THE IMPLICIT INFORMATION IN AN INITIAL STATE

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) agents optimize only the features specified in a reward function and are indifferent to anything left out inadvertently. This means that we must not only tell a household robot what to do, but also prevent it from knocking over a vase or stepping on a toy train. It is easy to forget these preferences, since we are so used to having them satisfied. Our key insight is that *when a robot is deployed in an environment that humans act in, the state of the environment is already optimized for what humans want*. We can therefore use this implicit information from the state to fill in the blanks. We develop an algorithm based on Maximum Causal Entropy IRL and use it to evaluate the idea in a suite of proof-of-concept environments designed to show its properties. We find that information from the initial state can be used to infer both side effects that should be avoided as well as preferences for how the environment should be organized.

1 INTRODUCTION

Reinforcement learning (RL) has been shown to succeed at a wide variety of complex tasks given a correctly specified reward function. Unfortunately, for many real-world tasks it can be challenging to specify a reward function that fully captures human preferences, particularly the preference for avoiding unnecessary side effects, while still accomplishing the goal (Amodei et al., 2016). As a result, there has been much work recently on learning human preferences (Fu et al., 2017; Christiano et al., 2017; Sadigh et al., 2017), which aims to learn specifications for tasks a robot should perform.

Typically when learning about what people want and don't want, we look to human action as evidence: what reward they specify, how they themselves do a task, what choices they make, how they rank certain options (citations for each of these). Here, we argue that there is an additional source of information that is potentially rather helpful, but that we have been ignoring thus far:

> *The key insight of this paper is that when a robot is deployed in an environment that humans have been acting in, the state of the environment is already optimized for what humans want.*

For example, consider an environment in which a household robot must navigate to a goal location without breaking any vases in its path, illustrated in Figure 1. The human operator, Alice, asks the robot to go to the purple door, forgetting to specify that it should also avoid breaking vases along the way. However, since the robot has been deployed in a state that only contains unbroken vases, it can infer that while acting in the environment (prior to robot's deployment), Alice was using one of the relatively few policies that do not break vases, and so must have cared about keeping vases intact.

The initial state s_0 can contain information about arbitrary preferences, including on tasks that the robot should actively perform. For example, if the robot observes a basket full of apples near an apple tree, it can reasonably infer that Alice wants to harvest apples. However, s_0 is particularly useful for inferring which side effects humans care about. Recent approaches for avoiding unnecessary side effects (Krakovna et al., 2018; Turner, 2018) do so by penalizing changes from an inaction baseline. The inaction baseline is appealing precisely because the initial state has already been optimized for human preferences, and action is more likely to ruin s_0 than inaction. If our robot can infer preferences from s_0, it should be able to avoid the side effects we care about.

Our contributions are threefold. First, we identify the state of the world at initialization as a source of information about human preferences that can correct a mis-specified reward. Second, we leverage this insight to derive an algorithm, Reward Learning by Simulating the Past (RLSP), which infers

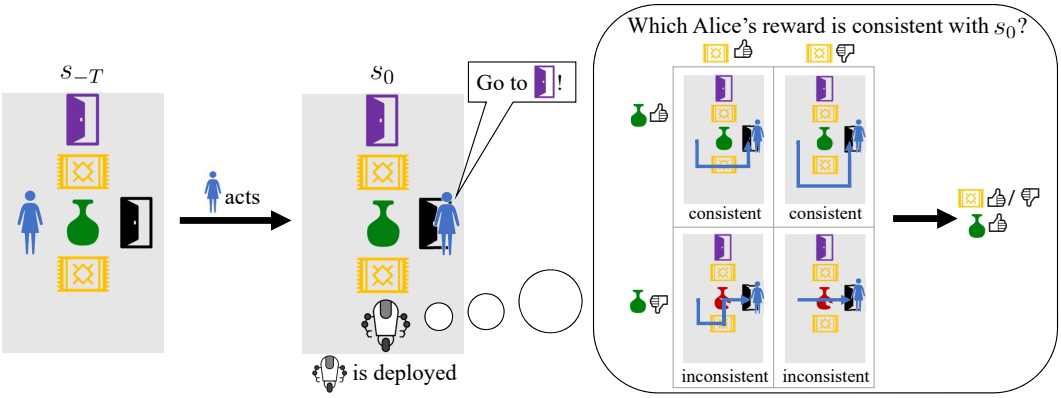


Figure 1: An illustration of learning preferences from an initial state. Alice attempts to accomplish a goal in an environment with an easily breakable vase in the center. The robot observes the state of the environment, s_0, after Alice has acted for some time from an even earlier state s_{-T}. It considers multiple possible human reward functions, and infers that states where vases are intact usually occur when Alice's reward penalizes breaking vases. In contrast, it doesn't matter much what the reward function says about carpets, as we would observe the same final state either way. Note that while we consider a specific s_{-T} for clarity here, the robot could also reason using a distribution over s_{-T}.

reward from initial state based on a Maximum Causal Entropy (Ziebart, 2010) model of human behavior. Combining the inferred reward with the specified reward encourages the robot to accomplish the task while still respecting the part of Alice's reward function that was left out of the specification. In Figure 1 the robot would move to the purple door without breaking the vase, even though Alice's instruction said nothing about breaking vases. Third, we demonstrate the properties and limitations of this idea on a suite of proof-of-concept environments: we use it to avoid side effects, as well as to learn implicit preferences that require active action from the robot.

This work is about highlighting the potential of this observation, and as such makes unrealistic assumptions, such as known dynamics and hand-coded features. Nonetheless, we are optimistic that there is a path towards relaxing these assumptions, and suggest some approaches in our discussion.

2 Related Work

Preference learning. Preference learning or specification learning has had a surge of interest in recent years, with preferences learned from demonstrations (Ziebart, 2010; Ramachandran & Amir, 2007; Ho & Ermon, 2016; Fu et al., 2017; Finn et al., 2016), comparisons (Christiano et al., 2017; Sadigh et al., 2017; Wirth et al., 2017), human reinforcement signals (Knox & Stone, 2009; Warnell et al., 2017; MacGlashan et al., 2017), and more (Hadfield-Menell et al., 2017) . We suggest preference learning with a new source of data: the state of the environment when the robot is first deployed. It can also be seen as a variant of Maximum Causal Entropy Inverse Reinforcement Learning (Ziebart, 2010). While recent work has extended IRL to learn a reward (Edwards et al., 2018) or loss (Yu et al., 2018) given a demonstration without actions, we learn a reward function from a single state, albeit with the simplifying assumption of known dynamics.

Frame properties. The frame problem in AI (McCarthy & Hayes, 1981) refers to the issue that we must specify what stays the same in addition to what changes. In formal verification, this manifests as a requirement to explicitly specify the many quantities that the program does not change (Andreescu, 2017). Analogously, rewards are likely to specify what to do (the task), but may forget to say what not to do (the frame properties). One of our goals is to infer frame properties automatically.

Side effects. One way to mitigate reward specification problems is to use an impact penalty in order to disincentivize making unnecessary "large" changes (Armstrong & Levinstein, 2017). Compared to the baseline of doing nothing, we could penalize a reduction in the number of reachable states (Krakovna et al., 2018) or attainable utility (Turner, 2018). We claim that the desire to avoid side effects stems from the fact that the world is already optimized for human preferences, and so by

inferring the preferences that lead to the observed world state we can infer which side effects humans care about.

Goal states as specifications. Desired behavior in RL can be specified with an explicitly chosen goal state (Kaelbling, 1993; Schaul et al., 2015; Nair et al., 2018; Bahdanau et al., 2018; Andrychowicz et al., 2017), where the robot stops acting once it has reached the goal state. In contrast, we consider the case where s_0 is not explicitly chosen by the designer, but nonetheless contains information about the desired behavior, and the robot *starts* acting from this state.

3 PRELIMINARIES

A finite-horizon Markov decision process (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, H \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition probability function, $r : \mathcal{S} \mapsto \mathbb{R}$ is the reward function, and $H \in \mathbb{Z}_+$ is the finite planning horizon. We consider MDPs where the reward is linear in features of the state, and does not depend on action: $r(s) = \theta^T f(s)$.

Inverse Reinforcement Learning (IRL). In IRL, the aim is to infer the reward function r given an MDP without reward $\mathcal{M} \backslash r$ and expert demonstrations $\mathcal{D} = \{\tau_1, ..., \tau_n\}$, where each $\tau_i = (s_0, a_0, ..., s_H, a_H)$ is a trajectory sampled from the expert policy acting in the MDP.

Maximum Causal Entropy IRL. As human demonstrations are rarely optimal, Ziebart (2010) introduced the Maximum Causal Entropy IRL (MCEIRL) framework that models the expert as a Boltzmann-rational agent that maximizes the sum of the expected causal entropy of its policy and the expected total reward. This leads to the policy $\pi(a \mid s) = \exp(Q_t(s, a) - V_t(s))$, where $V_t(s) = \ln \sum_a \exp(Q_t(s, a))$. Intuitively, the expert is assumed to act according to a stochastic policy that is close to random when the difference in expected total reward across the actions is small, and is close to deterministic when one action leads to a substantially higher expected return than the others. The state-action value function Q is obtained by applying the soft Bellman backup, $Q_t(s, a) = r_s + \sum_{s'} \mathcal{T}(s' \mid s, a) V_{t+1} s'$.

The likelihood of a trajectory τ given the reward parameters θ is:

$$p(\tau \mid \theta) = p(s_0) \left(\prod_{t=0}^{H-1} \mathcal{T}(s_{t+1} \mid s_t, a_t) \pi(a_t \mid s_t) \right) \pi(a_H \mid s_H) \tag{1}$$

MCEIRL finds the reward parameters θ^* that maximize the log-likelihood of the demonstrations:

$$\theta^* = \text{argmax}_\theta \ln p(\mathcal{D} \mid \theta) = \text{argmax}_\theta \sum_i \sum_t \ln \pi(a_{i,t} \mid s_{i,t}) \tag{2}$$

Maximizing the likelihood from Equation 2 with gradient descent results in an algorithm that finds a reward giving rise to a policy whose feature expectations match those of the expert demonstrations.

4 INVERSE REINFORCEMENT LEARNING FROM ONE STATE

In this work we solve the problem of learning the reward function of an expert given a single final state of the expert's trajectory; we refer to this problem as *inverse reinforcement learning from a single state*. Formally, the aim of IRL from a single state is to infer the expert's reward parameters θ given an environment $\mathcal{M} \backslash r$ and s_0, the last state of the expert's trajectory.

4.1 REWARD LEARNING BY SIMULATING THE PAST

Formulation. To adapt MCEIRL to the one state setting we modify the observation model from Equation 1. Since we only have a single end state s_0 of the trajectory $\tau_0 = (s_{-T}, a_{-T}, ..., s_0, a_0)$, we marginalize over all of the other variables in the trajectory:

$$p(s_0 \mid \theta) = \sum_{s_{-T}, a_{-T}, ... s_{-1}, a_{-1}, a_0} p(\tau_0 \mid \theta) \tag{3}$$

where $p(\tau_0 \mid \theta)$ is given in Equation 1. The new problem is as follows:

$$\theta^* = \text{argmax}_\theta \ln p(s_0 \mid \theta) \tag{4}$$

Solution. Similarly to MCEIRL, we use a gradient ascent algorithm to solve the IRL from one state problem. We explain the key steps here and give the full derivation in Appendix A. We start by taking the gradient of the new objective w.r.t. θ and simplify to a form where we can use the gradient from Ziebart (2010):

$$\nabla_\theta \ln p(s_0 \mid \theta) = \sum_{s_{-T:-1}, a_{-T:-1}} p(\tau_{-T:-1} \mid s_0, \theta) \nabla_\theta \ln p(\tau_{-T:0} \mid \theta)$$

This has a nice interpretation – compute the Maximum Causal Entropy gradients for each trajectory, and then take their weighted sum, where each weight is the probability of the trajectory given the evidence s_0 and current reward θ. Substituting in the gradient from Ziebart (2010) gives:

$$\nabla_\theta \ln p(s_0 \mid \theta) = \frac{1}{p(s_0 \mid \theta)} \mathbb{E}_{\tau_{-T:-1}} \left[p(s_0 \mid \tau_{-T:-1}, \theta) \sum_{t=-T}^{0} f(s_t) \right] - \mathbb{E}_{\tau_{-T:0}} \left[\sum_{t=-T}^{0} f(s_t) \right] \quad (5)$$

Intuitively, we take the feature expectations weighted by the likelihood of the observed state s_0, and subtract the unweighted feature expectations. The factor $\frac{1}{p(s_0|\theta)}$, which we call the s_0 *occupancy measure*, is a normalizer. Note that we can easily incorporate a prior on θ by adding the gradient of the log prior to the gradient in Equation 5.

Since our algorithm works by simulating trajectories in the past and combining their gradients, we name our method Reward Learning by Simulating the Past (RLSP).

Fast computation. The unweighted feature expectations $\mathbb{E}_{\tau_{-T:0}} \left[\sum_{t=-T}^{0} f(s_t) \right]$ can be computed with Algorithm 2 from (Ziebart, 2010). The s_0 occupancy measure $p(s_0 \mid \theta)$ and the weighted feature expectations $\mathbb{E}_{\tau_{-T:-1}} \left[p(s_0 \mid \tau_{-T:-1}, \theta) \sum_{t=-T}^{0} f(s_t) \right]$ can be computed using dynamic programming, detailed in Appendix A.

4.2 Alternative: MCMC sampling

An alternative to estimating the MLE as above (or MAP if we have a prior) is to approximate the entire posterior distribution. This can be useful especially for risk-averse planning, or for deciding to actively get more information from the human. One standard way to address the computational challenges involved with the continuous and high-dimensional nature of θ is to use MCMC sampling to sample from $p(\theta \mid s_0) \propto p(s_0 \mid \theta) p(\theta)$. The resulting algorithm resembles Bayesian IRL (Ramachandran & Amir, 2007) and is presented in Appendix B. In this paper, we follow Bayesian IRL by collapsing the full distribution into a point estimate by taking the mean, and leave more sophisticated approaches to future work.

5 Combining the specified reward with the inferred reward

The algorithms in Section 4 allow us to invert our observation model $P(s_0 \mid \theta)$ to get $P(\theta \mid s_0)$. However, we are also presented with a reward specification θ_{spec}, which we must also incorporate in order to get $P(\theta \mid s_0, \theta_{\text{spec}})$. A Bayesian approach would be to condition the prior over θ on θ_{spec}, which gives us $P(\theta \mid s_0, \theta_{\text{spec}}) \propto P(s_0 \mid \theta) P(\theta \mid \theta_{\text{spec}})$. This can easily be incorporated by replacing the prior centered at zero with a prior centered at θ_{spec}. However, this introduces a subtlety. The θ that we wish to infer is the ideal reward that the robot should optimize, θ_R, whereas the θ in our observation model $P(s_0 \mid \theta)$ is the reward θ_{Alice} that *Alice* was optimizing. Thus, the two terms pull in different directions – the observation model $P(s_0 \mid \theta)$ is high when θ is close to θ_{Alice}, whereas the prior $P(\theta \mid \theta_{\text{spec}})$ is high when θ is close to θ_{spec}.

This tradeoff is inevitable given our problem formulation, since we have two different sources of information (θ_{spec} and s_0) and they will conflict in some cases. However, we can make this tradeoff outside of the probabilistic model, by first inferring θ_{Alice} by inverting our observation model $P(s_0 \mid \theta_{\text{Alice}})$ and then adding it to the specified reward, giving us a final reward of $\theta_{\text{Alice}} + \lambda \theta_{\text{spec}}$. We call this the *Additive* method, as opposed to the *Bayesian* method above.

6 EVALUATION

We demonstrate RLSP in a number of environments designed to show its properties and limitations. In all settings, we design an environment with a true reward R_{true}, a specified reward R_{spec} that ignores some aspects of the true reward, the state s_{-T}, and the initial state for the robot s_0. We inspect the inferred reward R_{inferred} qualitatively and measure the expected amount of true reward obtained when planning with traditional value iteration using R_{inferred}, as a fraction of the expected true reward from the optimal policy. We tune the hyperparameter controlling the tradeoff between R_{spec} and the human reward for all algorithms, including baselines. We use a Gaussian prior over the reward parameters.

In initial experiments, we found that sampling from the posterior distribution was significantly slower and noisier than the gradient-based RSLP, and so we did not test it further.

6.1 BASELINES

Specified reward policy π_{spec}. We act as if the true reward is exactly the specified reward.

Policy that penalizes deviations $\pi_{\text{deviation}}$. This baseline minimizes change in a general sense by penalizing any deviations from the observed state s_0 in feature space, giving the augmented reward $R'(s) = \theta_{\text{spec}}^T f(s) + \lambda ||f(s) - f(s_0)||$.

Relative reachability policy $\pi_{\text{reachability}}$. Relative reachability (Krakovna et al., 2018) considers a change to be negative when it decreases *coverage*, relative to what would have happened had the agent done nothing. Here, coverage is a measure of how easily states can be reached from the current state. We compare against the variant of relative reachability that uses undiscounted coverage and a baseline policy where the agent takes no-op actions, as in the original paper. Note that while relative reachability makes use of known dynamics, it does not benefit from our handcoded featurization.

6.2 COMPARISON TO BASELINES

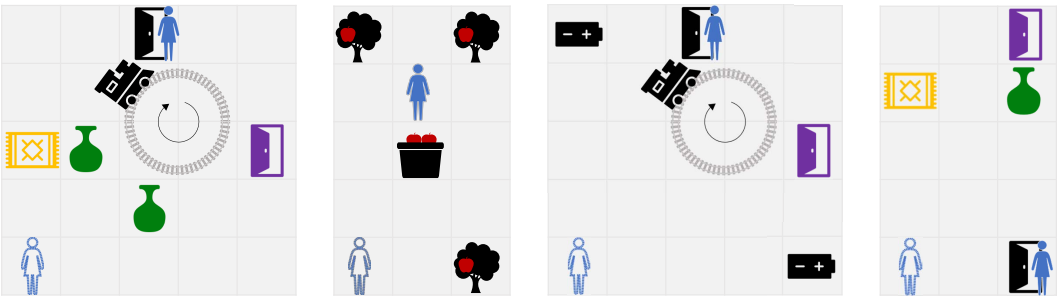
For these experiments, we consider RLSP with the Additive combination method, and compare to our baselines. We make the unrealistic assumption of known s_{-T}, because it makes it easier to analyze RLSP's properties, and consider the case of unknown s_{-T} in Section 6.4. Results are summarized in Table 1.

Table 1: Performance of algorithms on environments designed to test particular properties.

	Side effects Room	**Env effect** Toy train	**Implicit reward** Apple collection	**Desirable effect** Batteries		**Unseen effect** Far away vase
				Easy	Hard	
π_{spec}	✗	✗	✗	✓	✗	✗
$\pi_{\text{deviation}}$	✓	✗	✗	≈	✗	✓
$\pi_{\text{reachability}}$	✓	✓	✗	≈	✗	✓
π_{RLSP}	✓	✓	✓	✓	✓	✗

Side effects: Room with vase *(Figure 1)*. The room tests whether the robot can avoid breaking a vase as a side effect of going to the purple door. There are features for the number of broken vases, standing on a carpet, and each door location. π_{spec} takes the shortest path to the door, breaking the vase. Since Alice didn't walk over the vase, RLSP infers a negative reward on broken vases, and a small positive reward on carpets (since paths to the top door usually involve carpets). π_{RLSP} successfully avoids breaking the vase. The baselines also achieve the desired behavior, albeit for different reasons. $\pi_{\text{deviation}}$ avoids breaking the vase since it would change the "number of broken vases" feature, while relative reachability avoids breaking the vase since doing so would result in all states with intact vases becoming unreachable.

Distinguishing environment effects: Toy train *(Figure 2a)*. To test whether algorithms can distinguish between effects caused by the agent and effects caused by the environment, as suggested in Krakovna et al. (2018), we add a toy train that moves along a predefined track. The train breaks if the agent steps on it. We add a new feature indicating whether the train is broken and new features



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

Figure 2: The environments used to examine the properties of our approach. (a) Distinguishing environment effects: Toy train (b) Implicit reward: Apple collection (c) Desirable side effect: Batteries (d) "Unseen" side effect: Room with far away vase. The transparent human silhouettes indicate Alice's position at s_0.

for each possible train location. As before, the specified reward only has a positive weight on the purple door, while the true reward also penalizes broken trains and vases.

π_{spec} steps on the train on its way to the door. In contrast, RLSP infers a negative reward on broken vases and broken trains, for the same reason as in the previous environment. It also infers not to put any weight on any particular train location, even though they change frequently, because those features don't help explain the observed state s_0. As a result, π_{RLSP} walks over a carpet to get to the goal, but not a vase or a train. $\pi_{\text{deviation}}$ immediately breaks the train, so that the train location feature stays the same. $\pi_{\text{reachability}}$ deduces that breaking the train would make states with intact trains unreachable, and so follows the same trajectory as π_{RLSP}.

Implicit reward: Apple collection *(Figure 2b)*. This environment tests whether the algorithms can learn tasks implicit in s_0. There are three trees that grow apples, as well as a basket for collecting apples, and the goal is for the robot to harvest apples. However, the specified reward is zero: the robot must infer the task from the observed state. We have features for the number of apples in baskets, the number of apples on trees, whether the robot is carrying an apple, and each location that the agent could be in. s_0 has two apples in the basket, while s_{-T} has none.

Since the specified reward is zero, when optimizing it in isolation, every policy is optimal, and π_{spec} is arbitrary. Since deviation applies a non-negative penalty, its maximum reward is zero, so $\pi_{\text{deviation}}$ does nothing. $\pi_{\text{reachability}}$ also does not harvest apples. RLSP infers a positive reward on the number of apples in baskets, as well as a small negative reward for apples on trees and a small positive reward for carrying apples. Despite these spurious weights, π_{RLSP} harvests apples as desired.

Desirable side effect: Batteries *(Figure 2c)*. This environment tests whether the algorithms can tell when a side effect is allowed. We take the toy train environment, remove the vases and the carpets, and add batteries. The robot can pick up batteries and put them into the (now unbreakable) toy train, but the batteries are never replenished. If no batteries have been put in the train for 10 timesteps, it stops operating. There are features for the number of batteries, whether the train is operational, the train location, and each door location. There are two batteries at s_{-T} but only one at s_0. The true reward puts a positive weight on the train being operational and the robot being at the purple door. We consider two variants for the task reward – an "easy" case, where the task reward equals the true reward, and a "hard" case, where the task reward only rewards being at the purple door.

Unsurprisingly, π_{spec} succeeds at the easy case, and fails on the hard case, where it allows the train to run out of power. Both $\pi_{\text{deviation}}$ and $\pi_{\text{reachability}}$ see the action of putting a battery in the train as a side effect to be penalized, and so neither can solve the hard case, and they only solve the easy case if the penalty weight is sufficiently small. RLSP sees that one battery is gone and that the train is operational, and infers that Alice wants the train to be operational and doesn't want batteries, since it is difficult to distinguish a preference against batteries from a preference for an operational train. This allows it to solve both the easy and the hard case, with π_{RLSP} picking up the battery, then staying at the purple door except to deliver the battery to the train when it runs out of power.

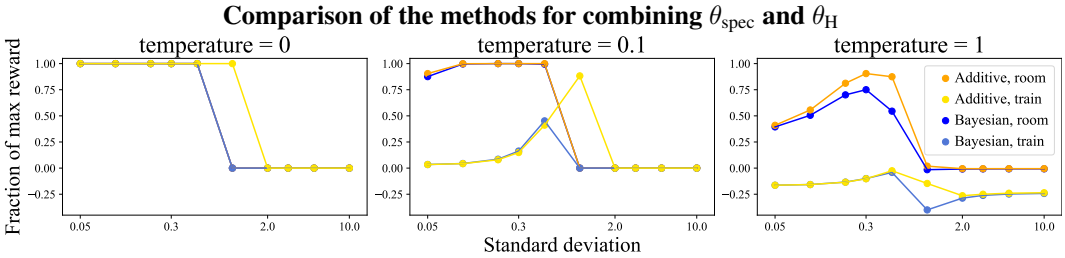


Figure 3: Comparison of the Bayesian and Additive methods on the **Room with vase** and **Toy train** environments. We show how the percentage of true reward obtained by π_{RLSP} varies with the standard deviation of the prior for each method and environment. The softmax temperature for π_{RLSP} is varied across subplots; temperature equal to zero indicates a robot following a greedy policy obtained with traditional value iteration. With traditional value iteration, behavior is often identical and so the lines overlap. Overall, there is not much difference between the two methods.

"Unseen" side effect: Room with far away vase *(Figure 2d)*. This environment demonstrates a limitation of our algorithm: it cannot identify side effects that Alice would never have triggered. In this room, the vase is nowhere close to the shortest path from the Alice's original position to her goal, but is on the path to the robot's goal. Since our baselines don't care about the trajectory the human takes, they all perform as before – π_{spec} walks over the vase, while $\pi_{\text{deviation}}$ and $\pi_{\text{reachability}}$ both avoid it. Our method infers a near zero weight on the broken vase feature, since it is not present on any reasonable trajectory to the goal, and so breaks it when moving to the goal. Note that this only applies when Alice is known to be at the bottom left corner at s_{-T}: if we have a uniform prior over s_{-T} (considered in Section 6.4) then we do consider trajectories where vases are broken.

6.3 Comparison between methods for combining θ_{spec} and θ_{H}

Section 5 introduced two methods for combining the inferred reward θ_{H} with θ_{spec} to obtain θ_{R}, the reward the robot should optimize. The Bayesian method uses θ_{spec} as the mean of the prior in RLSP, while the Additive method computes a linear combination of θ_{Alice} and θ_{spec}. To compare these methods, we evaluate their robustness by varying the standard deviation σ of the Gaussian prior over θ_{Alice} and reporting the true reward obtained by π_{RLSP}, as a fraction of the maximum expected true reward. While we typically create π_{RLSP} using value iteration, this leads to deterministic policies with very sharp changes in behavior that make it hard to see differences between methods, and so we also show results with soft value iteration, which creates stochastic policies that vary more continuously. As demonstrated in Figure 3, our experiments show that overall the two methods perform very similarly, with some evidence that the Additive method is slightly more robust.

6.4 Comparison between knowing s_{-T} vs. a distribution over s_{-T}

So far, we have considered the setting where the robot knows s_{-T}, since it is easier to analyze what happens. However, oftentimes we will not know s_{-T} exactly, and will instead have some prior over s_{-T}. Here, RLSP with the Additive reward combination method is compared in two settings: perfect knowledge of s_{-T} (as in Section 6.2) and a uniform distribution over all states.

Side effects: Room with vase *(Figure 1)* **and toy train** *(Figure 2a)*. In room and toy train, a uniform prior leads to *better* results. In room with vase, RLSP learns a stronger negative reward on broken vases, and the small positive reward on carpets changes to a near-zero negative reward on carpets. In toy train, RLSP learns much stronger negative rewards on broken vases and trains, while leaving the other feature weights approximately the same. We hypothesize that this results from considering many more feasible trajectories when using a uniform prior. On all of these trajectories, vases and trains are not broken, whereas in some of them Alice walks over carpets and in others Alice does not. In contrast, with known s_{-T}, there are only a few trajectories consistent with the evidence, and so the reward overfits to those trajectories. For example, in room with vase with known s_{-T}, the shortest trajectories go over carpets, and so RLSP infers a positive weight on carpets.

Implicit preference: Apple collection *(Figure 2b)*. Here, a uniform prior leads to a very strong negative weight on the number of apples in baskets, which is exactly the opposite of what we want. Intuitively, this is because RLSP is considering cases where s_{-T} already has two or more apples in the basket. Indeed, if we instead have a uniform prior over all states s_{-T} where the basket is empty, we recover the good apple harvesting behavior from the known s_{-T} case, with a slightly stronger positive reward for number of apples in baskets.

Desirable side effects: Batteries *(Figure 2c)*. With the uniform prior, we see the same issue as in apples, where the negative reward on batteries becomes positive. This causes π_{RLSP} to fail in the hard case, while in the easy case the extra incentive from the task reward causes it to succeed anyway. If we again restrict the prior to be uniform over states with exactly two batteries, then we once again recover the negative reward on batteries which leads to good behavior from π_{RLSP} in both cases.

"Unseen" side effect: Room with far away vase *(Figure 2d)*. With a uniform prior, we "see" the side effect: if Alice started at the purple door, then the shortest trajectory to the black door would break a vase. As a result, π_{RLSP} successfully avoids the vase (whereas it previously did not).

Overall, having uncertainty over the initial state s_{-T} counterintuitively *improves* results, because it increases the diversity of trajectories considered in $p(s_0 \mid \theta)$, which prevents RLSP from "overfitting" to the few trajectories consistent with a known s_{-T} and s_0.

7 Discussion and future work

Summary. Our key insight is that when a robot is deployed, the state that it observes has already been optimized to satisfy human preferences. This explains our preference for a policy that generally avoids side effects. We formalized this by assuming that Alice has been acting in the environment prior to the robot's deployment. We developed an algorithm, RLSP, that computes a MAP estimate of Alice's reward function. The robot then acts according to a tradeoff between Alice's reward function and the specified reward function. Our evaluation showed that information from the initial state can be used to successfully infer side effects to avoid as well as tasks to complete, though there are cases in which we cannot infer the relevant preferences. While we believe this is an important step forward, there is still much work to be done to make this accurate and practical.

Realistic environments. The primary avenue for future work is to scale to realistic environments, where we cannot enumerate states, we don't know dynamics, and the reward function may be nonlinear. This could be done by adapting existing IRL algorithms (Fu et al., 2017; Ho & Ermon, 2016; Finn et al., 2016). Unknown dynamics is particularly challenging, since we cannot learn dynamics from a single state observation. We could consider the case where we have access to a simulator, from which we can learn an approximate dynamics model. Alternatively, we can learn dynamics while acting, and update the learned preferences as our model improves over time.

Combining θ_{spec} and θ_{Alice}. In Section 5, we saw that combining the inferred θ_{Alice} with the specified reward led to a strong tradeoff between the two. Intuitively, θ_{Alice} can be decomposed into $\theta_{\text{Alice,task}}$, which says which task Alice is performing ("go to the black door"), and θ_{frame}, which consists of the frame conditions ("don't break vases"). The robot could then optimize $\theta_{\text{spec}} + \theta_{\text{frame}}$, ameliorating the tradeoff significantly. Since θ_{frame} is in large part shared, we could accomplish this using models where multiple humans are optimizing their own unique $\theta_{H,\text{task}}$ but the same θ_{frame}, or we could have one human with goals that change over time. Another direction would be to assume a different structure for the frame conditions, such as constraints, and learn frame conditions separately.

Learning tasks to perform. The apples and batteries environments demonstrate that RLSP can learn preferences that require the robot to actively perform a task. It is not clear that this is desirable, since the robot may perform an inferred task instead of the task Alice explicitly sets for it.

Preferences that are not a result of human optimization. While the initial state of the environment is optimized towards human preferences, it need not be the case that this is a result of *human* optimization, as assumed in this paper. For example, we have a preference for the atmosphere to contain a certain percentage of oxygen and carbon dioxide, and the atmosphere meets this preference, but not as a result of human optimization. In this case, humans were "optimized" by evolution to prefer the environment that already exists. While this seems to be of limited relevance for household robots, it may become important for more capable AI systems.

REFERENCES

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Oana Fabiana Andreescu. *Static analysis of functional programs with an application to the frame problem in deductive verification*. PhD thesis, Rennes 1, 2017.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Stuart Armstrong and Benjamin Levinstein. Low impact artificial intelligences. *arXiv preprint arXiv:1705.10720*, 2017.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. Learning to follow language instructions with adversarial reward induction. *arXiv preprint arXiv:1806.01946*, 2018.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pp. 4299–4307, 2017.

Ashley D Edwards, Himanshu Sahni, Yannick Schroeker, and Charles L Isbell. Imitating latent policies from observation. *arXiv preprint arXiv:1805.07914*, 2018.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pp. 49–58, 2016.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, pp. 6765–6774, 2017.

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

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

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

Victoria Krakovna, Laurent Orseau, Miljan Martic, and Shane Legg. Measuring and avoiding side effects using relative reachability. *arXiv preprint arXiv:1806.01186*, 2018.

James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, David Roberts, Matthew E Taylor, and Michael L Littman. Interactive learning from policy-dependent human feedback. *arXiv preprint arXiv:1701.06049*, 2017.

John McCarthy and Patrick J Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Readings in artificial intelligence*, pp. 431–450. Elsevier, 1981.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *arXiv preprint arXiv:1807.04742*, 2018.

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

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

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015.

Alex Turner. Towards a new impact measure, 2018. https://www.alignmentforum.org/posts/yEa7kwoMpsBgaBCgb/towards-a-new-impact-measure, Last accessed on 2018-09-26.

Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. *arXiv preprint arXiv:1709.10163*, 2017.

Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A Survey of Preference-Based Reinforcement Learning Methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017. ISSN 15337928. doi: 10.1109/CIS.2008.204. URL http://jmlr.org/papers/v18/16-634.html.

Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

A

This section provides a derivation of the gradient $\nabla_\theta \ln p(s_0 \mid \theta)$, which is needed to solve $\text{argmax}_\theta \ln p(s_0 \mid \theta)$ with gradient ascent. The full expert trajectory is $\tau_{-T:0} = \{s_{-T:0}, a_{-T:0}\}$, thus the expert is assumed to receive the last-step reward. This derivation makes use of the fact that

$$\nabla_\theta \ln p(\tau_{-T:0} \mid \theta) = \sum_{t=-T}^{0} f(s_t) - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0}|\theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right],$$

which is derived in (Ziebart, 2010).

$$\begin{aligned}
&\nabla_\theta \ln p(s_0 \mid \theta) \\
&= \frac{1}{p(s_0 \mid \theta)} \nabla_\theta p(s_0 \mid \theta) \\
&= \frac{1}{p(s_0 \mid \theta)} \sum_{s_{-T:-1}, a_{-T:0}} \nabla_\theta p(\tau_{-T:0} \mid \theta) \\
&= \frac{1}{p(s_0 \mid \theta)} \sum_{s_{-T:-1}, a_{-T:0}} \nabla_\theta \exp(\ln p(\tau_{-T:0} \mid \theta)) \\
&= \frac{1}{p(s_0 \mid \theta)} \sum_{s_{-T:-1}, a_{-T:0}} p(\tau_{-T:0} \mid \theta) \nabla_\theta \ln p(\tau_{-T:0} \mid \theta) \\
&= \frac{1}{p(s_0 \mid \theta)} \sum_{a_0} \sum_{s_{-T:-1}, a_{-T:-1}} p(\tau_{-T:-1}, s_0 \mid \theta) \pi(a_0 \mid s_0) \nabla_\theta \ln p(\tau_{-T:0} \mid \theta) \\
&= \sum_{s_{-T:-1}, a_{-T:-1}} \frac{p(\tau_{-T:-1}, s_0 \mid \theta)}{p(s_0 \mid \theta)} \nabla_\theta \ln p(\tau_{-T:0} \mid \theta) \\
&= \sum_{s_{-T:-1}, a_{-T:-1}} p(\tau_{-T:-1} \mid s_0, \theta) \nabla_\theta \ln p(\tau_{-T:0} \mid \theta)
\end{aligned}$$

This has a nice interpretation – compute the Maximum Causal Entropy gradients for each trajectory, $\nabla_\theta \ln p(\tau_{-T:0} \mid \theta)$, and then take their weighted sum, where each weight is the probability of the trajectory given the evidence s_0 and current reward θ, that is $p(\tau_{-T:-1} \mid s_0, \theta)$. Now, we can substitute in the gradient from Ziebart (2010):

$$\begin{aligned}
&= \sum_{s_{-T:-1}, a_{-T:-1}} p(\tau_{-T:-1} \mid s_0, \theta) \left(\sum_{t=-T}^{0} f(s_t) - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0}|\theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right] \right) \\
&= \sum_{s_{-T:-1}, a_{-T:-1}} \left[p(\tau_{-T:-1} \mid s_0, \theta) \sum_{t=-T}^{0} f(s_t) \right] \\
&\quad - \left[\sum_{s_{-T:-1}, a_{-T:-1}} p(\tau_{-T:-1} \mid s_0, \theta) \right] \left[\mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0}|\theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right] \right] \\
&= \sum_{s_{-T:-1}, a_{-T:-1}} \left[p(\tau_{-T:-1} \mid s_0, \theta) \sum_{t=-T}^{0} f(s_t) \right] - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0}|\theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right]
\end{aligned}$$

Continuing with the interpretation from before, we compute the weighted sum of the features of all trajectories, and then subtract the expected features under the reward θ. However, the term $p(\tau_{T-1} \mid s_0, \theta)$ is still hard to compute, so we break it apart into simpler terms:

$$
\begin{aligned}
&= \sum_{s_{-T:-1}, a_{-T:-1}} \left[\frac{p(\tau_{-T:-1}, s_0 \mid \theta)}{p(s_0 \mid \theta)} \sum_{t=-T}^{0} f(s_t) \right] - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0} \mid \theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right] \\
&= \sum_{s_{-T:-1}, a_{-T:-1}} \left[\frac{p(\tau_{-T:-1} \mid \theta) p(s_0 \mid \tau_{-T:-1}, \theta)}{p(s_0 \mid \theta)} \sum_{t=-T}^{0} f(s_t) \right] - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0} \mid \theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right] \\
&= \frac{1}{p(s_0 \mid \theta)} \mathbb{E}_{\tau_{-T:-1} \sim p(\tau_{-T:-1} \mid \theta)} \left[p(s_0 \mid \tau_{-T:-1}, \theta) \sum_{t=-T}^{0} f(s_t) \right] - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0} \mid \theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right] \\
&= \frac{1}{p(s_0 \mid \theta)} \mathbb{E}_{\tau_{-T:-1} \sim p(\tau_{-T:-1} \mid \theta)} \left[\mathcal{T}(s_0 \mid s_{-1}, a_{-1}) \sum_{t=-T}^{0} f(s_t) \right] - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0} \mid \theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right]
\end{aligned}
$$

The second term $\mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0} \mid \theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right]$ is simply the feature expectations of the policy that arises from θ, and can be computed with Algorithm 2 from (Ziebart, 2010). The term $p(s_0 \mid \theta)$ can be computed using the base case $p(s_{-T} \mid \theta) = p(s_{-T})$ and the recursive rule

$$
p(s_{t+1} \mid \theta) = \sum_{s_t, a_t} p(s_t \mid \theta) \pi(a_t \mid s_t, \theta) \mathcal{T}(s_{t+1} \mid s_t, a_t) \tag{6}
$$

For the remainder of the first term, we define:

$$
G_k(s_k) = \mathbb{E}_{\tau_{-T:k-1} \sim p(\tau_{-T:k-1} \mid \theta)} \left[\mathcal{T}(s_k \mid s_{k-1}, a_{k-1}) \sum_{t=-T}^{k} f(s_t) \right]
$$

Note that our gradient is then given by:

$$
\nabla_\theta \ln p(s_0 \mid \theta) = \frac{G_0(s_0)}{p(s_0 \mid \theta)} - \mathbb{E}_{\tau'_{-T:0} \sim p(\tau_{-T:0} \mid \theta)} \left[\sum_{t=-T}^{0} f(s'_t) \right]
$$

We now derive a recursive relation for G:

$$
\begin{aligned}
&G_{k+1}(s_{k+1}) \\
&= \mathbb{E}_{\tau_{-T:k} \sim p(\tau_{-T:k} \mid \theta)} \left[\mathcal{T}(s_{k+1} \mid s_k, a_k) \sum_{t=-T}^{k+1} f(s_t) \right] \\
&= \mathbb{E}_{\tau_{-T:k-1} \sim p(\tau_{-T:k-1} \mid \theta)} \left[\sum_{s_k, a_k} \mathcal{T}(s_k \mid s_{k-1}, a_{k-1}) \pi(a_k \mid s_k) \mathcal{T}(s_{k+1} \mid s_k, a_k) \sum_{t=-T}^{k+1} f(s_t) \right] \\
&= \sum_{s_k, a_k} \mathcal{T}(s_{k+1} \mid s_k, a_k) \pi(a_k \mid s_k) \mathbb{E}_{\tau_{-T:k-1} \sim p(\tau_{-T:k-1} \mid \theta)} \left[\mathcal{T}(s_k \mid s_{k-1}, a_{k-1}) \left(f(s_{k+1}) + \sum_{t=-T}^{k} f(s_t) \right) \right] \\
&= \sum_{s_k, a_k} \mathcal{T}(s_{k+1} \mid s_k, a_k) \pi(a_k \mid s_k) \left(\mathbb{E}_{\tau_{-T:k-1} \sim p(\tau_{-T:k-1} \mid \theta)} \left[\mathcal{T}(s_k \mid s_{k-1}, a_{k-1}) f(s_{k+1}) \right] + G_k(s_k) \right) \\
&= \sum_{s_k, a_k} \mathcal{T}(s_{k+1} \mid s_k, a_k) \pi(a_k \mid s_k) \left(\sum_{s_{-T:k-1}, a_{-T:k-1}} \left(p(\tau_{-T:k-1} \mid \theta) \mathcal{T}(s_k \mid s_{k-1}, a_{k-1}) f(s_{k+1}) \right) + G_k(s_k) \right)
\end{aligned}
$$

$$= \sum_{s_k, a_k} \mathcal{T}(s_{k+1} \mid s_k, a_k)\pi(a_k \mid s_k) \left(\sum_{s_{-T:k-1}, a_{-T:k-1}} \left(p(\tau_{-T:k-1} \mid \theta) p(s_k \mid \tau_{-T:k-1}) f(s_{k+1}) \right) + G_k(s_k) \right)$$

$$= \sum_{s_k, a_k} \mathcal{T}(s_{k+1} \mid s_k, a_k)\pi(a_k \mid s_k) \left(\sum_{s_{-T:k-1}, a_{-T:k-1}} \left(p(s_k, \tau_{-T:k-1} \mid \theta) f(s_{k+1}) \right) + G_k(s_k) \right)$$

$$= \sum_{s_k, a_k} \mathcal{T}(s_{k+1} \mid s_k, a_k)\pi(a_k \mid s_k) \left(p(s_k \mid \theta) f(s_{k+1}) + G_k(s_k) \right)$$

For the base case, note that

$$\begin{aligned}
&G_{-T+1}(s_{-T+1}) \\
&= \mathbb{E}_{\tau_{-T:-T} \sim p(\tau_{-T:-T} \mid \theta)} \left[\mathcal{T}(s_{-T+1} \mid s_{-T}, a_{-T}) \sum_{t=-T}^{-T+1} f(s_t) \right] \\
&= \sum_{s_{-T}, a_{-T}} p(s_{-T} \mid \theta) p(a_{-T} \mid s_{-T}) \mathcal{T}(s_{-T+1} \mid s_{-T}, a_{-T}) \sum_{t=-T}^{-T+1} f(s_t) \\
&= \sum_{s_{-T}, a_{-T}} \mathcal{T}(s_{-T+1} \mid s_{-T}, a_{-T}) p(a_{-T} \mid s_{-T}) \left(p(s_{-T} \mid \theta) f(s_{-T+1}) + p(s_{-T} \mid \theta) f(s_{-T}) \right)
\end{aligned}$$

So, for the base case we can simply set $G_{-T}(s_{-T}) = p(s_{-T} \mid \theta) f(s_{-T})$.

B

We provide the algorithm to sample from the posterior distribution $p(\theta \mid s_0)$.

Algorithm 1 MCMC sampling from the one state IRL posterior

Require: MDP $\mathcal{M}$, prior $p(\theta)$, step size δ

$\theta \leftarrow$ random sample$(p(\theta))$
$\pi, V =$ soft value iteration$(\mathcal{M}, \theta)$
$p \leftarrow p(s_0 \mid \theta) p(\theta)$ ▷ $p(s_0 \mid \theta)$ is computed with Equation 6 using policy π.
repeat
  $\theta' \leftarrow$ random sample$(\mathcal{N}(\theta, \delta))$
  $\pi', V' =$ soft value iteration$(\mathcal{M}, \theta')$ ▷ The value function is initialized with V.
  $p' \leftarrow p(s_0 \mid \theta') p(\theta')$
  if random sample(Unif$(0,1)$) $\leq \min(1, \frac{p'}{p})$ **then**
    $\theta \leftarrow \theta'$; $V \leftarrow V'$
  end if
  append θ to the list of samples
until have generated the desired number of samples