

Figure 7: On (left) and off (right) policy 40x40 driving sim returns.

## A Proofs

**Lemma 2.** *The set  $\Upsilon$  is non-empty, convex, compact, and polyhedral.*

*Proof.* The set is non-empty because  $u_e^* \in \Upsilon$ . It is convex and polyhedral from its construction and is compact because it is bounded ( $1^\top u \leq 1/(1-\gamma)$ ) and closed.  $\square$

**Lemma 1.** *If  $\mathcal{D}$  is generated by a deterministic policy  $\pi \in \Pi_D$ . Then*

$$u \in \Upsilon \quad \Leftrightarrow \quad (u = u^\pi, \exists \pi \in \Pi_R(\mathcal{D})).$$

*Proof.* We prove the two implications individually.

*Case 1:* Prove  $u \in \Upsilon \Rightarrow \exists \pi \in \Pi_R(\mathcal{D}), u = u^\pi$ ,

Given a  $u \in \Upsilon$ , we have that  $u^{\pi_u} = u$  since  $u \in \mathcal{U}$  and  $p_0 > 0$ . Recall that  $p_0 > 0$  guarantees that

$$\sum_{a \in \mathcal{A}} u(s, a) \geq p_0 > 0, \quad \forall s \in \mathcal{S}.$$

We now show that  $\pi_u \in \Pi_R(\mathcal{D})$ . The constraint on  $u \in \Upsilon$  combined with  $u > 0$  implies that for each  $(\cdot, s, \cdot) \in \mathcal{D}$ :

$$\begin{aligned} \sum_{a \in \mathcal{A}: a \neq \pi_e(s)} u(s, a) &= 0, \\ \Downarrow \\ u(s, a) &= 0, \quad \forall a \in \mathcal{A}, a \neq \pi_e(s) \\ \Downarrow \\ \pi^u(s, a) &= 0, \quad \forall a \in \mathcal{A}, a \neq \pi_e(s). \end{aligned}$$

Therefore  $\pi^u \in \Pi_R(\mathcal{D})$ , which proves the implication.

*Case 2:* Prove  $\pi \in \Pi_R(\mathcal{D}) \Rightarrow u_\pi \in \Upsilon$

This follows because

$$c^\top u^\pi = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} c(s, a) u(s, a) = 0,$$

since  $c(s, a) > 0 \Rightarrow \pi(s, a) = 0 \Rightarrow u(s, a) = 0$ .  $\square$

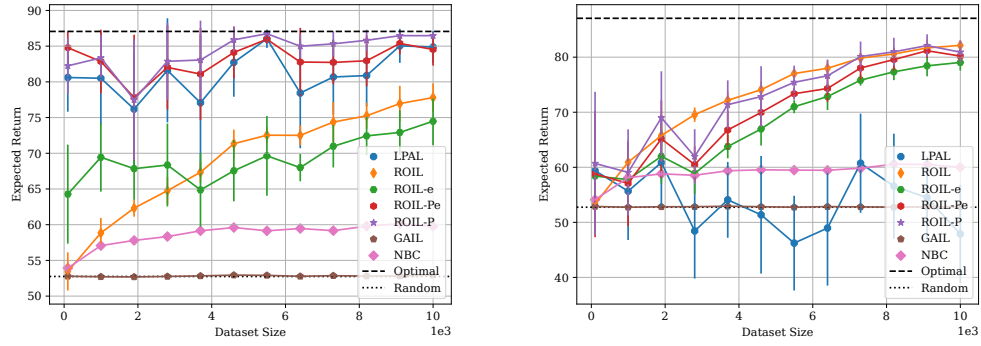


Figure 8: On (left) and off (right) policy 40x40 gridworld returns with more methods from Section 3

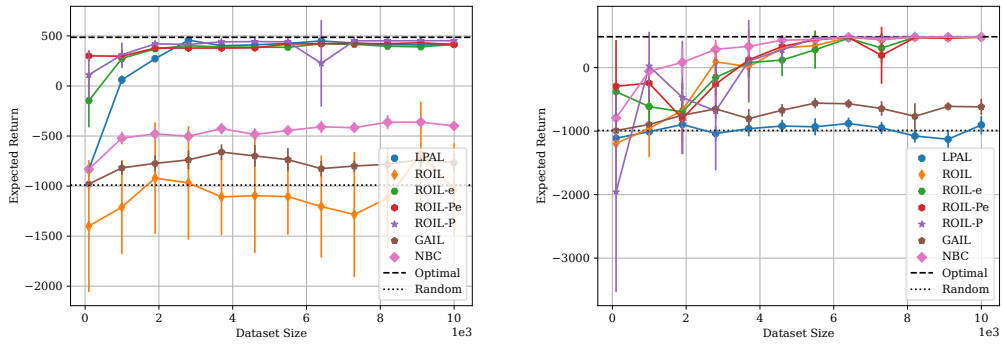


Figure 9: On (left) and off (right) policy 40x40 driving sim returns with more methods from Section 3

## B Experimental Results Revisited

We focus on describing the grid world environment; the driving environment is implemented similarly. The states of a grid world are integers from 0 to  $N \times N$  where  $N$  is the number of rows in the grid world, we assume we have square grid worlds where the number of columns and rows are the same. The actions an actor can take are up, down, left, and right. And the reward for a particular  $(s, a)$  tuple is dictated by some reward vector  $r \in \mathbb{R}^{SA} = \Phi^\top w$  for some  $w \in \mathcal{W}$ . To solve all methods that rely on an LP, such as all ROIL variants described here, we rely on Gurobi ([Gurobi Optimization, LLC, 2023](#)). The feature vector for the grid world environment was the color of the current cell which was randomized for each seed, out of 4 total colors. For the driving simulator, there were also 4 features, current row, current column, whether the actor crashed in the current state, or whether the actor went off-road. When constructing the dataset  $\mathcal{D}$  for our off-policy experiments, the policy  $\pi_e$  decides the actions for each state. However, the state transitions came from a uniformly random policy  $\pi_b$ . More precisely  $\mathcal{D} = (s_t, \pi_e(s_t))_{t=0}^D$  where  $s_{t+1} \sim P(s_t, \pi_b(s_t))$  and  $s_0 \sim p_0$ .

### B.1 ROIL Variants

We focus on ROIL and ROIL-P. ROIL uses the linear constraint on  $\mathcal{U}$  resulting in an outer minimization over  $\Upsilon$  instead of  $\mathcal{U}$  in (11) with the same inner maximization. ROIL-P uses pruning which is a technique to reduce the number of inner maximizations needed in the final minimization in (11), we use  $\hat{u}_e$  to prune away extreme points (or sampled points in cases where  $\mathcal{W}$  leads to non-enumerable extreme points) where  $\hat{u}_e^\top \text{ext}(\mathcal{W}) \leq \tau$ .

For convenience, let  $\psi \in \mathbb{R}^d = \max_{v \in \Upsilon} v^\top \Phi w \quad \forall w \in \text{ext}(\mathcal{W})$  where  $d$  is the number of inner maximizations you wish to solve, whether that be all extreme points of the L infinity norm ball or some sub-sample of a different norm ball. ROIL-P chooses  $\tau$  to be the 90% quantile of  $\psi$ . While it may seem ignorant to ignore some of these extreme points, it is a helpful heuristic that can help calibrate our pessimism as described in Section 3. This extension as well as many others is what we find new and exciting about ROIL, however, we have too few pages to describe all such tweaks. For those interested, in Figures 5 and 7 we use 100 samples from the  $\mathbb{R}_{1+}$  for ROIL-P.

We also implemented ROIL-e and ROIL-Pe which optimize over  $\hat{\Upsilon}_e$  as in Section 3.2, this led to comparable on policy performance to LPAL and ROIL-P while maintaining stable gains in expected return with new state action pairs off policy as seen in Figures 8 and 9. However, due to how cluttered the graphs got, we decided to leave those as an extension and show off the two methods that were most representative of our formulation as a whole, putting the choice of robustness into the user’s hands.

### B.2 GAIL

While the performance of most methods seems about on par with what one would expect it may seem that our implementation of GAIL ([Ho and Ermon, 2016](#)) has some bugs. We assure the reader and refer the reader to our [code](#) that we did indeed implement GAIL correctly and it does converge in smaller domains when  $\mathcal{D}$  is on policy as the expectation in Algorithm 1 of ([Ho and Ermon, 2016](#)) relies on  $\hat{u}_e$  being close to  $u_e^*$ . Due to computation constraints, and to be fair to the other methods we compared with, we had to limit the computation time per method. GAIL, being based on policy gradient methods, is rather sample inefficient concerning environment interactions ([Ho and Ermon, 2016](#), Section 7) thus it incurs extra computation collecting on policy samples from  $\pi_\theta$ . We would also like to note that GAIL must bootstrap itself with a new random policy parameter  $\theta$  each run as the other methods we tested against it also did not get to keep any information from previous runs.