Asssited Robust Reward Design

Jerry Zhi-Yang He [1] **Anca D. Dragan** [1]

Abstract

Real-world AI systems often need complex reward functions. When we define the problem the AI needs to solve, we pretend that an engineer specifies this complex reward exactly and it is set in stone from then on. In practice, however, reward design is an *iterative* process: the engineer designs a reward, eventually encounters an environment where the reward incentivizes the wrong behavior, revises the reward, and repeats until convergence. What would it mean to rethink AI to formally account for this iterative nature of reward design? We propose that the AI system needs to not take the specified reward for granted, but rather have *uncertainty* about what the reward is, and account for the future iterations as *future evidence* it will receive. We contribute an AI-assisted reward design method that speeds up the design process by taking control over this future evidence: rather than letting the designer eventually encounter environments that require revising the reward, the system actively exposes the designer to the environments that have the most potential to narrow down what the reward should be. We test this method in an autonomous driving task, and find that it more quickly improves the car's behavior in held-out environments, and iteratively proposes environments that are "edge cases" for the current reward.

1. Introduction

The job of an AI agent is to maximize its cumulative reward. From the perspective of the agent, it is as if the reward function fell from the sky: it is just there, it is something embedded in the very problem definition. In reality, there is a lot happening behind the scenes, where a human designer has to actually specify this reward. This is almost always an iterative process, where the reward specification evolves over time as the designer tests the agent in more and more environments.

Take, for instance, autonomous cars. They have to correctly evaluate and balance safety, efficiency, comfort and abiding by the law. An engineer might start by looking at some representative environments, and specifying a reward function that leads to the behavior they want in each of them. But working well in this set of environments is not enough — the reward function has to incentivize the right behavior in *any* environment the car will encounter in its lifetime[1]. This initial reward function is just the first of many to come. The engineer will test the car in further environments, maybe in simulation, or maybe by test-driving in the real world. Almost inevitably at some point, optimizing for the same reward will lead to some undesirable behavior. The engineer will then *revise* the initial reward, and repeat the process. If they are lucky, by the time they deploy the car, they will have converged to a reward function that actually captures what they want. But for many systems, this iteration might continue well into deployment: the car gets used, encounters some edge-case environment they did not account for during development — perhaps just the wrong combination of who is around, in what configuration, how the road is laid out, etc. — and the engineer revises the reward again to take the new environment into account.

The AI's job in this is to maximize whatever the current specified reward is. We argue that it should also be the AI's job to help the engineer figure out the reward — we refer to this as *assisted reward design*.

To capture this formally, we can no longer use a model where the AI treats the reward as set in stone. We have to formulate the problem in way that accounts explicitly for the iterative nature of reward design. To that end, we use a model in which the AI has *uncertainty* about the reward function. Every revision of the reward done by the designer does *not* define the reward function fully, but is instead an *evidence* about it. The AI's ultimate goal is to do well with respect to the true reward at deployment time.

[1]Department of Computer Science, University of California, Berkeley, California, USA. Correspondence to: Jerry Zhi-Yang He <hzyjerry@berkeley.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

[1]We assume that everything else such as world model, planning algorithms, etc. work out well for the agent. In this paper we focus only on the reward.

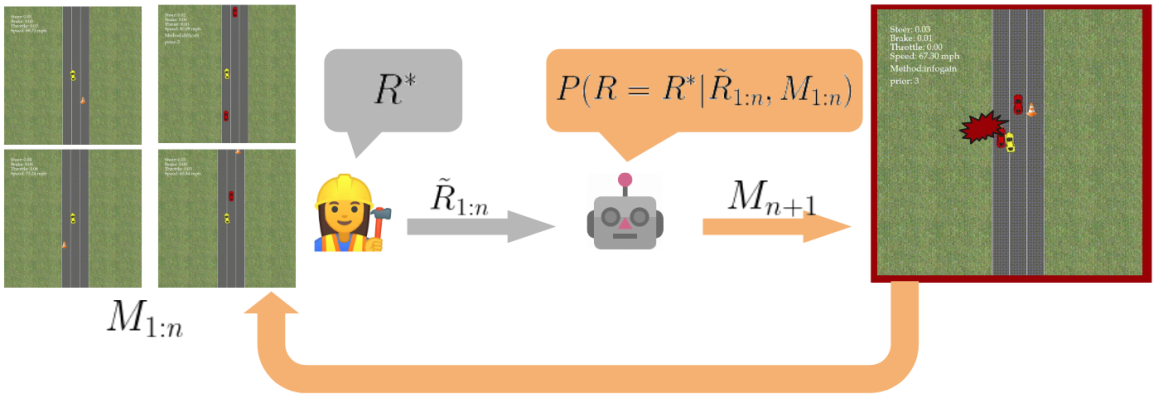

Figure 1: Assisted Reward Design Process. Left: reward designer specifies proxy reward on a set of environments for highway merging. Right: our algorithm takes the current design, and queries the designer with a new environment. Our algorithm effectively finds environments at the long-tail of environment distributions.

How does this formulation lead to design assistance? A naive agent would passively estimate the reward from observations so far, and not contribute to the design process. However, this would only be optimal when there is no more evidence to be gained — when the reward is set and there are no more future iterations. In contrast, the iterative nature of the design process means the agent can and should account for the future iterations of the reward as *future observations*. The agent can then act to make these observations more informative, by exposing the designer to environments where revisions are most likely needed and will have high expected information gain. Rather than letting the designer eventually encounter edge-case environments, our method proposes candidate environments and asks for reward revisions, inserting itself in the loop of the design process.

This has two implications. For systems where the designer would converge to a good reward before deployment time, our method speeds up the iterations by focusing on informative environments. Perhaps more importantly, for case where the designer might still otherwise have the wrong reward when deploying the system, our method has the potential to expose the edge-case environments that would fail during deployment, prompting fixes in the reward ahead of time.

We test our method in an autonomous driving domain, both in simulation with ground truth rewards, as well as for a real reward design task with end users. We find that it leads to improvements in behavior on difficult test scenarios more quickly than the passive baseline, and that the environments it produces are ones in which the current reward estimate fails.

Overall, we are excited to contribute a framework in which AI systems do not just optimize what we design for them, but actually help us design better and more robust reward functions. Our work fits broadly in reward learning (Ng & Russell, 2000; Abbeel & Ng, 2004; Ramachandran & Amir, 2007; Wirth et al., 2017; **?**; Fu et al., 2019; Jain et al., 2015; Bajcsy et al., 2017; Ziebart et al., 2008; Ratliff et al., 2006; Bloem & Bambos, 2014; Ho & Ermon, 2016; Levine & Koltun, 2012; Loftin et al., 2014), and is related to methods that actively query for demonstrations (Lopes et al., 2009; Brown et al., 2019), comparisons (Christiano et al., 2017; Sadigh et al., 2017), advice (Odom & Natarajan, 2015), or scalar feedback (Reddy et al., 2019), especially those that query states or environments – our focus is on expert users that are actually specifying rewards, and query for revisions by tapping into the natural iterative reward design process. Our finding is that by keeping a belief over reward functions, agents can identify edge-cases where the currently probable rewards contradict, and help the expert iterate.

2. Assisted Reward Design

2.1. Problem Setup

Let an "environment" M be an MDP without the reward function. We assume access to a large set of such environments at development time, $\mathcal{M}_{\text{devel}}$. We assume this is a large set that the designer cannot exhaustively test. This can be, for instance, all the environments in a several-million mile dataset of human driving for the autonomous car domain. Or, it can be a set implicitly induced by a set of parameters an expert designs, like possible configurations of objects, roads, and agents. Finally, it can also be implicitly induced by a generative model of the world.

The agent will act at deployment time in a potentially different set of environments $\mathcal{M}_{\text{deploy}}$, which we do *not* have access to. Our method works best when can over-parametrize environments to induce a vast set $\mathcal{M}_{\text{devel}}$ that includes everything we might see at deployment time in $\mathcal{M}_{\text{deploy}}$ — otherwise, if $\mathcal{M}_{\text{deploy}}$ is allowed to be drastically different,

and if $\mathcal{M}_{\text{devel}}$ is not enough to disambiguate the reward function, this still leaves the robot exposed to potential failures after deployment.

We further assume access to a space of reward functions parametrized by $w \in W$ — these can be linear weights on pre-defined features, or weights in a neural network that maps raw input to scalar reward. We denote by w^* the parameters of the desired reward function, which would induce the desired behavior in $\mathcal{M}_{\text{devel}}$ and $\mathcal{M}_{\text{deploy}}$. We denote by $\xi_{w,M}$ the trajectory (or policy, but for the purpose of this work we consider deterministic MDPs with set initial states) that is optimal with respect to the reward induced by w in environment M, $\xi_{w,M} = \arg\max_\xi R_w(\xi; M)$. Therefore, our assumption is that there exists a w^* such that $\xi_{w^*,M}$ is the desired behavior for any $M \in \mathcal{M}_{\text{devel}}$ and any $M \in \mathcal{M}_{\text{deploy}}$. We do *not* have access to w^*.

We first detail how unassisted reward design works, then define the assisted reward design problem and introduce our method.

2.2. The Process of Unassisted Reward Design.

The designer can take a subset of environments $\mathcal{M} \subseteq \mathcal{M}_{\text{devel}}$ and specify a reward function (via parameters $\tilde{w}$) that leads to good behavior on those environments. In one-shot design, they would select a set $\mathcal{M}$, assume it is representative of $\mathcal{M}_{\text{deploy}}$, design a $\tilde{w}$ for that $\mathcal{M}$, and deploy the system with $\tilde{w}$ (never changing the reward again). However, in reality, this is an iterative process. They start with an $\mathcal{M}_0$, design a $\tilde{w}_0$, and eventually encounter during their testing a new environment M' on which optimizing $\tilde{w}_0$ does not lead to desirable behavior. Then, they would augment their set to $\mathcal{M}_1 = \mathcal{M}_0 \cup \{M'\}$, and re-design the reward:

$$\mathcal{M}_0 \rightarrow \tilde{w}_0 \rightarrow \mathcal{M}_1 = \mathcal{M}_0 \cup \{M'\} \rightarrow \tilde{w}_1 \rightarrow ...$$

This might continue into deployment, if the reward at the time of deployment still fails to induce desired behavior on $\mathcal{M}_{\text{deploy}}$. Implicitly, at every step along the way, the AI agent treats the current $\tilde{w}_i$ as equivalent to w^*.

2.3. The Assisted Reward Design Problem.

The main difference in assisted reward design is that the AI agent no longer treats proxy reward $\tilde{w}$ as equivalent to w^* — rather, it is a proxy observation of w^* based on the current $\mathcal{M}_i$. We can thus formulate the assisted reward design as a POMDP problem with state uncertainty. The goal of the assisted reward design is to minimize regret over deployment $\mathcal{M}_{\text{deploy}}$.

State, actions, observations, transitions. In the assisted reward design problem, the notion of a "state" is a set $\mathcal{M}$ of environments — this is what changes over time and is directly observable — along with the hidden state w^*. The agent (and designer) starts at state $(\mathcal{M}_0, w^*)$ (environments either chosen by the designer to be representative, or simply the empty set). We can transition from a state to the next by "acting", i.e. adding one more environment M_i to consider, $\mathcal{M}_{i+1} = \mathcal{M}_i \cup M_i$. Every time we enter a state $(\mathcal{M}, w^*)$, we receive an observation $\tilde{w}$ about w^*.

The first step to providing assistance is for the AI agent to know that the reward it is receiving from the designer, $\tilde{w}_i$, is not necessarily one and the same as w^* — rather, it is a proxy that emulates the behavior w^* would produce *only on the current* M_i. The second step is to be able to interpret $\tilde{w}$ as evidence about w^*, and use all past evidence to obtain a belief of w^*. Of course, just that is not helping the design process yet. The key (third) step is to be able to account for (and thus influence) the future evidence in order to get a more successful reward estimate. We next introduce the observation model for this POMDP, methods for computing belief distribution in Section 2.4.

Observation Model. Previous work on Inverse Reward Design (Hadfield-Menell et al., 2017) introduced an observation model that maps the true reward w^* and designer's training environment set $\mathcal{M}$ to a distribution over the proxy reward $\tilde{\omega}$:

$$P_{\text{design}}(\tilde{\omega}|\omega^*, \mathcal{M}) \propto \exp\left(\sum_{i=1}^{|\mathcal{M}|} \beta\big[R_{w^*}(\xi_{\tilde{w}_i,M_i})\big]\right) \quad (1)$$

with β controlling how close to optimal we assume the person to be. This distribution informs the AI agent that the reward it is receiving proxy rewards from an approximately optimal designer who only looks at the current M_i. We build on this work here, adopting the observation model for any intermediate iteration stage.

Objective. The objective of assisted reward design is to achieve high reward at deployment time. What the robot controls is a sequence of actions, i.e. a sequence of environments it can propose. These lead to observations about w^*. At deployment time, the robot uses its belief to generate optimal trajectories, and cumulates reward according to w^*:

$$\min_{M_1,..,M_T} \mathop{\mathbb{E}}_{M\sim\mathcal{M}_{\text{deploy}}} \max_{\xi} \mathop{\mathbb{E}}_{w\sim b(\mathcal{M}_1,..\mathcal{M}_T)} [R_{w^*}(\xi, M)] \quad (2)$$

where $b(\mathcal{M}_1, .., \mathcal{M}_T)$ is the robot's belief at deployment time, based on the evidence gathered from the states visited $\mathcal{M}_1, .., \mathcal{M}_T$.

2.4. Computing Belief over w^*

Belief Distribution. At iteration i, we can compute a belief distribution w^* based on past obervations $P_i(w =$

$w^*|\tilde{w}_i, \mathcal{M}_i)$ using Eq. (1) and Bayes Rule:

$$P_i(w = w^*|\tilde{w}_i, \mathcal{M}_i) \propto P(w)\prod_{k=1}^{|\mathcal{M}|}\frac{\exp\left[\beta R_w(\xi_{\tilde{w},M_k})\right]}{\tilde{Z}_k(w)},$$
$$\tilde{Z}_k(w) = \int_{\tilde{w}} \exp\left[\beta R_w(\xi_{\tilde{w},M_k})\right] d\tilde{w} \quad (3)$$

Algorithm 1 Particle Filter for Reward Inference

Require distribution $P_i(w = w^*)$, $\mathcal{M}_{i+1}$, $\tilde{w}_{i+1}$, number of particles N
Initialize particle sets $\bar{\chi}_{i+1} = \chi_{i+1} = \emptyset$
for $k = 1, 2, 3, 4, ..., N$ **do**
  Sample $\bar{w}_k \sim P_i(w = w^*)$
  $p_k = P_{\text{update}}(\tilde{w}|\bar{w}_k, \mathcal{M}_{i+1})$
  Add $(\bar{w}_k, p_k)$ to $\bar{\chi}_{i+1}$
end
for $k = 1, 2, 3, 4, ..., N$ **do**
  Draw k with probability $\sim p_k$ from $\bar{\chi}_{i+1}$
  Add $\bar{w}_k$ to χ_{i+1}
end
Return $P_{i+1}(w = w^*)$ as χ_{i+1}

where $P(w)$ is the prior and $\tilde{Z}$ is the normalizing constant. Computing the posterior distribution $P_i(w = w^*)$ gives us uncertainty over true reard w^*. We next introduce how the AI agent can reason about the influence of its action over this uncertainty via belief update.

Belief Update. When the AI agent takes an action by selecting M_{i+1} and receives a new observation $\tilde{w}_{i+1}$, inputs a new proxy reward $\tilde{w}_{\mathcal{M}_{i+1}}$, it can updates its belief distribution via Eq. (3). Because Eq. (3) is a computationally expensive step requires running MCMC sampling, we provide an alternative filtering algorithm for one-step belief update.

Given a model of how the designer chooses the next proxy $\tilde{w}_{i+1}$, we can compute new belief via the law of conditional probability.

$$P_{i+1}(w|\tilde{w}_{i+1}, \mathcal{M}_{i+1}) = \eta \cdot P_{\text{update}}(\tilde{w}_{i+1}|w, \mathcal{M}_{i+1})P_i(w = w^*) \quad (4)$$

where η is the normalizing constant, and P_{update} is the observation model defined in Eq. (3). Notice that the assumption of this observation model is that upon receiving M_{i+1}, the designer will go back to all previously received environemnts, and come up with a proxy $\tilde{w}^i$ that works on all of them. We call this joint update:

$$P^{\text{joint}}_{\text{update}}(\omega|\tilde{w}_{i+1}, \mathcal{M}_{i+1}) = P_{\text{design}}(\tilde{\omega}_{i+1}|w, \mathcal{M}_{i+1}) \quad (5)$$

Divide-and-conquer: one environment at a time. There exists an alternative design scheme (Ratner et al., 2018), where the designer specifies $\tilde{\omega}_{i+1}$ by only looking at M_{i+1}. The Divide and Conquer method has been shown to reduce the workload for designers when the number of environments becomes large. The Divide and Conquer method adopts a different update function:

$$P^{\text{indep}}_{\text{update}}(w|\tilde{w}_{i+1}, \mathcal{M}_{i+1}) = P_{\text{design}}(\tilde{\omega}_{i+1}|w, M_{i+1}) \quad (6)$$

We now introduce our filtering algorithm for one-step update in Algorithm 1, where we represent belief distribution $P_n(w = w^*)$ using a particle set. Our algorithm is based on Particle filter and is computationally efficient. Using this algorithm, we can compute the updated belief distribution $P_{i+1}(w = w^*|\tilde{w}_{i+1}, \mathcal{M}_{i+1})$ for different actions without re-running MCMC sampling.

2.5. Approximate Solution via Info-Gathering.

Maximal Information. Since the robot's objective depends on the distribution of deployment environments $\mathcal{M}_{deploy}$, which is unknown, our solution uses a proxy objective: disambiguate the reward as much as possible; ideally, if we identify w^* exactly, the robot attains 0 regret at deployment. Thus, we introduce an action strategy for the AI agent, by selecting actions that leads to the most information gain over its belief distribution. This is a a common used criteria in active learning (Houlsby et al., 2011; Gal et al., 2017) and robot active exploration (bur, 1997). Given an action M (MDP), we compute its mutual information with w:

$$f(M) = \mathbb{I}[w; M] = \mathbb{H}[w|(\mathcal{M}_i, w_i)] - \mathbb{E}_{\tilde{w}_{i+1}\sim P(w^*)}\mathbb{H}[w|(\mathcal{M}_{i+1}, \tilde{w}_{i+1})] \quad (7)$$

Here $\mathbb{H}[w]$ is the entropy of posterior distribution of $P(w = w^*)$ and $\mathbb{H}[w|(\mathcal{M}_{i+1}, \tilde{w}_{i+1})]$ is the conditional entropy after the new observation, which we can compute using Algorithm 1. Queries that tend to maximize this acquisition function are environments that have high probability of narrowing down our uncertainty over w^*. A mobile robot can select the best actions to localize itself in a building; an assisted reward design system should select the MDP that has the most potential to narrow down what the reward should be. For instance, the autonomous vehicle may be highly uncertain whether it should overtake the nearby car or not. In this case, we should inform the reward designer and ask for her input.

Algorithm. We hereby present our main algorithm for assisted reward design. At every iteration t, we use the information-based acquisition functions to select the next environment M_{t+1} to query reward designer. Because the AI's action space is vast containing all possible MDPs from $\mathcal{M}_{\text{devel}}$, we only sample a candidate set $\mathcal{M}_{\text{cand}}$ to select $\mathcal{M}_{t+1}$ from it.

Algorithm 2 Assisted Reward Design via Info-Gathering

Require prior $P_0(w)$, $\mathcal{M}_{\text{devel}}$, N_{cand}, initial training environments $\mathcal{M}_0$
Initialize posterior $\tilde{P}_0(w = w^*) = P_0(w)$
for $i = 0, 1, 2, 3, 4, ...$ **do**
  $\tilde{w}_i \sim P_{\text{user}}(w|w^*, \mathcal{M}_i)$ { Query the designer $\mathcal{M}_o$ }
  Compute posterior $\tilde{P}_{i+1}(w = w^*)$ using Algorithm 1
  Sample candidate environments from $\mathcal{M}_{\text{cand}} \subseteq \mathcal{M}_{\text{devel}}$, $|\mathcal{M}_{\text{cand}}| = N_{\text{cand}}$
  for $M \in \mathcal{M}_{\text{cand}}$ **do**
    Compute $f(M)$
  end
  Select $M_{i+1} = \arg\max_{M \in \mathcal{M}_{\text{cand}}} f(M)$
end

3. Experiments with Simulated Ground Truth

We evaluate Assisted Reward Design in a autonomous driving task. In this section, we assume that there exists a ground truth reward function. In Section 4, we evaluate our procedure in a case study where we need to capture subjective reward fuction that does not have an explicit form.

3.1. Task: Merging on Highway

Driving Environment We study merging behaviors on the highway illustrated in Fig. 2. The goal of the autonomous car is to merge to the left lane. Because there may be other human vehicles or obstacles on that lane, the autonomous car needs to decide whether to overtake or play it safe by slowing down. Real world autonomous vehicles need to handle situations like this on a daily basis, and a good reward function should correctly describe what we would like the autonomous car to do in all different situations.

There are 2 constant-speed human vehicles and 2 traffic cones placed randomly on the 3-lane highway. Each environment consists of the initial positions of human vehicles and traffic cones. The human vehicles drive forward at a constant speed.

Our cost function consists of 11 different features, including distance to other vehicles, distance to traffic cones, distance to left lane, distance to fences, control effort, etc. We design these features such that they can sufficiently capture all the nuances of different environments. Please refer to supplementary material for their detailed definitions. Note that real world systems with perception can often contain feature sets much larger than this. In those cases, it can be much more challenging for engineers to hand design robust reward functions.

Environment Distribution Successful merge tends to be more difficult when human vehicles and traffic cones are positioned closer to the autonomous vehicle. Thus we can use a simple metric to describe each environment:

$$\text{difficulty} = \sum_i \frac{1}{d_{\text{human vehicle }_i}} + \sum_j \frac{1}{d_{\text{traffic cone}_j}} \tag{8}$$

Using this metric, we visualize the distribution of $\mathcal{M}_{\text{devel}}$ and $\mathcal{M}_{\text{deploy}}$ in Fig. 2. Note that we design $\mathcal{M}_{\text{deploy}}$ to contain a higher density of difficult environments to measure the autonomous car's performance on challenging environments during deployment. Notice that both $\mathcal{M}_{\text{devel}}$ and $\mathcal{M}_{\text{deploy}}$ have a "long-tail" distribution of events. These events are rare to encounter on a daily basis, but could be highly useful for the reward design. An ideal acquisition function in assisted reward design should be able to identify these events.

Baselines We compare our information-based acquisition function with two baselines: (1) heuristic difficulty where we only sample $M \in \mathcal{M}_{\text{devel}}$ ranked highest by Eq. (8) and (2) random baseline where we select environments uniformly at random from the $\mathcal{M}_{\text{devel}}$. We also compare with directly using the specified rewards (the proxies $\tilde{w}$).

Evaluation A good assisted reward design system should help the reward designer quickly recover high-quality reward functions. To evaluate this, we specify one set of w^* as the ground truth reward, and perform assisted reward design. At every iteration, we simulate designer using w^* and Eq. (1). Please refer to the supplementary material for experiment hyperparameters.

We evaluate Assisted Reward Design method in two modes: *independent* mode (divide-and-conquer) where the designer specifies proxy reward for each environments ignoring the other environments, and *joint* mode where the designer must specify proxy reward that works across all environments so far.

Our result is visualized in Fig. 3. Our active method decreases the regret in both joint and independent modes, compared to random and compared to the rewards designed by hand (the proxies). We see in Fig. 2 that it generates environments that are more difficult. We also tested just using the environment difficulty as a proxy and directly sampling difficult environments (blue), and find that assisted reward design (orange) does outperform it, and without relying on the hand-crafted and domain-specific difficulty metric.

Edge-Case Nature of Proposed Environments To study why certain acquisition performs better than the other, we measure the quality of the proposed environments using:

$$r(M_{\text{next}}) = \frac{\mathbb{E}_{\tilde{w} \sim P(w=w^*)}\left[\text{Regret}(\tilde{w}; w^*, M_{\text{next}})\right]}{\mathbb{E}_{\tilde{w} \sim P(w=w^*)}\mathbb{E}_{\mathcal{M} \sim M_{\text{deploy}}}\left[\text{Regret}(\tilde{w}; w^*, M)\right]} \tag{9}$$

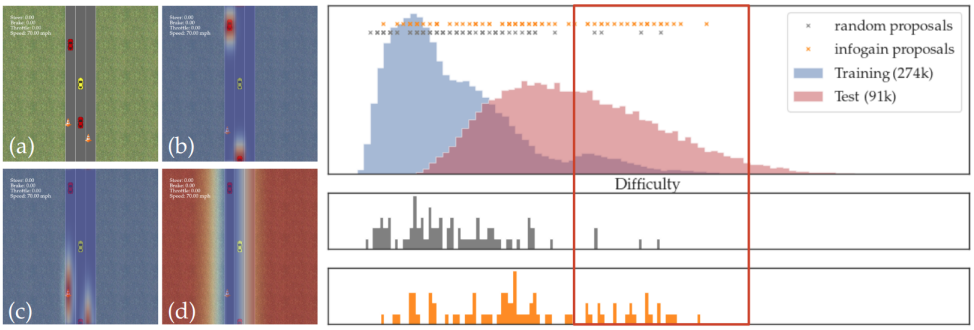

Figure 2: Left: (a) The goal of the autonomous vehicle (yellow) is to merge to the left lane while keeping safety with other human-driving vehicles (red). Environment features are highlighted in (b)(c)(d). Right: distribution of the environments based on difficulty metric. The distribution of $\mathcal{M}_{\text{devel}}$ is displayed in blue and $\mathcal{M}_{\text{deploy}}$ is displayed in pink. Note that test distribution noticeably contains a higher concentration of environments with dense interactions.

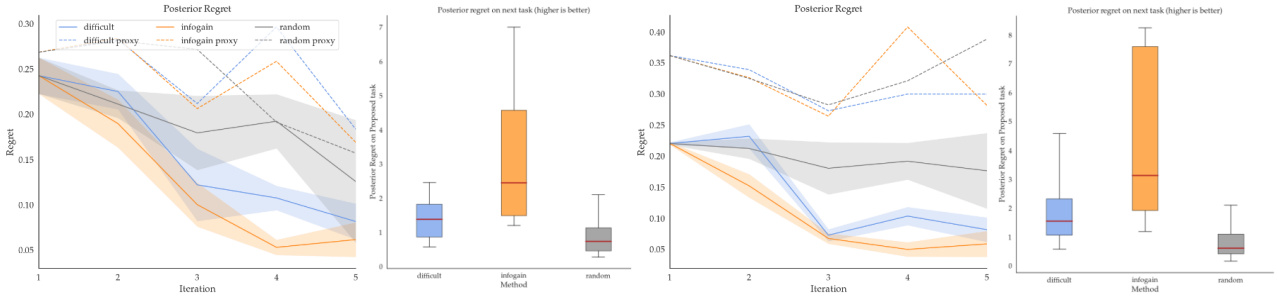

Figure 3: Experiment with simulated true reward. Left: joint mode, Right: independent design mode. We visualize the regret compared to true reward under different acquisition functions in (a)(c). We visualize the efficacy of the proposed environment in (b)(d). The "box" denotes 5th to 95th percentile

This computes the ratio of regret on next environment versus average regret on test environments. The higher $r(M_{\text{next}})$ is, the more the next environment uncovers the overall regret of the current posterior.

We visualize the values of proposed environments under different acquisitions in Fig. 2. Notice that Maximum Information proposes environments of highest value. This suggests that the to find the most useful environment for Assisted Reward Design, simply relying on heuristic environment ranking is not enough. It is much more effective to utilize all proxy rewards and their induced uncertainty over w^*.

4. Experiments on Real Reward Design

We evaluate our algorithm on the autonomous driving task where we use it to design a reward function that matches our internal desired criteria. Since we cannot explicitly write down the reward function, we cannot measure posterior regret as in Section 3. For evaluation purposes only, besides qualitative assessment, we introduce a violation criterion to measure the quality of our reward design, which can be thought of as unit tests on the behavior. Even though technically we could explicitly incorporate these tests as hard constraints in the "reward" function, we keep the reward features oblivious to these unit tests in order to demonstrate that assisted reward design can help us tune a reward function that fails fewer tests, as a measurable proxy for the quality of real-world behavior.

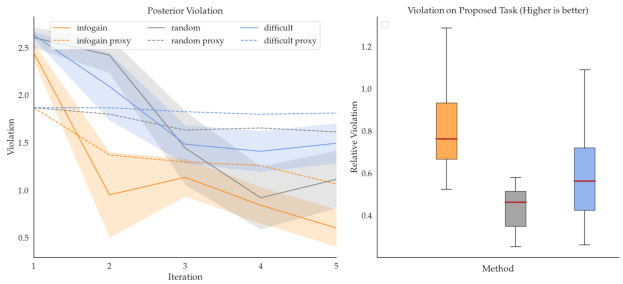

Figure 4: Experiment with real user. We visualize the violation compared to true violation under different acquisition functions on the left. On the right, we compare the relative value of proposed environments.

Violation We can define a set of hard constraints including (1) collision with other vehicles, (2) collision with traffic cones, (3) driving off track, (4) stopping, (5) driving over

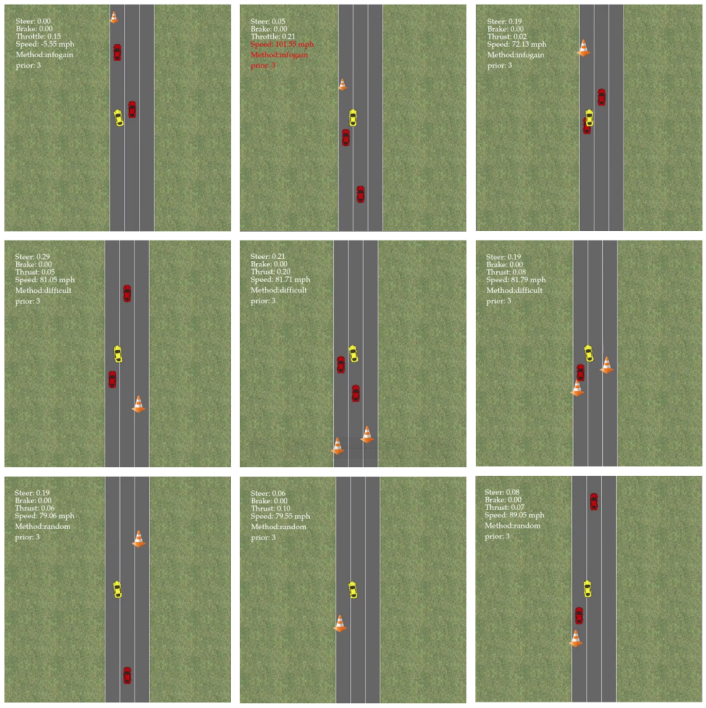

Figure 5: Visualization of the top 3 proposed environments based on each acquisition function.

speed, and (6) driving onto the right lane. The constraints capture the unwanted behaviors when the autonomous vehicle performs merging, and better reward functions lead to lower violation counts.

Evaluation We perform assisted reward design where we answer the queries stemming from the our algorithm, as well as the baselines. To remove our bias, we blind ourselves to which algorithm produced the query. We visualize the result in Fig. 4. Maximal Information outperforms the other acquisition functions in causing fewest average violations on $\mathcal{M}_{\text{deploy}}$.

Edge-Case Nature of Proposed Environments We measure the quality of the proposed environments at each iteration similar to Section 3 based on relative violation counts:

$$r(M_{\text{next}}) = \frac{\mathbb{E}_{\tilde{w}\sim P(w=w^*)}\left[\text{Violate}(\tilde{w}; M_{\text{next}})\right]}{\mathbb{E}_{\tilde{w}\sim P(w=w^*)}\mathbb{E}_{M\sim\mathcal{M}_{\text{deploy}}}\left[\text{Violate}(\tilde{w}; M)\right]} \tag{10}$$

Higher relative value means that the proposed environment uncovers more of the violations of the current posterior. We visualize the values of proposed environments under different acquisitions in Fig. 2. Again maximum information outperforms the heuristic difficulty metric in finding environments that contains more violations and leads to overall better reward design. We will next qualitatively study these edge cases.

Qualitative Analysis of Proposed Environments We random sample 3 starter environments and query designer on each of them independently. We then use maximum information, heuristic difficulty and random acquisition function to select the top 3 proposed environments as the next environment. We visualize the trajectory of MAP estimate w_{MAP} on each of these environments in Fig. 5. Notice that interestingly, Maximum Information tends to find environments where the current MAP estimate fails. Even though we do not explicitly optimize for finding failure cases, it turns out that these environments help reward designers effective narrow down on the true reward. In comparison, heuristic difficulty finds environments with dense interactions. Yet, they do not induce failure in the resulting trajectories. This is likely because such difficulties have been addressed by past reward designs. Since heuristic difficulty do not account for previous designs, it generates environments that are repetitive.

References

Active mobile robot localization. Citeseer, 1997.

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Bajcsy, A., Losey, D. P., O'Malley, M. K., and Dragan,

A. D. Learning robot objectives from physical human interaction. *Conference on Robot Learning (CoRL)*, 2017.

Bloem, M. and Bambos, N. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *53rd IEEE Conference on Decision and Control*, pp. 4911–4916. IEEE, 2014.

Brown, D. S., Cui, Y., and Niekum, S. Risk-aware active inverse reinforcement learning. *CoRR*, abs/1901.02161, 2019. URL http://arxiv.org/abs/1901.02161.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pp. 4299–4307, 2017.

Fu, J., Korattikara, A., Levine, S., and Guadarrama, S. From language to goals: Inverse reinforcement learning for vision-based instruction following. *arXiv preprint arXiv:1902.07742*, 2019.

Gal, Y., Islam, R., and Ghahramani, Z. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1183–1192. JMLR. org, 2017.

Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., and Dragan, A. Inverse reward design. In *Advances in neural information processing systems*, pp. 6765–6774, 2017.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Jain, A., Sharma, S., Joachims, T., and Saxena, A. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 34(10):1296–1313, 2015.

Levine, S. and Koltun, V. Continuous inverse optimal control with locally optimal examples. *arXiv preprint arXiv:1206.4617*, 2012.

Loftin, R. T., MacGlashan, J., Peng, B., Taylor, M. E., Littman, M. L., Huang, J., and Roberts, D. L. A strategy-aware technique for learning behaviors from discrete human feedback. In *AAAI Conference on Artificial Intelligence*, 2014.

Lopes, M., Melo, F., and Montesano, L. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 31–46. Springer, 2009.

Ng, A. Y. and Russell, S. J. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pp. 2, 2000.

Odom, P. and Natarajan, S. Active advice seeking for inverse reinforcement learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Ramachandran, D. and Amir, E. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pp. 2586–2591, 2007.

Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 729–736. ACM, 2006.

Ratner, E., Hadfield-Menell, D., and Dragan, A. D. Simplifying reward design through divide-and-conquer. *arXiv preprint arXiv:1806.02501*, 2018.

Reddy, S., Dragan, A. D., Levine, S., Legg, S., and Leike, J. Learning human objectives by evaluating hypothetical behavior, 2019.

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

Wirth, C., Akrour, R., Neumann, G., and Fürnkranz, J. A survey of preference-based reinforcement learning methods. *The Journal of Machine Learning Research*, 18(1): 4945–4990, 2017.

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