}}_{pool}$. When evaluating a query $\widetilde{\mathcal{R}}_t \subset \widetilde{\mathcal{R}}_{pool}$, we reuse the pre-computed plans. This means that no plans have to be computed at interaction time for active query selection or inference. Holding the number of features constant, at interaction the algorithm is therefore independent of the difficulty of the training environment. Moreover, computation for query selection only increases linearly with the number of features.

This is a key efficiency advantage over vanilla IRD, which computes plans for normalization (section 2.2) but does not reuse them to learn about the user's preferences between *suboptimal* plans. In other words, vanilla IRD merely learns which proxy $\widetilde{w} \in \widetilde{\mathcal{R}}_{pool}$ maximizes the reward $w^{*T}\tilde{\phi}$ (assuming a perfectly rational user), which means that $|\widetilde{\mathcal{R}}|$ possible outcomes could be inferred *a priori*. Instead, our method can learn a preference ordering on $\widetilde{\mathcal{R}}_{pool}$, which means $k_{pool}!$ outcomes can be inferred in principle. This ordering conveys the maximal amount of information about w^* that can be learned from computing plans only for proxies in $\widetilde{\mathcal{R}}_{pool}$, since the user's answers can be perfectly predicted using this ordering. Our experiments show that a series of small random queries indeed reduces the entropy $\mathcal{H}[w^*]$ more than IRD does, and even more than a series of large random queries. This leads to the result in figure 3.

Greedy query selection. Searching over all queries of size-k requires $\binom{k_{pool}}{k}$ evaluations of the expected information gain. We therefore grow queries greedily up to size k, requiring only $O(k_{pool}k)$ evaluations. Empirically we find this compares favorably to random search with a much larger number of evaluations.

3.2. Feature queries

While *small* discrete queries are computationally attractive, it can be impractical for users to pick from a *large*, unstructured set, even though this may convey more information about the true reward. We therefore propose large structured *feature queries*, where the user tunes a subset of the individual weights while the other weights are fixed. In other words, feature queries are low-dimensional affine subspaces of the proxy reward function space. This decomposes the full reward design problem (a search over a potentially high-dimensional space) into a series of reward design queries where the user can judge the effects of a few individual features that are currently most informative to tune. We could imagine a graphical user interface in which the user can move sliders for each weight, and see the effect on the agent behavior.

Let $(\phi^1(s), \ldots, \phi^F(s)) = \phi(s)$ be the feature function. We define a feature query $\widetilde{\mathcal{R}}_S$ for a set of free weights $S \subset \{1, \ldots, F\}$ and a set of fixed weights with indices $\{1, \ldots, F\} \setminus S$ to be the set of vectors in $\mathbb{R}^F$ which has all combinations of $w_i \in \mathbb{R}$ (or a bounded subset of $\mathbb{R}$) for $i \in S$ and fixed real values for the other weights w_i for $i \in \{1, \ldots, F\} \setminus S$. The *size* of the query is given by $|S|$.

Actively selecting feature queries sacrifices some of the computational advantages of discrete queries for higher usability and sample-efficiency of user responses.

3.2.1. Feature query optimization

Recall that we want to choose the feature query that maximizes the expected information gain. There are two variables to optimize over: the indices of the free features and the values of the fixed features. Our best-performing algorithm greedily searches over combinations of free feature indices and optimizes the fixed weights for each combination. We also tested a cheaper algorithm that leaves the fixed weights at 0 or the current prior average.

We greedily add free features to maximize expected information gain. Tuning the fixed weights is more difficult as we are optimizing over a continuous space $\mathbb{R}^{F-k}$. We tried two optimization methods, gradient descent and random search. To see that our objective function is a differentiable function of the $F-k$ fixed weights in the query, note that we can use a differentiable planning algorithm, such as value iteration (Tamar et al., 2017). Other differentiable planning algorithms have been introduced recently (Srinivas et al., 2018; Buesing et al., 2018; Guez et al., 2018) but value iteration was sufficient for our purposes. Similar to (Sadigh et al., 2017), who optimize an active learning objective di-

rectly in continuous trajectory space, we optimize the fixed weights directly in an $(F-k)$-dimensional subspace of reward function space. We observed that simple gradient descent converges to a (local) maximum in a small number of steps, usually $\sim$10. Additionally, we used a small random search over $\mathbb{R}^{F-k}$ to find a good initialization. This improves results considerably and helps mitigate the effects of local maxima. Random search by itself provides a considerable benefit and can be used for non-differentiable planners. Our simple optimization scheme may be improved upon in future work.

Discretization. Exactly evaluating the expected information gain is only tractable for finite queries. We therefore discretize the free features in $[-1,1]^k$. A coarse discretization is used during query selection, while a finer grid is used for the actual user input.

4. Evaluation

We use two metrics for evaluation: (1) the entropy of the agent's belief $\mathbb{H}[w^*]$, and (2) the regret obtained by optimizing the posterior mean of the robot's belief across a set of novel test environments when the agent optimizes for the posterior average reward $\bar{w} = \mathbb{E}[w^*|\mathcal{D}]$. We selected 20 queries per experiment and averaged results over 100 runs.

We designed our experiments to answer the following questions: (1) how performance varies as we increase discrete query size from minimal (2 proxy rewards) to maximal (full IRD), (2) can active query selection outperform random queries and can greedily grown queries match a large random search, (3) how do the two feature query optimizations (selecting free features and also optimizing fixed features) perform compared to a random baseline, (4) which query type is most sample efficient?

We tested on two distributions of environments: stores ('shopping' domain) with many different products, and a 2D navigation task with hot and cold objects to which the agent has to minimize resp. maximize its distance. The shopping domain consists of randomly generated environments with 100 actions and states, and 20 normally distributed features $\phi(s)$ for each state s. Action i deterministically leads to state i. Therefore, the agent merely has to calculate which action (product) leads to the highest reward. We used an approximately optimal agent which is differentiable as a function of the reward.

Our 2D navigation task, which we term 'chilly worlds', consist of a 10×10 grid with random walls and 20 objects in random positions, which define a 20-dimensional feature vector of distances to each object. The agent is a simple differentiable implementation of value iteration.

4.1. Results

Experiment 1: Discrete query sizes and full IRD (left in figure 4). For the first experiment, we tested the performance of random queries from size 2 to 10000 to test our hypothesis that full IRD (equivalent to a maximal query size) can be outperformed if we also learn preferences between suboptimal behaviors. We reduced the size of the reward space $\mathcal{R}$ to 10000 to be able to do exact inference for the full IRD baseline. Here we used the maximal proxy space $\widetilde{\mathcal{R}} = \mathcal{R}$. We can see full IRD as a special case of active IRD in this setting, with a query size of 10000. Full IRD was run 20 times to show its convergence behavior, although it would normally be run only once.

Figure 4 shows that we can quickly outperform the generalization performance of full IRD simply by asking random queries which compare between suboptimal proxies. Using our method, the entropy drops to nearly zero, indicating that the true reward function has been successfully identified, while full IRD retains some uncertainty about the true reward function. As expected, increasing the query size initially helps. Interestingly, larger queries already hurt starting from a size of > 10, with full IRD performing worst. Note that performance on the first query strictly increases with query size, and a larger query size only decreases performance once the entropy has been somewhat reduced. This underscores the point made in figure 3: we want to learn preferences between *suboptimal* options once we know what optimal behavior is (and smaller queries are more likely to only contain suboptimal choices). For all further experiments we increased $|\mathcal{R}|$ to 10^6.

Experiment 2: Discrete queries and full IRD (right in figure 4). Next, we compared our active learning algorithm for discrete queries to a baseline of random queries and compared the greedy selection to an expensive large query search. The hyperparameter k_{pool} (size of the proxy space $\widetilde{\mathcal{R}}_{pool}$) was set to 100. Figure 4 shows for a query size of 5 that greedy active query selection matches a random search of size 10000 over queries (the lines for the shopping domain are indistinguishable), confirming previous literature which found empirically that greedy algorithms can approach optimality for information gain (Sharma et al., 2015). Data for query sizes $2, 3$ and 10 are not presented here, but are summarized in figure 5. Random queries and full IRD (shades of grey) are clearly outperformed by active selection. Another observation is that active selection becomes more important the more entropy has already been reduced. This is likely because a random query is unlikely to target the small amount of remaining uncertainty at later stages.

Experiment 3: Feature query selection methods. Figure 5 shows feature queries with 1 free feature. The conditions

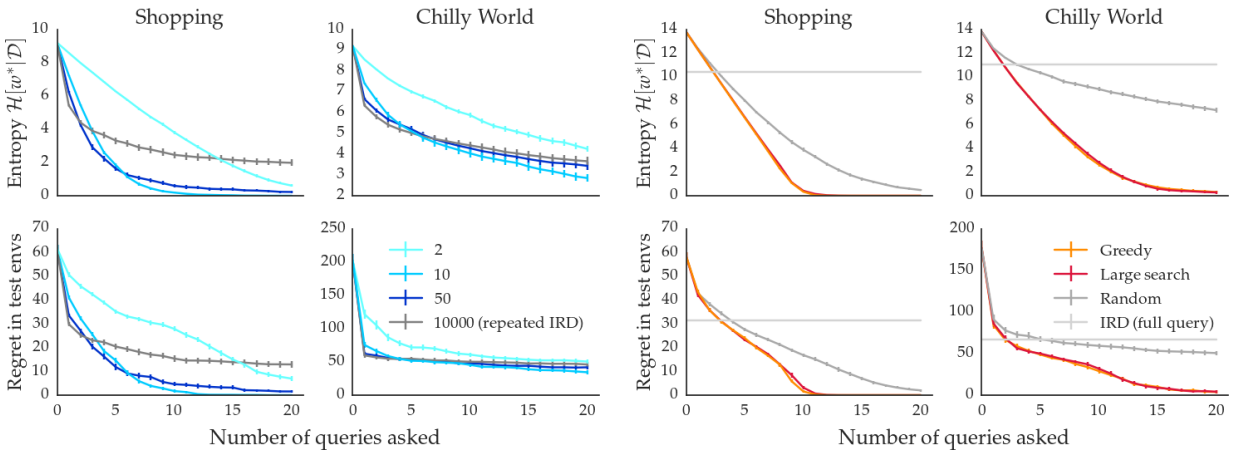

Figure 4. **Left:** Random discrete queries of sizes 2, 10, 50, and repeated exact full IRD with a proxy reward space of size 10000. Larger queries lead to faster initial learning, but can lead to worse final performance. **Right:** Discrete query selection methods (query size 5) and full IRD. Note that the cheap greedy selection matches the expensive search and IRD remains too uncertain to generalize well.

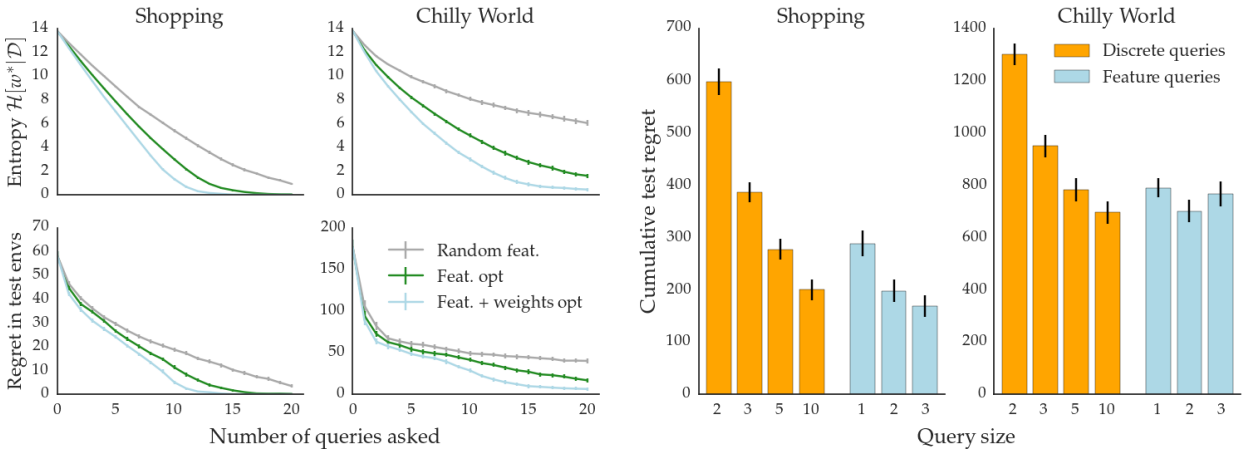

Figure 5. **Left:** Feature query selection methods with 1 free feature. Comparing 1) Unoptimized random feature query 2) free feature actively selected 3) additionally, fixed weights optimized. There is a trade-off between computational efficiency (2) and sample efficiency (3). **Right:** Cumulative test regret (area under curve) for discrete and feature queries of different sizes using the best-performing selection methods.

are: (1) fully random queries, (2) only free features actively selected, and (3) both free features and fixed weights actively selected. In both (1) and (2), the fixed weights are set to 0. Optimizing fixed weights can cause a significant benefit and is therefore recommended if the sample efficiency of answers is crucial. If computational efficiency is crucial, simple greedy selection of the free features is a cheap and adequate alternative.

Sample efficiency: Comparison between differently sized discrete and feature queries. We summarized the sample efficiency for different query types and sizes as the area under the curve (cumulative test regret). Figure 5 shows that a larger query size significantly improves discrete, greedily optimized queries. The data on feature queries indicate that tuning only a single feature at a time can be effective (matching size 10 discrete in the 'chilly world' environment), which is attractive from a usability point of view. Binary feature-based queries, which only ask the designer to specify two numbers, lead to behavior equivalent to selecting between an unstructured set of 10 discrete reward functions.

5. Discussion

Summary. This paper introduced active learning methods for inverse reward design which use the uncertainty in the IRD posterior to extract as much information as possible from the user's choice of reward function. The presented approach helps identify the true reward, leading to better generalization in test environments; improves the computational limitations of IRD; and potentially helps the user be more accurate and save time. We evaluated on two distributions of environments and explored various optimization methods and trade-offs between computation and sample-efficiency.

Limitations and future work. There is significant potential to scale active inverse reward design in future work. Firstly, the most straightforward extension would simply precompute a set of policies from proxies in larger environments with linear reward functions. As discussed, our discrete queries are independent of the environment's difficulty and scale linearly with the number of features. Secondly, reward functions could also be defined on learned features. The querying process described in this paper could itself be repeated so that new features can be learned each time using improved reward estimates. This would address the limitation that initially sampled proxies may not lead to interesting behavior. Thirdly, the use of nonlinear reward functions goes well with our approach as we only need to evaluate trajectory returns to calculate likelihoods. During inference and query selection, the reward over distributions of buffered trajectories would be evaluated instead of computing inner products with feature expectations, which is still fast and does not involve planning.

We hope that our work inspires new methods for reward design, e.g. new types of reward design queries. Our feature queries ask the user to tune weights for individual features, but these could instead be informative latent variables in reward function space. Overall, we are excited about the implications active IRD has not only in the short term, but also about its contribution to the general study of the value alignment problem.

References

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *Twenty-first international conference on Machine learning - ICML '04*, page 1.

Akrour, R., Schoenauer, M., and Sebag, M. (2012). APRIL: Active preference learning-based reinforcement learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7524 LNAI, pages 116–131.

Amodei, D. and Clark, J. (2016). Faulty reward functions in the wild.

Basu, C., Singhal, M., and Dragan, A. D. (2018). Learning from richer human guidance: Augmenting comparison-based learning with feature queries. *CoRR*, abs/1802.01604.

Buesing, L., Weber, T., Racaniere, S., Eslami, S. M. A., Rezende, D., Reichert, D. P., Viola, F., Besse, F., Gregor, K., Hassabis, D., and Wierstra, D. (2018). Learning and Querying Fast Generative Models for Reinforcement Learning.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4302–4310.

Daniel, C., Viering, M., Metz, J., Kroemer, O., and Peters, J. (2014). Active reward learning. In *Robotics: Science and Systems*.

Fürnkranz, J., Hüllermeier, E., Cheng, W., and Park, S.-H. (2012). Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine Learning*, 89(1-2):123–156.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep Bayesian Active Learning with Image Data. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1183–1192.

Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., and Silver, D. (2018). Learning to Search with MCTSnets.

Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., and Dragan, A. (2017). Inverse reward design. In *Advances in Neural Information Processing Systems*, pages 6768–6777.

Knox, W. B. and Stone, P. (2009). Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM.

Ng, A. and Russell, S. (2000). Algorithms for inverse reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

Runarsson, T. P. and Lucas, S. M. (2014). Preference learning for move prediction and evaluation function approximation in othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):300–313.

Sadigh, D., Dragan, A., Sastry, S., and Seshia, S. A. (2017). Active preference-based learning of reward functions. In *Robotics: Science and Systems (RSS)*.

Sharma, D., Kapoor, A., and Deshpande, A. (2015). On Greedy Maximization of Entropy. *Proceedings of the 32nd International Conference on Machine Learning*, 37:1330–1338.

Singh, S., Lewis, R. L., and Barto, A. G. (2010). Where Do Rewards Come From? *Proceedings of the International Symposium on AI Inspired Biology - A Symposium at the AISB 2010 Convention*, pages 2601–2606.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. (2018). Universal planning networks. *CoRR*, abs/1804.00645.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2017). Value iteration networks. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 4949–4953.

Warnell, G., Waytowich, N., Lawhern, V., and Stone, P. (2017). Deep tamer: Interactive agent shaping in high-dimensional state spaces. *arXiv preprint arXiv:1709.10163*.

Wirth, C., Akrour, R., Neumann, G., and Fürnkranz, J. (2017). A Survey of Preference-Based Reinforcement Learning Methods. *Journal of Machine Learning Research*, 18(136):1–46.

Wirth, C., Furnkranz, J., Neumann, G., et al. (2016). Model-free preference-based reinforcement learning. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 2222–2228.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA.