# Environment Design for Inverse Reinforcement Learning

**Thomas Kleine Buening** [1 *] **Victor Villin** [2 *] **Christos Dimitrakakis** [2]

## Abstract

Learning a reward function from demonstrations suffers from low sample-efficiency. Even with abundant data, current inverse reinforcement learning methods that focus on learning from a single environment can fail to handle slight changes in the environment dynamics. We tackle these challenges through adaptive environment design. In our framework, the learner repeatedly interacts with the expert, with the former selecting environments to identify the reward function as quickly as possible from the expert's demonstrations in said environments. This results in improvements in both sample-efficiency and robustness, as we show experimentally, for both exact and approximate inference.

## 1. Introduction

Reinforcement Learning (RL) is a powerful framework for autonomous decision-making in games (Mnih et al., 2015), continuous control problems (Lillicrap et al., 2015), and robotics (Levine et al., 2016). However, specifying suitable reward functions remains one of the main barriers to the wider application of RL in real-world settings and methods that allow us to communicate tasks without manually defining reward functions could be of great practical value. One such approach is Inverse Reinforcement Learning (IRL), which aims to find a reward function that explains observed (human) behaviour (Russell, 1998; Ng & Russell, 2000).

Much of recent effort in IRL has been devoted to making existing methods more sample-efficient as well as robust to changes in the environment dynamics (Arora & Doshi, 2021). Sample-efficiency is crucial, as data requires expensive human input. We also need robust estimates of the unknown reward function, so that the resulting policies remain near-optimal, even when the deployed environment dynamics differ from the ones we learned from.

However, recent work has found that IRL methods tend to heavily specialise ("overfit") to the specific transition dynamics under which the demonstrations were provided, thereby failing to generalise even across minor changes in the environment (Toyer et al., 2020). More generally, even with unlimited access to expert demonstrations, we may still fail to learn suitable reward functions from a fixed environment. In particular, prior work has explored the identifiability problem in IRL (Cao et al., 2021; Kim et al., 2021), illustrating the inherent limitations of IRL when learning from expert demonstrations in a single, fixed environment.

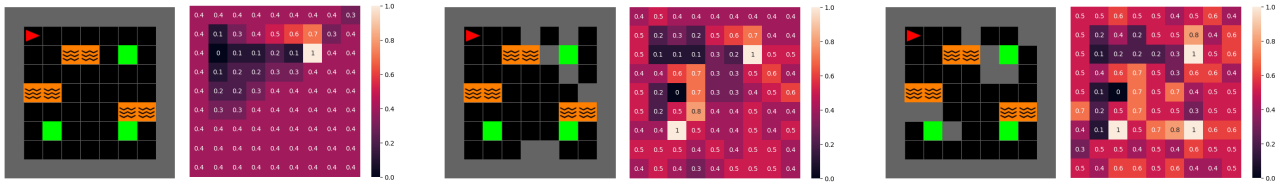
In our study, we consider the situation where we can *design* a sequence of *environments*, in which the expert will demonstrate the task. This can either mean slightly modifying a base environment, or selecting an environment from a finite set. Crafting new environments can involve simple adjustments such as relocating objects or adding obstacles, which can be done with little effort and cost. In open-world settings, simply having different task conditions (e.g., different cars, locations, or time-of-day in a vehicle scenario) amounts to a different environment.

**Contributions.** We propose algorithms for designing environments in order to infer human goals from demonstrations. This requires two key components: firstly, an *environment design* algorithm and secondly, an *inference* algorithm for data from multiple environment dynamics. Our hypothesis is that intelligent environment design can significantly improve both sample-efficiency of IRL methods and the robustness of learned rewards against variations in the environment dynamics. An example where our approach is applicable is given in Figure 1, where we need to learn the reward function (i.e., the location of the goal and lava squares). In summary, our contributions are:

1. An environment design framework that selects informative demo environments for the experts (Section 3).
2. An objective based on maximin Bayesian regret to choose environments in a way that compels the expert to provide useful information about the unknown reward function (Section 4).
3. An extension of Bayesian IRL and Maximum Entropy IRL to multiple environments (Section 5). We provide

[*]Equal contribution [1]The Alan Turing Institute, London, UK [2]Université de Neuchâtel, Neuchâtel, Switzerland. Correspondence to: Thomas Kleine Buening <tbuening@turing.ac.uk>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).



(a) 1st round (b) 2nd round (c) 3rd round

Figure 1: The expert navigates to the closest of the three possible goal squares while avoiding lava in adaptively elected maze environments. For three consecutive rounds (a)-(c), we display the mazes chosen by `ED-BIRL` (Algorithm 2 in Section 5) as well as the current reward estimate after observing an expert trajectory in the current and past mazes. By adaptively designing environments and combining the expert demonstrations, we can recover the locations of all goal and most lava squares. In contrast, from observations in a fixed environment, e.g., repeatedly observing the expert in maze (a), it would be impossible to recover all relevant aspects of the reward function, i.e., the location of the goal squares, as only the nearest goal square would be visited by the expert (repeatedly). Observing the human expert in new and carefully curated environments can lead to a more precise and robust estimate of the unknown reward function.

concrete implementations for the extensions of MCMC Bayesian IRL (Ramachandran & Amir, 2007) and Adversarial IRL (`AIRL`) (Fu et al., 2018).

4. We conduct extensive experiments to evaluate our approaches (Section 6). We test learned reward functions in unknown transition dynamics across various environments, including continuous mazes and MuJoCo benchmarks (Todorov et al., 2012). We compare against several other IRL and imitation learning algorithms, such as Robust Imitation learning with Multiple perturbed Environments (`RIME`) (Chae et al., 2022).

5. Our results illustrate the superior robustness of our algorithms and the effectiveness of the environment design framework. This shows that active environment selection significantly improves both the sample-efficiency of IRL and the robustness of learned rewards (generalisability).

## 2. Related Work

**Inverse Reinforcement Learning.** The goal of IRL (Russell, 1998; Ng & Russell, 2000) is to find a reward function that explains observed behaviour, which is assumed to be approximately optimal. Two of the most popular approaches to the IRL problem are Bayesian IRL (Ramachandran & Amir, 2007; Rothkopf & Dimitrakakis, 2011; Choi & Kim, 2011) and Maximum Entropy IRL (Ziebart et al., 2008; Ho & Ermon, 2016; Finn et al., 2016). In this work, we extend both IRL formulations to demonstrations under varying environment dynamics. Note that this differs from the situation, where we observe demonstrations by experts of varying quality (Castro et al., 2019), or demonstrations by experts that optimise different rewards (Ramponi et al., 2020; Likmeta et al., 2021), in a fixed environment. Moreover, Cao et al. (2021) and Rolland et al. (2022) study the identifiability of the true reward function in IRL and showed that when observing experts under different environment dynamics the true reward function can be identified up to a constant under certain conditions. However, it is important to note that in all of these cases, the learner is passive and does not actively seek information about the reward function by choosing specific experts or environments.

**Active Inverse Reinforcement Learning.** The environment design problem that we consider in this paper can be viewed as one of active reward elicitation (Lopes et al., 2009). Prior work on active reward learning has focused on querying the expert for additional demonstrations in specific states (Lopes et al., 2009; Brown et al., 2018; Lindner et al., 2021; 2022), mainly with the goal of resolving the uncertainty that is due to the expert's policy not being specified accurately in these states. In contrast, we consider the situation where we cannot directly query the expert for additional information in specific states, but instead sequentially choose environments for the expert to act in. Importantly, this means that the same state can be visited under different transition dynamics, which can be crucial to distinguish the true reward function among multiple plausible candidates (Cao et al., 2021; Rolland et al., 2022).

In other related work, Amin et al. (2017) consider a repeated IRL setting in which the learner can choose *any* task for the expert to complete (with full information of the expert policy). He & Dragan (2021) study an iterative reward design setup where a human provides the learner with a proxy reward function, upon which the learner tries to choose an edge-case environment in which the proxy fails so that the human revises their proxy. In a similar vein, Buening et al. (2022) introduced Interactive IRL, where the learner interacts with a human in a collaborative Stackelberg game without knowledge of the joint reward function. This setting is similar to the framework presented in this paper in that the leader in a Stackelberg game can be viewed as designing environments by committing to specific policies.

**Environment Design for Reinforcement Learning.** Environment design and curriculum learning for RL aim to design a sequence of environments with increasing difficulty to improve the training of an autonomous agent (Narvekar et al., 2020). However, in contrast to our problem setup, observations in generated training environments are cheap, since this only involves actions from an autonomous agent, not a human expert. As such, approaches like domain randomisation (Tobin et al., 2017; Akkaya et al., 2019) can be practical for RL, whereas they can be extremely inefficient and wasteful in an IRL setting. Moreover, in IRL we typically work with a handful of rounds only, so that slowly improving the environment generation process over thousands of training episodes (i.e., rounds) is impractical (Dennis et al., 2020; Gur et al., 2021). As a result, most methods, which are viable for the RL, can be expected to be unsuitable for the IRL problem.

## 3. Problem Formulation

We now formally introduce the Environment Design for Inverse Reinforcement Learning framework. A Markov Decision Process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, T, \boldsymbol{R}, \gamma, \omega)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function, $\boldsymbol{R} : \mathcal{S} \to \mathbb{R}$ is a reward function, $\gamma$ a discount factor, and $\omega$ an initial state distribution. We assume that there is a set transition functions $\mathcal{T}$ from which $T$ can be selected. Similar models have been considered for the RL problem under the name of Underspecified MDPs (Dennis et al., 2020) or Configurable MDPs (Metelli et al., 2018; Ramponi et al., 2021).

We assume that the true reward function, denoted $\boldsymbol{R}$, is unknown to the learner and consider the situation where the learner gets to interact with the human expert in a sequence of $m$ rounds.[1] More precisely, every round $k \in [m]$, the learner gets to select a demo environment $T_k \in \mathcal{T}$ for which an expert trajectory $\tau_k$ is observed. Our objective is to adaptively select a sequence of demo environments $T_1, \ldots, T_m$ so as to recover a robust estimate of the unknown reward function. We describe the general framework for this interaction between learner and human expert in Framework 1. To summarise, a problem-instance in our setting is given by $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \boldsymbol{R}, \gamma, \omega, m)$, where $\mathcal{T}$ is a set of environments, $\boldsymbol{R}$ is the *unknown* reward function, and $m$ the learner's budget.

From Framework 1 we see that the Environment Design for IRL problem has two main ingredients: a) choosing useful demo environments for the human to demonstrate the task in (Section 4), and b) inferring the reward function from expert demonstrations in multiple environments (Section 5).

[1] Typically, expert demonstrations are a limited resource as they involve expensive human input. We thus consider a limited budget of $m$ expert trajectories that the learner is able to obtain.

**Framework 1** Environment Design for IRL

**input** set of environments $\mathcal{T}$, resources $m \in \mathbb{N}$
**for** k = 1, . . . , m **do**
  Choose an environment $T_k \in \mathcal{T}$
  Observe expert trajectory $\tau_k$ in environment $T_k$
  Estimate rewards from observations up to round $k$

### 3.1. Preliminaries and Notation

Throughout the paper, $R$ denotes a generic reward function, whereas $\boldsymbol{R}$ refers to the true (unknown) reward function. We let $\Pi$ denote a generic policy space. Now, $\mathcal{V}^{\pi}_{R,T}(s) := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, T, s_0 = s]$ is the expected discounted return, i.e., value function, of a policy $\pi$ under some reward function $R$ and transition function $T$ in state $s$. For the value under the initial state distribution $\omega$, we then merely write $\mathcal{V}^{\pi}_{R,T} := \mathbb{E}_{s\sim\omega}[\mathcal{V}^{\pi}_{R,T}(s)]$ and denote its maximum by $\mathcal{V}^{*}_{R,T} := \max_{\pi} \mathcal{V}^{\pi}_{R,T}$. We accordingly refer to the $Q$-values under a policy $\pi$ by $Q^{\pi}_{R,T}(s, a)$ and their optimal values by $Q^{*}_{R,T}(s, a)$. In the following, we let $\pi^{*}_{R,T}$ always denote the *optimal policy* w.r.t. $R$ and $T$, i.e., the policy maximising the expected discounted return in the MDP $(\mathcal{S}, \mathcal{A}, T, R, \gamma, \omega)$.

In the following, we let $\tau$ denote expert trajectories. Note that in Framework 1 every such trajectory is generated w.r.t. some transition dynamics $T$. In round $k$, we thus observe $\mathcal{D}_k = (\tau_k, T_k)$, i.e., the expert trajectory $\tau_k$ in environment $T_k$. We then write $\mathcal{D}_{1:k} = (\mathcal{D}_1, \ldots, \mathcal{D}_k)$ for all observations up to (and including) the $k$-th round. Moreover, we let $\mathbb{P}(\cdot \mid \mathcal{D}_{1:k})$ denote the posterior over reward functions given observations $\mathcal{D}_{1:k}$. For the prior $\mathbb{P}(\cdot)$, we introduce the convention that $\mathbb{P}(\cdot) = \mathbb{P}(\cdot \mid \mathcal{D}_{1:0})$.

## 4. Environment Design via Maximin Regret

Our goal is to adaptively select demo environments for the expert based on our current belief about the reward function.

In Section 4.1, we introduce a maximin Bayesian regret objective for the environment design process which aims to select demo environments so as to ensure that our reward estimate is robust. Section 4.2 then deals with the selection of such environments when the set of environments exhibits a useful decomposable structure. We additionally provide a way to approximate the process when the set has an arbitrary structure or is challenging to construct.

### 4.1. Maximin Bayesian Regret

We begin by reflecting on the potential *loss* of an agent when deploying a policy $\pi$ under transition function $T$ and the *true* reward function $\boldsymbol{R}$, given by the difference

$$\ell_{\boldsymbol{R}}(T, \pi) := \mathcal{V}^{*}_{\boldsymbol{R},T} - \mathcal{V}^{\pi}_{\boldsymbol{R},T}.$$

The reward function $\boldsymbol{R}$ is unknown to us so that we can instead use our belief $\mathbb{P}$ over reward functions[2] and consider the *Bayesian regret* (i.e., loss) of a policy $\pi$ under $T$ and $\mathbb{P}$ given by

$$\mathrm{BR}_{\mathbb{P}}(T, \pi) \coloneqq \mathbb{E}_{R\sim\mathbb{P}}\big[\ell_R(T, \pi)\big].$$

The concept of Bayesian regret is well-known from, e.g., online optimisation and online learning (Russo & Van Roy, 2014) and has been utilised for IRL in a slightly different form by Brown et al. (2018). The idea is that given a (prior) belief about some parameter, we evaluate our policy against an oracle that knows the true parameter. Typically, under such uncertainty about the true parameter (in our case, reward function) we are interested in policies minimising the Bayesian regret:

$$\min_{\pi\in\Pi} \mathrm{BR}_{\mathbb{P}}(T, \pi).$$

To derive an objective for the environment design problem, we then consider the maximin problem given by the worst-case environment $T$ for our current belief over reward functions $\mathbb{P}$: [3]

$$\max_{T\in\mathcal{T}} \min_{\pi\in\Pi} \mathrm{BR}_{\mathbb{P}}(T, \pi). \tag{1}$$

What this means is that we search for an environment $T \in \mathcal{T}$ such that the regret-minimising policy w.r.t. $\mathbb{P}$ performs the worst compared to the optimal policies w.r.t. the reward candidates $R \sim \mathbb{P}$. In other words, the maximin environment $T$ from (1) can be viewed as the environment in which we expect our current reward estimate to perform the worst.

Choosing environments for the expert according to (1) also has the advantage that maximin environments are in general solvable for the expert, since the regret in degenerate or purely adversarial environments will be close to zero. Moreover, the regret objective is performance-based and not only uncertainty-based, such as entropy-based objectives (Lopes et al., 2009)). This is typically desired as reducing our uncertainty about the rewards in states that are not relevant under any transition function in $\mathcal{T}$ (e.g., states that are not being visited by any optimal policy) is unnecessary and generally a wasteful use of our budget. Finally, we also see that if the Bayesian regret objective becomes zero, the posterior mean is guaranteed to be optimal in every demo environment.

**Lemma 4.1.** *If for some posterior $\mathbb{P}(\cdot \mid \mathcal{D})$ we have $\max_{T\in\mathcal{T}} \min_{\pi\in\Pi} \mathrm{BR}_{\mathbb{P}}(T, \pi) = 0$, then the posterior mean $\bar{R} = \mathbb{E}_{\mathbb{P}}[R]$ is optimal for every $T \in \mathcal{T}$, i.e., $\bar{R}$ induces an optimal policy in every environment contained in $\mathcal{T}$.*

[2] When we do not have a posterior over rewards, it is still possible to build a pseudo-belief upon point estimates. This approach is later explained in Section 5.2.

[3] We consider $\max_T \min_\pi$ and not the reverse, as we are interested in the maximin environment (and not minimax policy).

It is worth noting that our maximin Bayesian regret objective resembles several approaches to robust reinforcement learning, e.g., (Roy et al., 2017; Zhou et al., 2021; Buening et al., 2023; Zhou et al., 2024). However, it differs in that we are interested in the maximin environment (not minimax policy) and the Bayesian regret is defined w.r.t. a set of environments $\mathcal{T}$ *and* a distribution over reward functions.

### 4.2. Finding Maximin Environments

**Structured Environments.** Often the set of environments has a useful structure that can be exploited to search the space of environments efficiently. We here consider the special case where each environment $T \in \mathcal{T}$ is build from a collection of transition matrices $T_s$. Similar setups can be found in the robust dynamic programming literature (e.g., (Iyengar, 2005; Nilim & El Ghaoui, 2005; Xu & Mannor, 2010; Mannor et al., 2016)).

Let $T_s \in \mathbb{R}^{\mathcal{S}\times A}$ denote a state-transition matrix dictating the transition probabilities in state $s$. We can identify any transition function $T$ with a family of state-transition matrices $\{T_s\}_{s\in\mathcal{S}}$. We then say that an environment set $\mathcal{T}$ allows us to make *state-individual transition choices* if there exist sets $\mathcal{T}_s$ such that $\mathcal{T} = \{\{T_s\}_{s\in\mathcal{S}} \colon T_s \in \mathcal{T}_s\}$. In other words, we can choose a new environment $T$ by arbitrarily combining transition matrices for each state. Note that this of course allows for the case where the transitions in some state $s$ are fixed, i.e., $\mathcal{T}_s = \{T_s\}$. When we can make such state-individual transition choices, we can use an extended value iteration approach to approximate the maximin environment efficiently. The extended value iteration algorithm is specified in Appendix B, Algorithm 4.

**Arbitrary Environments.** In some situations, the set of environments $\mathcal{T}$ may not exhibit any useful structure. Moreover, we may not even have explicit knowledge of the transition functions in $\mathcal{T}$, but can only access a set of corresponding simulators. In this case, we are left with approximating the maximin environment (1) by sampling simulators from $\mathcal{T}$ and performing policy rollouts. We describe the complete procedure in Appendix B, Algorithm 5.

**Flexible Environment Set Construction.** Although our assumption initially considers the simplest scenario where any environment within $\mathcal{T}$ is accessible at any time, this may become impractical when the process of building environments is labour-intensive. Instead of probing the entire set $\mathcal{T}$ at each environment design step, our framework allows some flexibility. With little approximation in Framework 1 Line 3, we can select the next environment from a new subset $\mathcal{T}_\subset \subset \mathcal{T}$. This allows users to progressively build new environments with every additionally desired round.

**Algorithm 2** ED-BIRL: Environment Design for BIRL

**input** environments $\mathcal{T}$, prior $\mathbb{P}$, budget $m \in \mathbb{N}$
**for** $k = 1, \dots, m$ **do**
  Sample rewards from $\mathbb{P}(\cdot \mid \mathcal{D}_{1:k-1})$ using BIRL
  Construct empirical distribution $\hat{\mathbb{P}}_{k-1}$ from samples
  Find $T_k \in \arg\max_T \min_\pi \mathrm{BR}_{\hat{\mathbb{P}}_{k-1}}(T, \pi)$
  Observe trajectory $\tau_k$ in $T_k$, i.e., $\mathcal{D}_k = (\tau_k, T_k)$
**return** $\mathrm{BIRL}(\mathcal{D}_{1:m})$

**Algorithm 3** ED-AIRL: Environment Design for AIRL

**input** environments $\mathcal{T}$, budget $m \in \mathbb{N}$
**for** $k = 1, \dots, m$ **do**
  **if** $k = 1$ **then**
    Choose $T_k$ arbitrarily from $\mathcal{T}$
  **else**
    Let $\hat{\mathbb{P}}_{k-1} \equiv \mathcal{U}(\{R_1, \dots, R_{k-1}\})$
    Find $T_k \in \arg\max_T \mathrm{BR}_{\hat{\mathbb{P}}_{k-1}}(T, \pi^*_{R_{1:k-1},T})$
  Observe trajectory $\tau_k$ in $T$, i.e., $\mathcal{D}_k = (\tau_k, T_k)$
  Compute point estimate $R_k = \mathrm{AIRL}(\mathcal{D}_k)$
  Compute best guess $R_{1:k} = \mathrm{AIRL\text{-}ME}(\mathcal{D}_{1:k})$
**return** $\mathrm{AIRL\text{-}ME}(\mathcal{D}_{1:m})$

## 5. Inverse Reinforcement Learning with Multiple Environments

We now explain how to learn about the reward function from demonstrations that were provided under multiple environment dynamics. To do so, we will extend Bayesian and MaxEnt IRL methods to this setting, and combine them with environment design to obtain the ED-BIRL and ED-AIRL algorithm, respectively. While ED-BIRL is designed for simple tabular problems due to its high complexity, ED-AIRL can be applied to environments with large or continuous action/observation spaces.

### 5.1. The Bayesian Setting: ED-BIRL

The Bayesian perspective to the IRL problem provides a principled way to reason about reward uncertainty (Ramachandran & Amir, 2007). Typically, the human is modelled by a Boltzmann-rational policy (Jeon et al., 2020). This means that for a given reward function $R$ and transition function $T$ the expert is acting according to a softmax policy $\pi^{\text{softmax}}_{R,T}(a \mid s) = \frac{\exp(cQ^*_{R,T}(s,a))}{\sum_{a'} \exp(cQ^*_{R,T}(s,a'))}$, where the parameter $c$ relates to our judgement of the expert's optimality.[4] Given a prior distribution $\mathbb{P}(\cdot)$, the goal of Bayesian IRL is to recover the posterior distribution $\mathbb{P}(\cdot \mid \mathcal{D})$ and to either sample from the posterior using MCMC (Ramachandran & Amir, 2007; Rothkopf & Dimitrakakis, 2011) or perform MAP estimation (Choi & Kim, 2011).

In our case, the data is given by the sequence $\mathcal{D}_{1:k} = (\mathcal{D}_1, \dots, \mathcal{D}_k)$ with $\mathcal{D}_i = (\tau_i, T_i)$. We see that this is no obstacle as the likelihood factorises as $\mathbb{P}(\mathcal{D}_{1:k} \mid R) = \prod_{i \le k} \mathbb{P}(\tau_i \mid R, T_i)$, since the expert trajectories (i.e., expert policies) are conditionally independent given the reward function and transition function. The likelihood of each expert demonstration is then given by $\mathbb{P}(\tau_i \mid R, T_i) = \prod_{(s,a)\in\tau_i} \pi^{\text{softmax}}_{R,T_i}(a \mid s)$. Hence, the reward posterior can be expressed as

$$\mathbb{P}(R \mid \mathcal{D}_{1:k}) \propto \prod_{i \le k} \prod_{(s,a)\in\tau_i} \pi^{\text{softmax}}_{R,T_i}(a \mid s) \cdot \mathbb{P}(R). \quad (2)$$

[4] Note that when using MCMC Bayesian IRL methods we can also perform inference over the parameter $c$ and must not assume knowledge of the expert's optimality.

As a result, we can, for instance, sample from the posterior using the Policy-Walk algorithm from (Ramachandran & Amir, 2007) with minor modifications or the Metropolis-Hastings Simplex-Walk algorithm from (Buening et al., 2022). We generally denote any Bayesian IRL algorithm that is capable of sampling from the posterior by BIRL.

Plugging BIRL into our environment design framework, we get the ED-BIRL procedure detailed in Algorithm 2. Note that, in practice, we approximate the posterior $\mathbb{P}(\cdot \mid \mathcal{D}_{1:k})$ by sampling rewards and constructing an empirical distribution $\hat{\mathbb{P}}_k$.

### 5.2. The MaxEnt Setting: AIRL-ME and ED-AIRL

In the following, we describe how to extend MaxEnt IRL methods to demonstrations from multiple environments, and use them in combination with environment design for IRL.

**Reward Inference with Multiple Environments.** In MaxEnt IRL, the reward function is assumed to be parameterised by some parameter $\theta$. Given observations $\mathcal{D}_{1:k} = (\mathcal{D}_1, \dots, \mathcal{D}_k)$ with $\mathcal{D}_i = (\tau_i, T_i)$, our goal for multiple environments is to solve the maximum likelihood problem

$$\arg\max_\theta \sum_{(\tau,T)\in\mathcal{D}_{1:k}} \log \mathbb{P}(\tau \mid \theta, T), \quad (3)$$

where we again used that trajectories are independent conditional on the reward parameter $\theta$ and the dynamics $T$. Consequently, the only difference to the original MaxEnt IRL formulation is that we now sum over pairs $(\tau, T)$ instead of just trajectories $\tau$. The specific algorithm we consider here is Adversarial IRL (AIRL) (Fu et al., 2018), which frames the optimisation of (3) as a generative adversarial network.

To extend AIRL to multiple environments, we can consider $k$ distinct policies $\pi_1, \dots, \pi_k$ used to generate trajectories in environments $T_1, \dots, T_k$, respectively, and discriminators

$D_{1,\theta,\phi_1}, \ldots, D_{k,\theta,\phi_k}$ given by

$$D_{i,\theta,\phi_i}(s,a,s') = \frac{\exp(f_{\theta,\phi_i}(s,s'))}{\exp(f_{\theta,\phi_i}(s,s')) + \pi_i(a \mid s)},$$

with $f_{\theta,\phi_i}(s,s') = g_\theta(s) + \gamma h_{\phi_i}(s') - h_{\phi_i}(s)$. Here, $g_\theta$ is the state-only reward approximator, and $h_{\phi_i}$ a shaping term specific to each environment $T_i$. We refer to this algorithm as AIRL−ME and defer for a detailed description to Algorithm 6 in Appendix B.

**Environment Design for AIRL: ED−AIRL.** This algorithm also relies on Bayesian regret, even though we do not have an analytical posterior as in the Bayesian setting. We substitute the posterior with a uniform distribution $\mathcal{U}(\{R_1, \ldots, R_k\})$ over point estimates $R_i$, each separately calculated with standard AIRL from demonstration $\mathcal{D}_i$. We then also replace the posterior mean $\bar{R}$ with the multi-environment estimate obtained from AIRL−ME$(\mathcal{D}_{1:k})$, denoted $R_{1:k}$.[5] Here, $R_{1:k}$ represents our best guess about the reward function after round $k$. In particular, for a given environment $T$, the optimal policy $\pi^*_{R_{1:k},T}$ w.r.t. $R_{1:k}$ and $T$ is the policy we expect to perform best in environment $T$. In round $k+1$, we then choose the environment $T$ in which the reward function $R_{1:k}$ performs the worst compared to the reward functions $R_1, \ldots, R_k$. This environment is given by $\arg\max_{T \in \mathcal{T}} \mathrm{BR}_{\mathcal{U}(\{R_1,\ldots,R_k\})}(T, \pi^*_{R_{1:k},T})$. The ED−AIRL procedure is detailed in Algorithm 3.

*Remark* 5.1. Note that, in every round $k$, we can of course let ED−BIRL and ED−AIRL query a batch of several demonstrations from the environment $T_k$. In the case of ED−AIRL, our experiments showed that this imporves the stability of the reward inference.

## 6. Experiments

The primary goal of our experiments is to address the following two questions: (1) Can we *recover the true reward function* by adaptively designing environments? (2) Are the reward estimates *more robust* by doing so?

To answer the first question, Section 6.1 considers the maze task already introduced in Figure 1 of the introduction, as it allows us to nicely visualise the learned reward functions. To answer the second question, we evaluate our environment design approach on continuous control tasks in Section 6.2, where we slightly perturb environment dynamics to test the robustness of learned rewards. We also include an ablation study by replacing our maximin Bayesian regret environment design with domain randomisation. This allows us to separate the effect of active environment design from the effect of learning a reward function from multiple (possibly randomly chosen) environments. Additional experimental results and details are provided in Appendix C and Appendix D.

[5]Note that $R_{1:k}$ is generally not equal to the mean of $\mathcal{U}(\{R_1, \ldots, R_k\})$.

### 6.1. Recovering the True Reward Function

In this experiment, we consider a discrete maze task inspired by minigrid (Chevalier-Boisvert et al., 2023), in which the learner has the ability to add obstacles to a base layout of the maze. We visualise the selected mazes and estimated rewards, and evaluate whether our approach can recover the true reward function.

**Experimental Setup.** In the maze task, the objective is to reach one of three goal squares while avoiding lava. Here, any maze from the set of environments is obtained by adding, removing, or moving gray obstacles. Note that the goal and lava squares cannot be moved. The learner gets to select mazes from this set and observes two expert trajectories for every chosen maze.[6] The true reward function, which is unknown to the learner, yields reward 1 in goal squares and reward $-1$ in lava squares. We discount the problem by $\gamma = 0.9$, so that the expert tries to reach a goal square as quickly as possible.

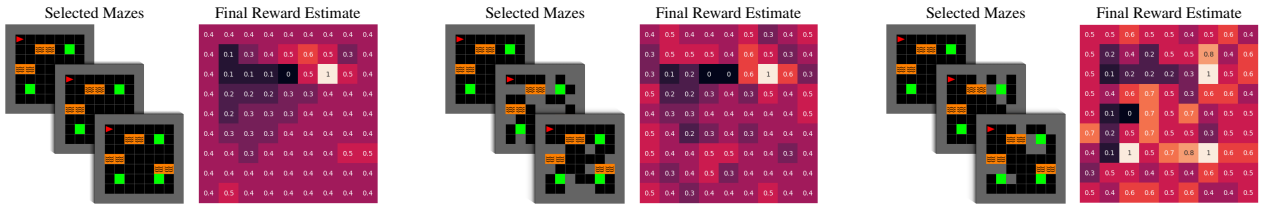
We compare our approach, ED−BIRL, with learning from a single, fixed maze as well as learning from mazes that were randomly generated. We randomly generate these mazes by adding an obstacle to a tile with probability $0.3$.[7] The inference for all three approaches is done using BIRL and the computed reward estimates are scaled to $[0, 1]$ and rounded to one decimal.

**Results.** In Figure 2, we observe that ED−BIRL recovers the location of all three goal squares after three rounds. Moreover, the learner is able to identify the location of all lava strips in Figure 2c, i.e., squares with negative reward. Hence, by adaptively designing a sequence of environments, ED−BIRL is capable of recovering all performance-relevant aspects of the unknown reward function.

In contrast, learning from a fixed environment (Figure 2a) as well as domain randomisation (Figure 2b) both fail to recover the location of all goal squares, let alone lava squares. In a fixed maze, any near-optimal policy will visit the closest goal state only, which in this case is in the top right corner of the maze. We also see that using domain randomisation is impractical for IRL, as we require carefully curated mazes to recover the true reward function. Even worse, by obliviously randomising the maze layout, we may create

[6]We give the learner two trajectories per chosen environment to provide a stronger learning signal to BIRL.

[7]Naturally, such randomly generated mazes can be very different every iteration, and we can only display exemplary mazes for domain randomisation in Figure 2. Nevertheless, the presented examples shall serve as an illustration of the disadvantages of using domain randomisation in IRL, where demonstrations are costly.




(a) Fixed Environment BIRL (b) Domain Randomisation BIRL (c) ED-BIRL

Figure 2: The discrete maze task from Figure 1. The goal in the discrete maze environment is to reach one of the three green goal squares while avoiding lava squares. For each approach, we show on the left the chosen mazes and visualise on the right the posterior mean reward after three rounds. In (a), the expert always acts in the same, fixed maze. In (b), the maze is randomly generated by adding obstacles, i.e., gray squares, uniformly at random. The proposed ED-BIRL approach, which adaptively chooses maze layouts based on past reward estimates, is shown in (c). We use the same colour scale as in Figure 1, which ranges from black (0.0) to red (0.5) to white (1.0).

unsolvable environments for the human expert, which yield no information at all (see Figure 2b). Additional results on the tabular maze task can be found in Appendix C.

### 6.2. Learning Robust Reward Functions in Continuous Control Problems

In this set of experiments, we evaluate ED-AIRL on continuous control tasks. We consider a continuous maze task (Fu et al., 2018) as well as MuJoCo environments (Todorov et al., 2012). For these test suites, we no longer have direct access to the transition function but can only access their corresponding simulators. We here compare ED-AIRL with Domain Randomisation AIRL (DR-AIRL) and standard AIRL (Fu et al., 2018). Moreover, we also benchmark against imitations learning approaches, including Behavioral Cloning (BC), Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016) and RIME (Chae et al., 2022). Notably, RIME is an imitation learning algorithm that tries to find a robust policy based on demonstrations from multiple environments.

**Experimental Setup.** We treat the continuous maze problem without additional walls as well as the original MuJoCo environments in their default settings as *base environments*. For each task, we then build a *disjoint* set of *demo environments* (i.e., environments in which we could potentially observe the expert) and *test environments* by altering their respective base environments.

In the continuous maze, this is done by adding walls to the maze. For the MuJoCo environments, we tweak the environment dynamics by changing parameters of the physics engine, which include body features of the agent as well as gravity parameters (see Appendix D for details). In total, we create 20 demo environments and 10 test environments for each task, where we want to emphasize that these sets are disjoint so that we never observe the expert in any test environments. We include the base environment by default in the set of demo environments. Exemplary demo and test environments are illustrated in Figure 3.

For the approaches that learn from multiple environments (ED-AIRL, DR-AIRL, RIME), the learner has access to the set of demo environments out of which it can sequentially choose 5 (10 for the continuous maze) demo environments for the expert to act in. The learner always chooses the base environment first, serving as the initialisation for ED-AIRL and ensuring a fair comparison. Subsequently, DR-AIRL chooses environments from the demo set uniformly at random without replacement, and then uses AIRL-ME to compute a reward estimate from all observed trajectories. As RIME does not estimate the reward function, we cannot plug-in our environment design approach. Instead, we select environments from the demo set analogously to DR-AIRL.

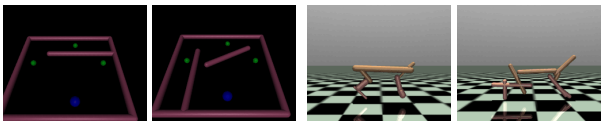
In total, every approach observes a total of $m = 50$ expert trajectories. Specifically, ED-AIRL, DR-AIRL, and RIME observe 10 trajectories (5 for the continuous maze) in each environment they select. In contrast AIRL, GAIL, and BC observe 50 trajectories in the base environment.

For the IRL approaches (ED-AIRL, DR-AIRL, AIRL), we evaluate the final reward estimates after observing all 50 expert trajectories. We do this by optimising a fresh policy in each of the demo and test environments using the learned reward function. We then evaluate this policy under the true reward function in each environment before reporting the average return across the demo and test sets. For the imitation learning algorithms (RIME, GAIL, BC) we report the average return the imitating policy achieves when deployed on the demo and test set environments.

**Results.** In Table 1, we see that ED-AIRL overall achieves the highest performance on the demo and test sets. In most cases, ED-AIRL outperforms DR-AIRL, sometimes significantly (Swimmer), with the exception of

| | Continuous maze | | Hopper | | HalfCheetah | | Swimmer | |
|---|---|---|---|---|---|---|---|---|
| | Demo | Test | Demo | Test | Demo | Test | Demo | Test |
| ED-AIRL | **68±04** | **71±02** | **63±07** | 52±04 | **40±11** | 35±13 | 80±19 | 69±12 |
| DR-AIRL | 52±07 | 53±12 | 59±06 | **56±04** | **40±13** | **40±11** | 45±04 | 53±05 |
| AIRL | 33±09 | 52±07 | 38±03 | 34±04 | 29±09 | 16±07 | 40±09 | 44±08 |
| RIME | -105±12 | -52±03 | 61±01 | 53±02 | -21±08 | -11±09 | -05±01 | -04±01 |
| GAIL | 20±05 | 17±01 | 40±02 | 34±01 | -12±02 | -06±01 | 111±00 | 110±01 |
| BC | 11±00 | 22±00 | -12±02 | -06±01 | -23±01 | -14±01 | **124±01** | **130±01** |

Table 1: Performance on the continuous control tasks. Average normalised returns and their standard error over the demo and test sets (averaged over 10 runs for MuJoCo and 5 runs for the continuous maze). 100 indicates expert performance, whereas 0 indicates the performance of a random policy.



Demo Test Demo Test

(a) Continuous maze (b) HalfCheetah

Figure 3: Examples of demo and test environments.

HalfCheetah, where both approaches perform similarly. Notably, ED-AIRL consistently outperforms AIRL by a large margin in all test suites.

As expected, the imitating policies from GAIL and BC fail to transfer across variations of the base environment (Table 1). An exemption is the Swimmer environment, in which the imitating policies of GAIL and BC were extremely robust to any changes in the environment dynamics (gravity and body features). Lastly, in our experimental setup, RIME struggles to learn a reasonable policy with the exception of Hopper where RIME performs similarly to ED-AIRL and DR-AIRL. The reason for its bad performance could be that in our case the environments in the demo and test set are much more different from one another compared to the study done in Chae et al. (2022).

In Figure 4, we also analyse the improvement of ED-AIRL versus AIRL as we increase the total number of expert trajectories in the continuous maze task. Whereas AIRL does not significantly improve with more trajectories, we observe that ED-AIRL shows steady improvement until plateauing at around 65% expert level performance when given a budget of 25 trajectories. More generally, even though AIRL is designed to learn a robust reward function, it appears that learning from a single environment is insufficient to generalise to slight variations of a base environments, independently of the number of trajectories.

We provide several additional experimental results for

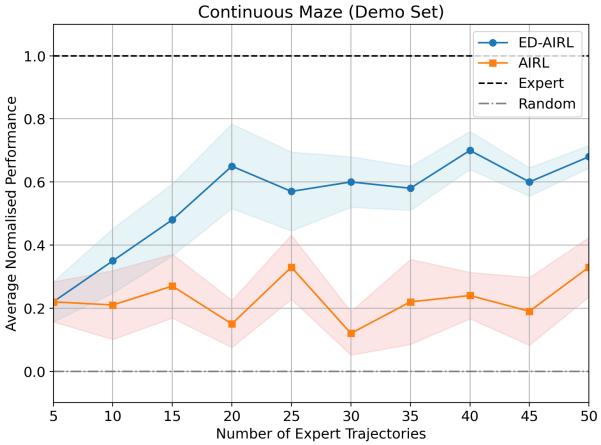



Figure 4: Average normalised performance on the demo set in the continuous maze as we increase the number of expert trajectories (averaged over 5 runs). The standard error is shown in shaded colour. Every 5 trajectories, ED-AIRL chooses a new environment for the expert to act in. In contrast, AIRL always observes the expert act in the same base environment.

ED-BIRL and ED-AIRL in Appendix C, including insights into the environments that are being selected by ED-AIRL in the continuous maze task.

## 7. Conclusion

We introduced Environment Design for Inverse Reinforcement Learning, which concerns how to optimally design a sequence of environments to acquire expert demonstrations. We proposed a general methodology for doing so, based on a maximin Bayesian regret objective. This idea can be directly combined with Bayesian IRL methods to obtain the ED-BIRL algorithm. We also propose an approximate method, ED-AIRL, based on an ensemble of point estimates from the AIRL procedure. Experimentally, we show that our

approach is able to recover almost all performance-relevant aspects of the unknown reward function, and improves the robustness of learned reward functions.

**Limitations.** The computational load of `ED-AIRL` can become heavy since the number of policy optimisations grows quadratically with the number of rounds. We can reduce the computational load by sampling a subset $\mathcal{T}_{\subset}$ of environments for the environment design process. In this case, `ED-AIRL` requires $\mathcal{O}(|\mathcal{T}_{\subset}|m^2)$ policy optimisations, where $m$ is the number of times we select an environment.

**Future directions.** In our experiments, we found the inference of `AIRL`, as well as other MaxEnt IRL methods, to be unstable. In future work, we could use a Gaussian process model for the reward function (Levine et al., 2011). This can be made scalable, for instance, through a variational inference approach (Chan & van der Schaar, 2021).

## Acknowledgements

Thomas Kleine Buening acknowledges the support of the Norwegian Research Council (Grant 302203) and the UKRI Prosperity Partnership Scheme (FAIR).

## Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

## References

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Amin, K., Jiang, N., and Singh, S. Repeated inverse reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Arora, S. and Doshi, P. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

Brown, D. S., Cui, Y., and Niekum, S. Risk-aware active inverse reinforcement learning. In *Conference on Robot Learning*, pp. 362–372. PMLR, 2018.

Buening, T. K., George, A.-M., and Dimitrakakis, C. Interactive inverse reinforcement learning for cooperative games. In *International Conference on Machine Learning*, pp. 2393–2413. PMLR, 2022.

Buening, T. K., Dimitrakakis, C., Eriksson, H., Grover, D., and Jorge, E. Minimax-bayes reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 7511–7527. PMLR, 2023.

Cao, H., Cohen, S., and Szpruch, L. Identifiability in inverse reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12362–12373, 2021.

Castro, P. S., Li, S., and Zhang, D. Inverse reinforcement learning with multiple ranked experts. *arXiv preprint arXiv:1907.13411*, 2019.

Chae, J., Han, S., Jung, W., Cho, M., Choi, S., and Sung, Y. Robust imitation learning against variations in environment dynamics. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2828–2852. PMLR, 17–23 Jul 2022.

Chan, A. J. and van der Schaar, M. Scalable Bayesian inverse reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=4qR3coiNaIv.

Chevalier-Boisvert, M., Dai, B., Towers, M., de Lazcano, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S., and Terry, J. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Choi, J. and Kim, K.-e. Map inference for bayesian inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/3a15c7d0bbe60300a39f76f8a5ba6896-Paper.pdf.

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., and Levine, S. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in Neural Information Processing Systems*, 33: 13049–13061, 2020.

Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pp. 49–58. PMLR, 2016.

Fu, J., Luo, K., and Levine, S. Learning robust rewards with adversarial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkHywl-A-.

Gleave, A., Taufeeque, M., Rocamonde, J., Jenner, E., Wang, S. H., Toyer, S., Ernestus, M., Belrose, N., Emmons, S., and Russell, S. imitation: Clean imitation learning implementations. *arXiv preprint arXiv:2211.11972*, 2022.

Gur, I., Jaques, N., Miao, Y., Choi, J., Tiwari, M., Lee, H., and Faust, A. Environment generation for zero-shot compositional reinforcement learning. *Advances in Neural Information Processing Systems*, 34:4157–4169, 2021.

He, J. Z.-Y. and Dragan, A. D. Assisted robust reward design. *arXiv preprint arXiv:2111.09884*, 2021.

Ho, J. and Ermon, S. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Iyengar, G. N. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

Jeon, H. J., Milli, S., and Dragan, A. Reward-rational (implicit) choice: A unifying formalism for reward learning. *Advances in Neural Information Processing Systems*, 33: 4415–4426, 2020.

Kim, K., Garg, S., Shiragur, K., and Ermon, S. Reward identification in inverse reinforcement learning. In *International Conference on Machine Learning*, pp. 5496–5505. PMLR, 2021.

Levine, S., Popovic, Z., and Koltun, V. Nonlinear inverse reinforcement learning with gaussian processes. *Advances in neural information processing systems*, 24, 2011.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Likmeta, A., Metelli, A. M., Ramponi, G., Tirinzoni, A., Giuliani, M., and Restelli, M. Dealing with multiple experts and non-stationarity in inverse reinforcement learning: an application to real-life problems. *Machine Learning*, 110:2541–2576, 2021.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lindner, D., Turchetta, M., Tschiatschek, S., Ciosek, K., and Krause, A. Information directed reward learning for reinforcement learning. *Advances in Neural Information Processing Systems*, 34:3850–3862, 2021.

Lindner, D., Krause, A., and Ramponi, G. Active exploration for inverse reinforcement learning. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 5843–5853. Curran Associates, Inc., 2022.

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.

Mannor, S., Mebel, O., and Xu, H. Robust mdps with k-rectangular uncertainty. *Mathematics of Operations Research*, 41(4):1484–1509, 2016. ISSN 0364765X, 15265471.

Metelli, A. M., Mutti, M., and Restelli, M. Configurable markov decision processes. In *International Conference on Machine Learning*, pp. 3491–3500. PMLR, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*, 2020.

Ng, A. Y. and Russell, S. J. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 2, 2000.

Nilim, A. and El Ghaoui, L. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

Ramachandran, D. and Amir, E. Bayesian inverse reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pp. 2586–2591, 2007.

Ramponi, G., Likmeta, A., Metelli, A. M., Tirinzoni, A., and Restelli, M. Truly batch model-free inverse reinforcement learning about multiple intentions. In *International conference on artificial intelligence and statistics*, pp. 2359–2369. PMLR, 2020.

Ramponi, G., Metelli, A. M., Concetti, A., and Restelli, M. Learning in non-cooperative configurable markov decision processes. *Advances in Neural Information Processing Systems*, 34, 2021.

Rolland, P., Viano, L., Schürhoff, N., Nikolov, B., and Cevher, V. Identifiability and generalizability from multiple experts in inverse reinforcement learning. In Koyejo,

S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 550–564. Curran Associates, Inc., 2022.

Rothkopf, C. A. and Dimitrakakis, C. Preference elicitation and inverse reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 34–48, 2011.

Roy, A., Xu, H., and Pokutta, S. Reinforcement learning under model mismatch. *Advances in neural information processing systems*, 30, 2017.

Russell, S. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on computational learning theory*, pp. 101–103, 1998.

Russo, D. and Van Roy, B. Learning to optimize via information-directed sampling. *Advances in Neural Information Processing Systems*, 27, 2014.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Toyer, S., Shah, R., Critch, A., and Russell, S. The magical benchmark for robust imitation. *Advances in Neural Information Processing Systems*, 33:18284–18295, 2020.

Xu, H. and Mannor, S. Distributionally robust markov decision processes. *Advances in Neural Information Processing Systems*, 23, 2010.

Zhou, R., Liu, T., Cheng, M., Kalathil, D., Kumar, P., and Tian, C. Natural actor-critic for robust reinforcement learning with function approximation. *Advances in neural information processing systems*, 36, 2024.

Zhou, Z., Zhou, Z., Bai, Q., Qiu, L., Blanchet, J., and Glynn, P. Finite-sample regret bound for distributionally robust offline tabular reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3331–3339. PMLR, 2021.

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

# Appendix

## A. Proofs

*Proof of Lemma 4.1.* For simplicity of exposition, we assume here that the posterior $\mathbb{P}$ is discrete. Now, as the value function is linear in rewards, we have

$$\min_{\pi} \mathrm{BR}_{\mathbb{P}}(T, \pi) = \mathrm{BR}_{\mathbb{P}}(T, \pi_{\bar{R},T}),$$

where $\pi_{\bar{R},T}$ is the optimal policy w.r.t. the posterior mean $\bar{R} = \mathbb{E}_{R\sim\mathbb{P}}[R]$ and the transition function $T$. If $\max_{T\in\mathcal{T}} \min_{\pi\in\Pi} \mathrm{BR}_{\mathbb{P}}(T, \pi) = 0$, it then follows that $\max_{T\in\mathcal{T}} \mathrm{BR}_{\mathbb{P}}(T, \pi_{\bar{R},T}) = 0$, i.e. $\mathcal{V}^*_{R,T} = \mathcal{V}^{\pi_{\bar{R},T}}_{R,T}$ for all $R \in \mathrm{supp}(\mathbb{P})$ and $T \in \mathcal{T}$. This must imply that $\mathcal{V}^*_{\boldsymbol{R},T} = \mathcal{V}^{\pi_{\bar{R},T}}_{\boldsymbol{R},T}$ for all $T \in \mathcal{T}$. In other words, $\bar{R}$ is optimal for all $T \in \mathcal{T}$ (under the initial state distribution $\omega$). □

## B. Algorithms

### B.1. Finding Maximin Environments

**Structured Environments.** In the structured environment setting, we can independently choose a transition distribution for each state-action pair. Extended Value Iteration for Structured Environments, provided in Algorithm 4, estimates the maximin environment by iteratively making state-individual transition choices. At each stage, and for each state in the state space, we determine the state-transition matrix that maximises the regret for this given state. At the end, we return the environment corresponding to the transition matrices selected in the final stage.

**Arbitrary Environments.** The algorithm, Environment Design with Arbitrary Environments, is provided in Algorithm 5. It determines an estimate of the maximin environment by computing another approximation of the Bayesian regret. This approximation is done by running full policy optimisation steps in order to recover the policy values for each environment-reward combination. To obtain the Bayesian regret of an environment, we take the sum of the reward losses for each point estimate rewards: the difference between the value of a policy trained with a point estimate reward function and the value of a policy trained with the best guess. We return the environment with the highest Bayesian regret. To alleviate the computation burden brought by this method, it is possible to sample a subset of environments for selection, rather than the full set of possible environments.

**`ED-AIRL` with Arbitrary Environments.** Note that in the special situation running `ED-AIRL`, when estimating the minimax environment at the second expert-learner round (Algorithm 3 line 7 when $k = 2$) we end up with two possibly different estimations of the same reward function : $R_{1:1}$ and $R_1$. In this case, it makes sense to consider the absolute reward loss when running environment design (Algorithm 5 line 8) as it let us discard environments that exhibit no information at all, while only having a single point estimate. Additionally, point estimates learned by `AIRL` must all be normalised with their minimum and maximum when computing the reward losses.

### B.2. MaxEnt IRL with Multiple Environments

`AIRL-ME`, given in Algorithm 6, is the extension of `AIRL` to multiple environments. To learn a common reward to $k$ chosen environments, we maintain $k$ generating policies, where policy $i$ only interacts with environment $i$, and $k$ discriminators, where each discriminator $i$ has its own shaping parameters $\phi_i$ but has shared reward approximator parameters $\theta$. Expert data for all $k$ considered environments are supposedly already collected. After generating trajectories with each policy, we train for a few steps discriminators $1, \ldots, k$ to classify expert and generated data from environments $1, \ldots, k$. Each policy $i$ is then trained for one policy optimisation step with their respective reward function $R_{i,\theta,\phi_i}$. We repeat this procedure for a fitting number of iterations and then extract the learned common reward function. Remark that `AIRL` has to be run in "state-only" mode, meaning that we learn a reward function that only depends on the current state. Since action meanings change with the transition dynamics, it does not make sense to include an action dependence to a reward that has to be common to multiple transition functions (Fu et al., 2018).

**Algorithm 4** Extended Value Iteration for Structured Environments

**input** environments $\mathcal{T} = \{\mathcal{T}_s\}_{s\in\mathcal{S}}$, empirical distribution $\hat{\mathbb{P}}$, best guess $\bar{R}$
**repeat** until $\mathcal{V}_{\bar{R}}$ and $\mathcal{V}_R$ converge:
**for** $s \in \mathcal{S}$ **do**
  $T_s = \arg\max_{T_s\in\mathcal{T}_s} \left\{ \mathbb{E}_{R\sim\hat{\mathbb{P}}}\big[\max_{a\in\mathcal{A}} T_{s,a}^\top \mathcal{V}_R\big] - \max_{b\in\mathcal{A}} T_{s,b}^\top \mathcal{V}_{\bar{R}} \right\}$
  $\mathcal{V}_R(s) = \max_{a\in\mathcal{A}} R(s) + \gamma T_{s,a}^\top \mathcal{V}_R$ for every $R \sim \hat{\mathbb{P}}$
  $\mathcal{V}_{\bar{R}}(s) = \max_{b\in\mathcal{A}} \bar{R}(s) + \gamma T_{s,b}^\top \mathcal{V}_{\bar{R}}$
**return** environment $T = \{T_s\}_{s\in\mathcal{S}}$

**Algorithm 5** Environment Design with Arbitrary Environments

**input** set of environments $\mathcal{T}$, point estimates $\{R_1, \dots, R_k\}$, best guess $\bar{R}$
// if necessary, sample a subset $\mathcal{T}_\subset$ from $\mathcal{T}$
**for** $T \in \mathcal{T}$ **do**
  calculate $\bar{\pi} \in \arg\max_\pi \mathcal{V}^\pi_{\bar{R},T}$ (policy optimisation)
  **for** $R \in \{R_1, \dots, R_k\}$ **do**
    evaluate $\mathcal{V}^{\bar{\pi}}_{R,T}$ (policy evaluation)
    calculate $\mathcal{V}^*_{R,T} = \max_\pi \mathcal{V}^\pi_{R,T}$ (policy optimisation)
    $\ell(R) = \mathcal{V}^*_{R,T} - \mathcal{V}^{\bar{\pi}}_{R,T}$
  $\mathrm{BR}(T) = \sum_{R\in\{R_1,\dots,R_k\}} \ell(R)$
**return** $T^* = \arg\max_{T\in\mathcal{T}} \mathrm{BR}(T)$

**Algorithm 6** `AIRL-ME` (`AIRL` with Multiple Environments)

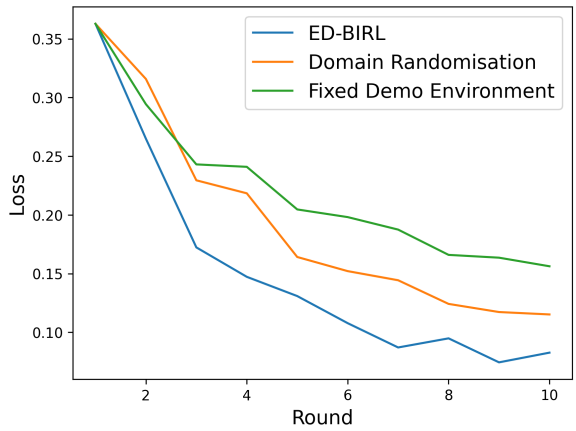
**input** Observations $\mathcal{D} = (\mathcal{D}_1, \dots, \mathcal{D}_k)$ with $\mathcal{D}_i = (\tau_i, T_i)$
Initialise policies $\pi_1, \dots, \pi_k$ and discriminators $D_{1,\theta,\phi_1}, \dots, D_{k,\theta,\phi_k}$
**for** $t = 0, 1, \dots$ **do**
  Collect trajectories $\tau^G_{i,j} = (s_0, a_0, \dots, s_H, a_H)$ by executing $\pi_i$ in $T_i$ for $i \in [k]$.
  Train discriminators $D_{1,\theta,\phi_1} \dots, D_{k,\theta,\phi_k}$ to classify expert data $\tau_1, \dots, \tau_k$ from samples $\{\tau^G_{1,j}\}_j, \dots, \{\tau^G_{k,j}\}_j$, respectively, via logistic regression with shared parameter $\theta$.
  Update reward $R_{i,\theta,\phi_i}(s,a,s') \leftarrow \log D_{i,\theta,\phi_i}(s,a,s') - \log(1 - D_{i,\theta,\phi_i}(s,a,s'))$ for $i \in [k]$.
  Update $\pi_1, \dots, \pi_k$ with respect to $R_{1,\theta,\phi_1}, \dots, R_{k,\theta,\phi_k}$ using any policy optimisation method.
**return** $g_\theta$ (extract the reward approximator)

## C. Additional Experimental Results

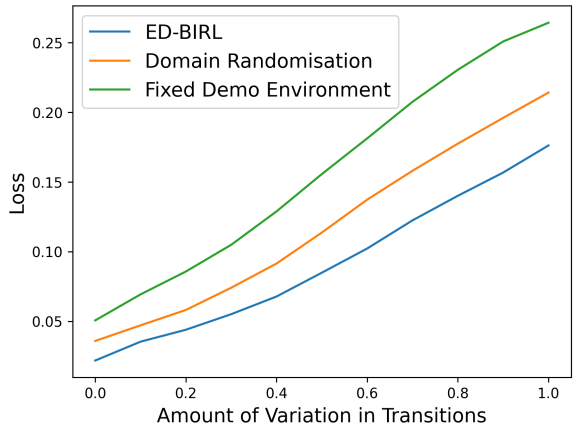
### C.1. Learning Robust Reward Functions on Randomly Generated MDPs

The learner is provided with a set of demo environments they can select for a demonstration. Afterwards, the agent is evaluated on a set of test environments. The performance on the test set captures the generalisation ability of the learned rewards to new dynamics.

**Experimental Setup.** We first randomly generate a base MDP $(\mathcal{S}, \mathcal{A}, T_{\texttt{base}}, \boldsymbol{R}, \gamma, \omega)$ with base transition function $T_{\texttt{base}}$. We then construct the set of possible demo environments, here denoted $\mathcal{T}_{\texttt{demo}}$ instead of $\mathcal{T}$ to clearly distinguish between demo and test environments, by sampling state-transition functions that differ from the base transitions $T_{\texttt{base}}$ by at most some value $\rho_{\texttt{demo}}$ in terms of $\ell_\infty$-distance. In our experiments, we set the maximum amount of variation in the demo environments to $\rho_{\texttt{demo}} = 0.5$. Similarly, we create a set of test environments $\mathcal{T}_{\texttt{test}}$ with a maximum amount of perturbation $\rho_{\texttt{test}}$ on which we evaluate the learned reward functions. For all three approaches, we evaluate the posterior mean, which is computed using `BIRL`. For all $T \in \mathcal{T}_{\texttt{test}}$, we optimise a policy w.r.t. the posterior mean and $T$ and evaluate the computed policy under the true reward function $\boldsymbol{R}$ and transition function $T$. Finally, we average the results over all environments in $\mathcal{T}_{\texttt{test}}$. We want to emphasise that the way we construct $\mathcal{T}_{\texttt{demo}}$ and $\mathcal{T}_{\texttt{test}}$, these sets are completely disjunct except for the




(a) Average utility *loss* of `ED-BIRL`, `Domain Randomisation`, and `Fixed Environment IRL` over 10 rounds. The learned rewards are evaluated on a set of test environments that differ from the base environment by at most $\rho_{\texttt{test}} = 0.5$.




(b) Along the $x$-axis we increase $\rho_{\texttt{test}}$, i.e. the amount of variation in the test environments. We evaluate the learned reward functions after 10 rounds of interaction with the expert, i.e. the final reward estimate from Figure 5a.

Figure 5: On a randomly generated MDP task, we evaluate the robustness of reward estimates learned by `ED-BIRL`, `Domain Randomisation`, and `Fixed Environment IRL`, respectively.

base transition function, i.e. $\mathcal{T}_{\texttt{demo}} \cap \mathcal{T}_{\texttt{test}} = \{T_{\texttt{base}}\}$. We therefore *do not* observe the expert in the environments that we evaluate our approaches on.

**Results.** In Figure 5a, we observe that `ED-BIRL` outperforms domain randomisation and learning from a fixed environments over the course of all rounds. As expected, the loss of all three approaches increases the more diverse the test environments are and the more they differ from the base environment, which can be seen in Figure 5b. Interestingly, even for $\rho_{\texttt{test}} = 0$, i.e. evaluation on the base environment only, `ED-BIRL` slightly outperforms learning directly from the fixed base environment suggesting a superior sample-efficiency of `ED-BIRL`.

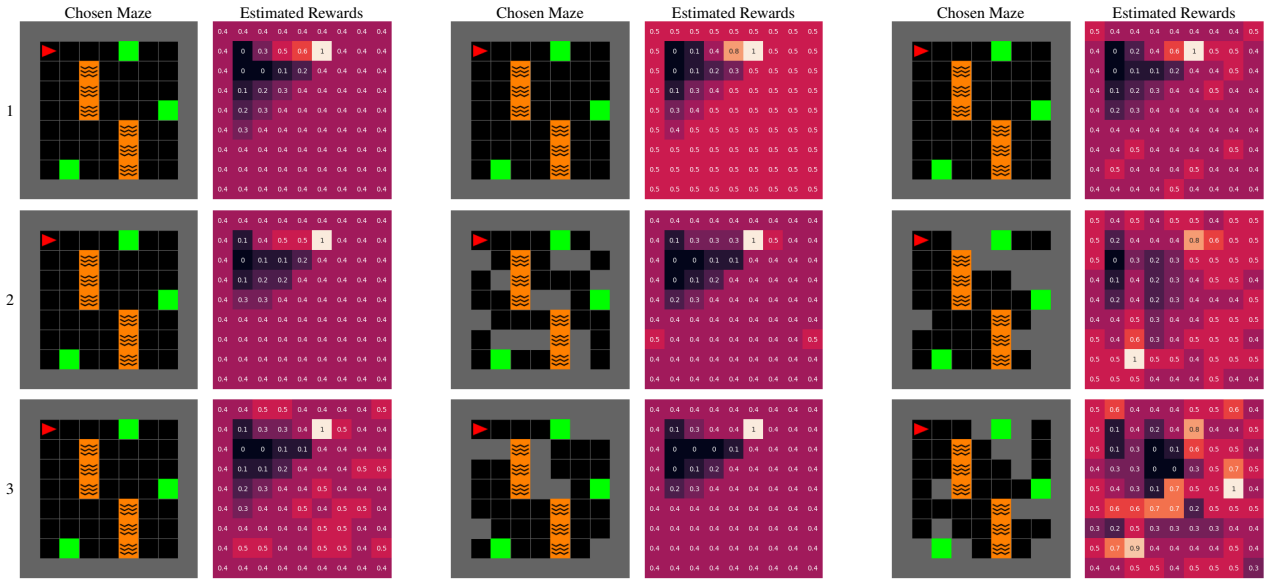
### C.2. Recovering the True Reward Function

We provide an additional visualisation of the learned rewards on another maze problem considered in the experiments from Section 6.1 in Figure 9. Results on this version of the maze comforts our past observations. `ED-BIRL`, Figure 6c, recovers the location of all three goal states and some lava states, after three round. We see that each round, the environment generated by `ED-BIRL` forces the expert to visit a goal state which is yet unknown to the reward function. In contrast, `Fixed Environment IRL` and `Domain Randomisation`, Figures 6a,6b, only retrieve the location of one goal state. Even when increasing the number of rounds (see Figure 10), `Domain Randomisation` does not manage to recover all three goals, pointing out clearly the lower sample efficiency of the method. For the initial maze and the randomly generated mazes, the (optimal) expert trajectory always corresponds to reaching the top right goal state. For this reason, those random environment designs never lead to discovering additional performance-relevant aspects of the reward function.

### C.3. Learning Robust Reward Functions for Continuous Control

With Figure 7, we provide a more interpretable visualisation of how `ED-AIRL` selects environments. After observing the expert on the base environment, `DR-AIRL` selects environments at uniform, while `ED-AIRL` performs a curated selection. Figure 7b shows how often each environment is selected by each approach, on the continuous maze task. We observe that on the 5 times we ran `ED-AIRL`, it consistently avoided environments similar to the base environment (12) and uninformative environments (15 and 20). This is a direct improvement from `DR-AIRL`. We also provide a visualisation of the rewards learned by `ED-AIRL` and `DR-AIRL` on the continuous maze.

We provide more detailed results in Table 3. Experiments on the MuJoCo Ant environment are also included. Based on the quantiles on both continuous maze and Swimmer, `ED-AIRL` reliably finds a reward function that solves at least 25% of the




(a) `Fixed Environment IRL` (b) `Domain Randomisation` (c) `ED-BIRL`

Figure 6: Comparison of `ED-BIRL`, `Fixed Environment IRL`, and `Domain Randomisation` on the first three rounds of another version of the maze problem seen in Section 6.1.

mazes on both sets, which is not ensured by `AIRL`. More importantly, this manifests that it is in general safer to learn from a reward function estimated by `ED-AIRL` as its 25%-quantile is consistently higher.

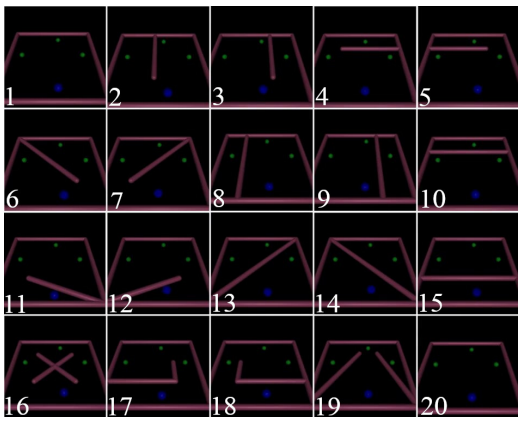
## D. Additional Experimental Details

**Recovering the True Reward Function.** For the experiments in Section 6.1, we let the learner observe two trajectories for each maze. This was done in order to speed up the inference of `BIRL` and reduce the computational cost. The expert was modeled by a Boltzmann-rational policy and thus uniformly selected an optimal action when there were several optimal ones in a given state.

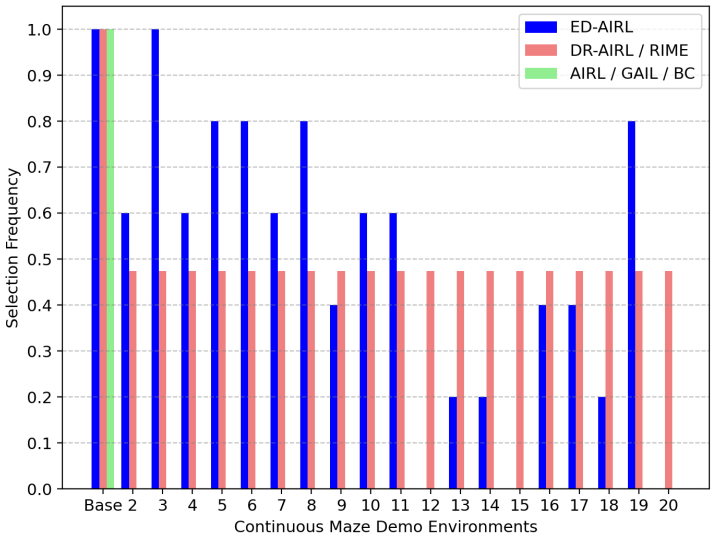
**Learning Robust Reward Functions for Continuous Control.** For the experiments in Section 6.2, We allow each approach to learn with unlimited interaction with their environments, but evaluate their learned rewards by optimising policies for a fixed amount of environment steps. This is justified by the fact that we want to compare the best reward estimation offered by each approach, only constrained by the amount of expert data $m$. Besides, because increasing the amount of observable environment on `ED-AIRL` quadratically rises the amount of policy optimisation required, we limited the selection size to $n = 10$ for the continuous maze task and $n = 5$ for the remaining tasks. Each time we select a new environment in `ED-AIRL` with Algorithm 5, we sample a subset $\mathcal{T}_{\subset}$ of size 10 and 5, for the continuous maze task and other tasks respectively. For all `AIRL`-based algorithms, we used a two-layer ReLU network with 32 units for the state-only reward approximator and shaping functions. We also smoothed the estimated rewards outputted by `AIRL` and `AIRL-ME` by taking the average of the estimated reward functions from the last 10 discriminator-generator rounds.

We chose to limit ourselves to the default MuJoCo environment settings as much as possible. This meant excluding robot positions from observations, except in Swimmer where it was deemed necessary. We utilised the implementations from (Gleave et al., 2022) for `GAIL` and `BC`, initially tuned with the inclusion of the position in the observation, explaining some variations in our results. Regarding `RIME`, we attempted to tune the original implementation from (Chae et al., 2022) and used their weight-shared discriminator approach. Lastly, `AIRL` is known to not perform on Ant. (Fu et al., 2018) and (Gleave et al., 2022) both considered modified versions of the environments, we in contrast kept the default settings in order to observe whether multi-environment approaches could improve the results or not.

All of our policies were optimised with Proximal Policy Optimisation (Schulman et al., 2017). Within the scope of each task, demo and test environment experts were trained with identical hyperparameters and for an equal amount of timesteps.




(a) Continuous maze demo environments.




(b) Frequency at which each continuous maze environment was picked for expert demonstrations.

Figure 7: For the continuous maze task, the learner can observe experts in $n = 10$ different environments. `AIRL`, `GAIL` and `BC` only observe the expert on the base environment, while `ED-AIRL` and `DR-AIRL` can observe the expert on any environments in the demo set. For a fair comparison, we force multi-environment approaches to always include the base environment in their selection. Environment labeled 20 is designed to be uninformative: the agent is disabled and can not move around, an expert policy is thus indistinguishable from any other policy under this setting.

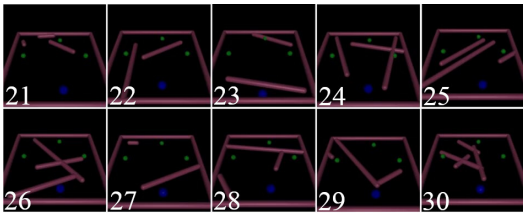



Figure 8: Continuous maze test environments.

For better hindsight, average values of random and expert policies on base environments are given in Table 2.

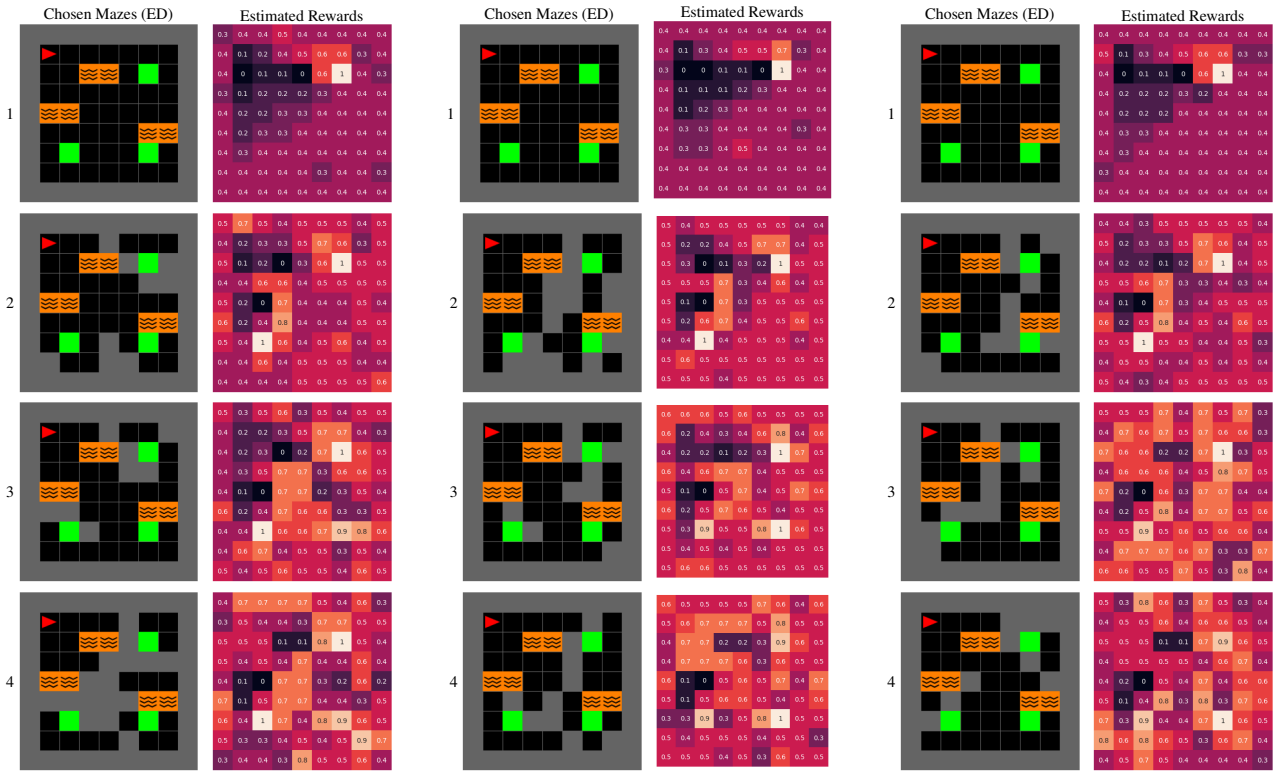
**Continuous maze.** The task consists in reaching all accessible green targets in whichever order. The agent observes its current position and whether each goal was already reached or not. While rewarded with 1 point for collecting a target, the agent is incurred with an action cost to motivate optimal path making. The environment is considered solved when all reachable targets were collected before the episode of fixed length ends. We provide a visual of the continuous maze demo and test set in Figure 7a and Figure 8.

**Learning Robust Reward Functions on Randomly Generated MDPs.** We randomly generated an MDP with 50 states and 4 actions using a Dirichlet distribution for the transitions and a Beta distribution for the reward function. For each state we let the demo set of environments contain 15 choices. The size of the test environments was set to be $|\mathcal{T}_{\texttt{test}}| = 500$. Every round, the learner got to select a demo environment and observe a single expert trajectory in that environment. We limited the amount of deviation from the base transitions in our experiments according to $\rho_{\texttt{demo}}$ and $\rho_{\texttt{test}}$. In particular, note that any choice of $\rho_{\texttt{demo}}$ implies that $\|T_{\texttt{base}} - T\|_\infty = \max_{s,a} \|T_{\texttt{base}}(\cdot \mid s,a) - T(\cdot \mid s,a)\|_1 \leq \rho_{\texttt{demo}}$ for all $T \in \mathcal{T}_{\texttt{demo}}$. The results were averaged over 5 complete runs, i.e. for 5 randomly generated problem instances.

**Implementation.** The code used for all of our experiments is available at github.com/Ojig/Environment-Design-for-IRL.

| | Continous Maze | Swimmer | HalfCheetah | Hopper | Ant |
|---|---|---|---|---|---|
| Random | 1.14±0.80 | 9±12 | -246±76 | 16±16 | -34±23 |
| Expert | 2.83±0.01 | 174±9 | 3728±447 | 2723±275 | 2331±725 |

Table 2: Average scores and their standard deviation, for random and expert policies on the base environments. Experts for base, demo and test environments for a given task were trained with the same hyperparameters for the same number of iterations.




(a) Run 1. (b) Run 2. (c) Run 3.

Figure 9: Selected mazes and reward estimates of `ED-AIRL` over the course of 4 environment selections (two trajectories per environment), for 3 separate runs.

**Compute.** Three AMD EPYC 7302P machines were used. Most of the computation time was spent running ED-AIRL, which requires a large amount of full policy optimisation steps. With 5 expert-learner rounds and depending on the MuJoCo environment ED-AIRL requires from half to a full day of wall-clock computation time on one machine.

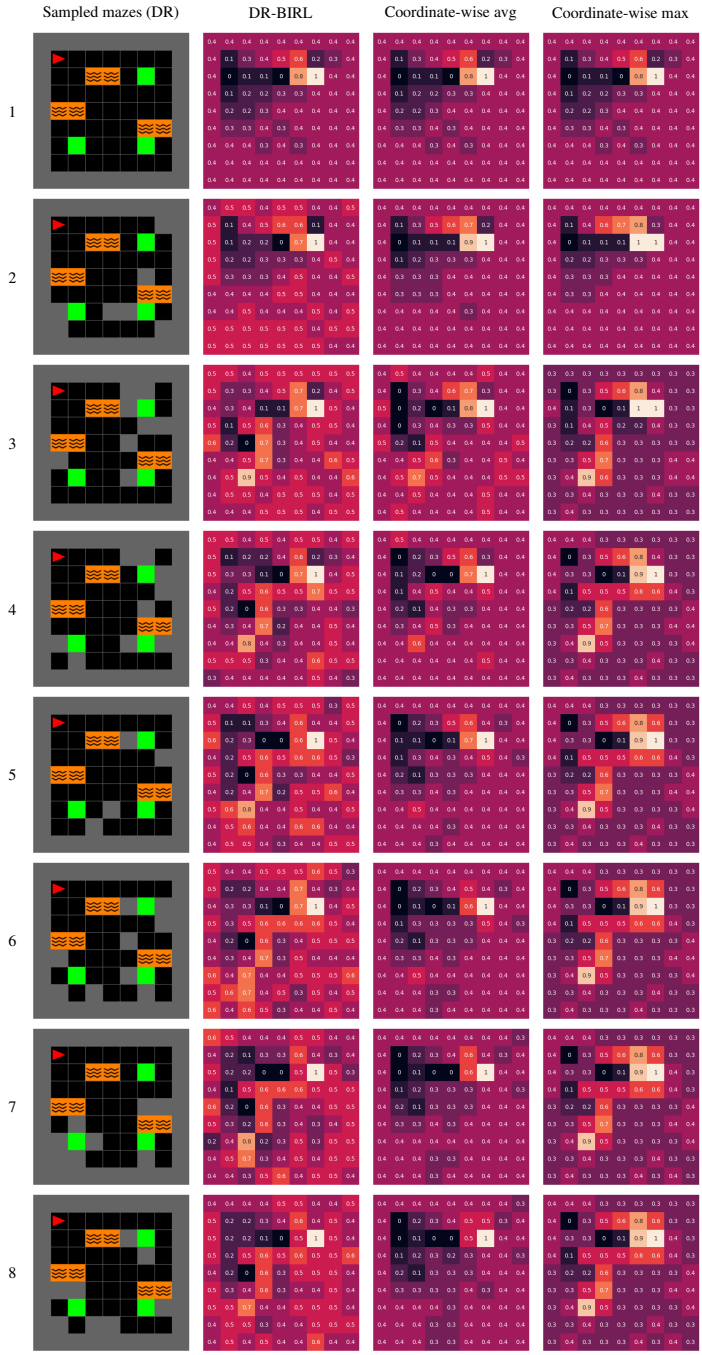



Figure 10: Selected mazes and reward estimates of DR−BIRL over the course of 8 environment selections (with two trajectories per environment). We here restrict the set of environments to those that do not block off the expert, i.e., no obstacles can be placed in the top left 4x4 squares. We additionally plot the estimated rewards when running BIRL in each environment separately and taking a coordinate-wise max or coordinate-wise average.

| Continuous Maze | Demo | | | Test | | |
|---|---|---|---|---|---|---|
| | Average | 25%-quantile | 75%-quantile | Average | 25%-quantile | 75%-quantile |
| ED-AIRL | **68±04** | **55±05** | **97±00** | **71±02** | **52±06** | **99±00** |
| DR-AIRL | 52±07 | 29±11 | 90±04 | 53±12 | 30±15 | 81±17 |
| AIRL | 33±09 | 08±11 | 69±09 | 52±07 | 30±09 | 76±09 |
| RIME | -105±12 | -115±05 | -45±02 | -52±03 | -57±03 | -39±02 |
| GAIL | 20±05 | -10±06 | 79±06 | 17±01 | -19±02 | 47±13 |
| BC | 11±00 | -23±00 | 71±01 | 22±00 | -18±00 | 49±00 |

| Hopper | Demo | | | Test | | |
|---|---|---|---|---|---|---|
| | Average | 25%-quantile | 75%-quantile | Average | 25%-quantile | 75%-quantile |
| ED-AIRL | **63±07** | 37±05 | 76±08 | 52±04 | 31±03 | **69±08** |
| DR-AIRL | 59±06 | 35±04 | 69±08 | **56±04** | 37±04 | 60±06 |
| AIRL | 38±03 | 20±02 | 51±05 | 34±04 | 19±02 | 44±07 |
| RIME | 61±01 | **47±01** | **80±03** | 53±02 | **42±02** | 66±02 |
| GAIL | 40±02 | 26±01 | 44±03 | 34±01 | 27±01 | 40±01 |
| BC | 32±01 | 23±01 | 33±01 | 27±01 | 22±00 | 30±01 |

| HalfCheetah | Demo | | | Test | | |
|---|---|---|---|---|---|---|
| | Average | 25%-quantile | 75%-quantile | Average | 25%-quantile | 75%-quantile |
| ED-AIRL | **40±11** | **17±10** | 61±14 | 35±13 | 20±13 | 46±16 |
| DR-AIRL | **40±13** | 16±11 | **63±14** | **40±11** | **23±10** | **54±12** |
| AIRL | 29±09 | 06±12 | 45±09 | 16±07 | 00±07 | 25±08 |
| RIME | -21±08 | -29±05 | -12±10 | -10±09 | -11±08 | -06±09 |
| GAIL | -12±02 | -28±02 | 01±02 | -06±01 | -14±01 | -06±01 |
| BC | -23±01 | -30±01 | -16±01 | -14±01 | -17±01 | -12±02 |

| Swimmer | Demo | | | Test | | |
|---|---|---|---|---|---|---|
| | Average | 25%-quantile | 75%-quantile | Average | 25%-quantile | 75%-quantile |
| ED-AIRL | 80±16 | 51±15 | 110±20 | 69±12 | 53±11 | 90±15 |
| DR-AIRL | 45±04 | 22±04 | 62±06 | 53±05 | 36±05 | 71±07 |
| AIRL | 40±10 | 11±07 | 66±13 | 44±08 | 19±08 | 68±10 |
| RIME | -05±01 | -08±01 | -03±01 | -04±01 | -07±01 | -02±00 |
| GAIL | 111±00 | 86±01 | 132±00 | 110±01 | 108±00 | 121±00 |
| BC | **124±01** | **100±01** | **156±00** | **130±01** | **116±00** | **160±00** |

| Ant | Demo | | | Test | | |
|---|---|---|---|---|---|---|
| | Average | 25%-quantile | 75%-quantile | Average | 25%-quantile | 75%-quantile |
| ED-AIRL | -71±03 | -79±03 | -46±03 | -86±05 | -98±05 | -55±04 |
| DR-AIRL | -73±03 | -88±03 | -47±03 | -95±03 | -114±05 | -61±03 |
| AIRL | -73±03 | -89±05 | -47±03 | -97±05 | -113±08 | -62±03 |
| RIME | -43±09 | -64±11 | -23±06 | -45±08 | -59±10 | -27±05 |
| GAIL | **69±03** | 45±03 | **89±05** | **75±04** | 44±03 | **96±04** |
| BC | 62±01 | **49±01** | 72±01 | 69±01 | **47±02** | 84±02 |

Table 3: Normalised scores for the continuous experiments. The quantiles are calculated by taking the average of the quantiles of each individual run.

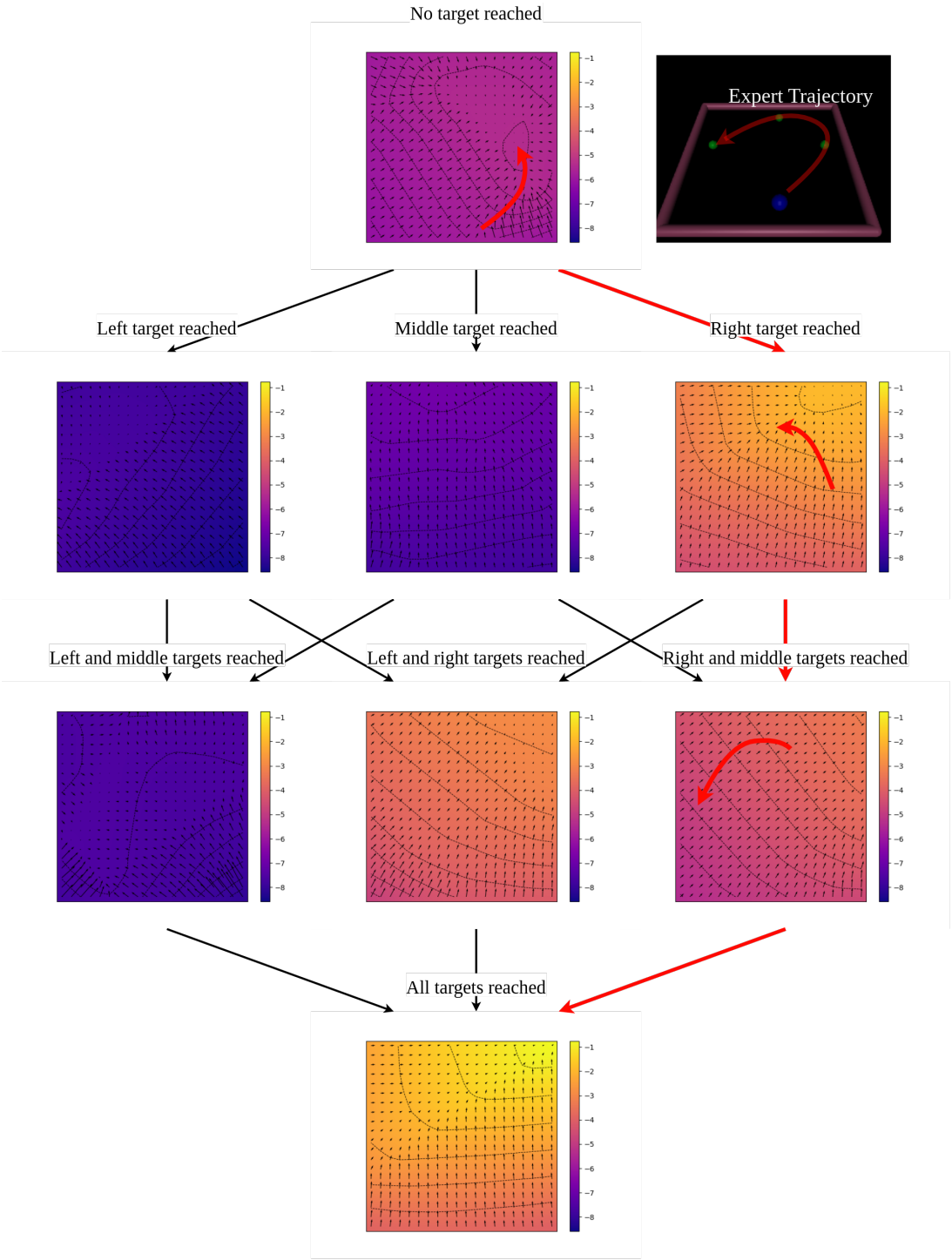



Figure 11: Reward function learned by AIRL on the continuous maze task. The reward function assigns large reward to the trajectories that gather targets in the same order as the expert, i.e., right-middle-left, which corresponds to the right side of the tree. It does not capture that the order in which the green states are reached does not matter. Hence, the learned reward function "overfits" to the expert trajectory. This function may yield undesired behaviors if the agent is forced to gather targets in a different order, or if a target is blocked off. For instance, if the right target is inaccessible, the reward function penalizes the agent for collecting the left target and/or the middle target.

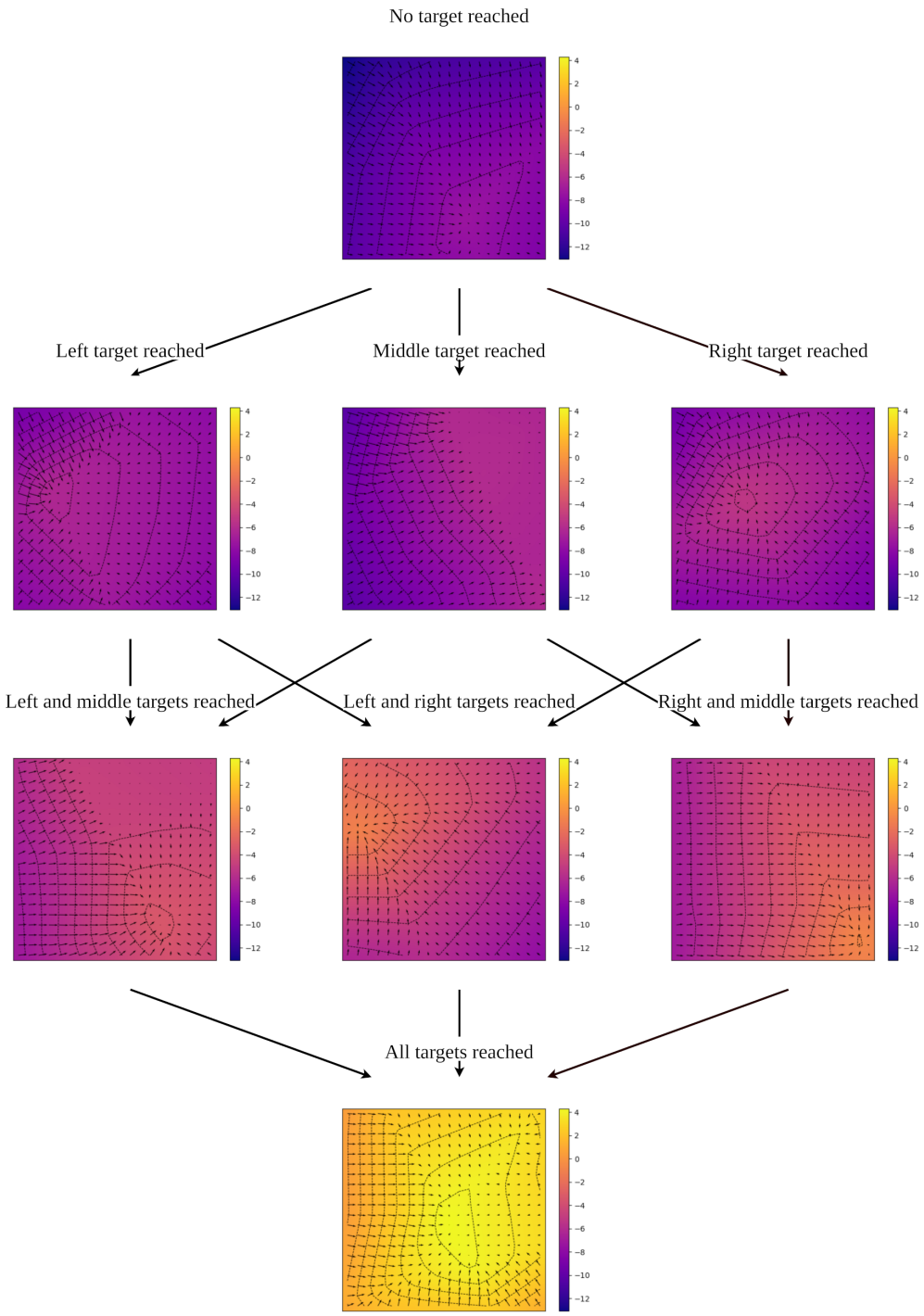



Figure 12: Reward function learned by ED-AIRL on the continuous maze task. The reward function does not favor a path more than another. It furthermore encourages the agent to collect targets even if some are blocked off.