17-740 Spring 2025

Algorithmic Foundations of Interactive Learning

Lecture 19: Efficient Algorithms for Inverse Reinforcement Learning

Lecturer : Gokul Swamy Scribe: Lujing Zhang, Jim Wang, Ryan Schuerkamp

19.1 Outline

We will cover the following topics today:

1. What makes inverse RL sample-inefficient?
   A: Repeatedly solving a global exploration (RL) problem.
2. Are best responses required for solving the IRL game?
   A: Actually, competing with the expert is "all you need".
3. What algorithms can we use in our new reduction?
   A: A wide variety of sample-efficient "local search" algorithms.

19.2 Recap: IRL as Game-Solving

Recall that we can frame *inverse reinforcement learning* (IRL) as solving a zero sum game:

$$\max_{\pi \in \Pi} \min_{f \in \mathcal{R}} J(\pi, f) - J(\pi_E, f), \tag{19.1}$$

where $J(\pi, f) = \mathbb{E}_{\xi \sim \pi}\left[\sum_h^H f(s_h, a_h)\right]$. Intuitively, when solving the above game, we're trying to find a policy π that isn't too distinguishable from π_E under any reward function $f \in \mathcal{R}$. More formally, an ϵ-approximate Nash Equilibrium of the above game guarantees that $J(\pi, r) - J(\pi_E, r) \leq \mathcal{O}(\epsilon H)$ if $r \in \mathcal{R}$. We discussed two strategies for solving this game:

	Dual	Primal
Policy Update	**BR**: RL	**NR**: GD
Reward Update	**NR**: GD	**NR**: GD

An example of a *dual* algorithm is MaxEnt IRL, while an example of a *primal* algorithm is GAIL. Observe that in computing the *best response* over policies, dual algorithms need to solve the global exploration problem inherent in RL. It is more subtle but the same is true

for primal algorithms as well. This is because the adversary could keep playing the same f and to satisfy the no-regret property, the policy player would need to compute π^* eventually. More formally, *any* no-regret algorithm can be used to compute a best response. And, as we discussed in earlier lectures, algorithms like *follow the regularized leader* use a best response oracle to implement a no-regret algorithm. Thus, NR and BR aren't that different.

19.3 What makes inverse RL sample-inefficient?

Repeatedly needing to solve a global exploration problem over the entire state-space is what makes inverse RL inefficient. We ground this point in a tree-structured MDP:

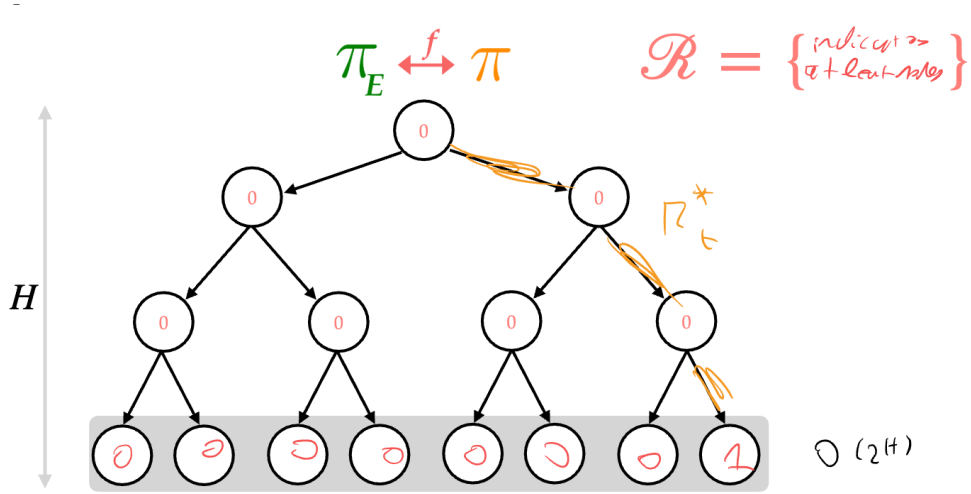


Figure 19.1: Consider a binary tree structured MDP of depth H where the reward function class $\mathcal{R}$ is indicator functions at "leaf nodes" at time H (1 node out of 2^H nodes at time H has reward 1, all other nodes 0). Thus, at each round of IRL, after the adversary picks some f, we might have to explore all leaf nodes before we find any reward, an amount of interaction that grows *exponentially* with the task horizon – $\mathsf{exp}(H)$. Then, at the next iteration, we need to solve a similar problem again after the adversary shifts the reward. In some sense, *we're using an RL hammer in a game of adversarial whack-a-mole.*

Beyond being a worst-case constructions, applications like auto-regressive language generation have this tree-structured property. Intuitively, this is strange because *we've reduced the "easier" problem of IL to repeatedly solving the "harder" problem of RL.*

19.4 Are BRs required for IRL?

Intuitively, we waste interaction in IRL by searching over policies that are dissimilar to the expert's. We sketch an example of this below:

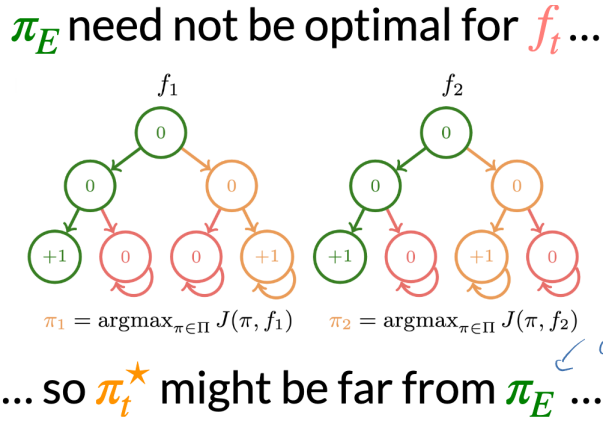


Figure 19.2: Consider some adversarially chosen reward f_t. The optimal policy for this reward, π_t^*, might be far from π_E. This means it can't be the policy we want, as we're just trying to compete with the expert for inverse RL.

This leads to our *key idea: saving interaction by competing with* π_E*, not* π_t^*. This allows us to prune large parts of the state / policy space. We now make this intuition more formal.

19.4.1 Reducing IRL to Expert-Competitive RL

We now derive a more informed reduction for IRL that allows us to eliminate the complexity of global exploration. First, we introduce the notion of an *expert-relative regret oracle,* i.e. an algorithm that merely competes with the expert on average:

Definition 1 (ERROr$\{\text{Reg}_\pi(T)\}$) *A policy-selection algorithm* $\mathbb{A}_\pi$ *satisfies the* $Reg_\pi(T)$ *expert-relative regret guarantee if given any sequence of reward functions* $f_{1:T}$*, it produces a sequence of policies* $\pi_{t+1} = \mathbb{A}_\pi(f_{1:t})$ *such that*

$$\sum_{t=1}^{T} J(\pi_E, f_t) - J(\pi_t, f_t) \leq Reg_\pi(T). \tag{19.2}$$

We never need to compute a best response to an f_t *– doing so is sufficient but not necessary!*

For completeness, we also define a no-regret reward selection algorithm.

Definition 2 *$\mathbb{A}_f$ is a no-regret reward selection algorithm if when given a sequence of policies $\pi_{1:t}$, it produces iterates $f_{1:t} = \mathbb{A}_f(\pi_{1:t})$ such that*

$$\sum_{t=1}^{T} J(\pi_t, f_t) - J(\pi_E, f_t) - \min_{f^* \in \mathcal{R}} \sum_{t=1}^{T} J(\pi_t, f^*) - J(\pi_E, f^*) \leq H Reg_f(T),$$

with $\lim_{T\to\infty} \frac{Reg_f(T)}{T} = 0$.

We now show that with just these two oracles, we can solve the IRL game:

Proof: Consider using an ERROr algorithm to select policies and a no-regret reward selection algorithm to select discriminators. Then, we can expand the average performance difference over T rounds of the algorithm as

$$\begin{aligned}
J(\pi_E, r) - J(\bar{\pi}, r) &= \frac{1}{T}\sum_{t=1}^{T} J(\pi_E, r) - J(\pi_t, r) \quad \text{(Linearity of expectation)} \\
&\leq \max_{f^* \in \mathcal{F}_r} \frac{1}{T}\sum_{t=1}^{T} J(\pi_E, f^*) - J(\pi_t, f^*) \quad \text{(Reward realizability)} \\
&\leq \frac{1}{T}\sum_{t=1}^{T} J(\pi_E, f_t) - J(\pi_t, f_t) + \frac{\text{Reg}_f(T)}{T} H \quad \text{(Defn. of regret)} \\
&\leq \frac{\text{Reg}_\pi(T)}{T} + \frac{\text{Reg}_f(T)}{T} H \quad \text{(Defn. of ERROr)}
\end{aligned}$$

■

In short, the above proof tells us that *competing with the expert is all you need for IRL!*

19.5 What algorithms can we use in our new reduction

A natural question at this point might be as to what an efficient algorithm that actually satisfies the ERROr property is. It turns out we've covered quite a few over the semester: PSDP, HyQ, Agnostic SysID. Basically, anything derived from a variation of the performance difference lemma works! We now step through the use of PSDP in IRL in detail.

19.5.1 Expert-Competitive RL via PSDP

Key Idea: Reset to states from the demonstrations to reduce unnecessary exploration in IRL!

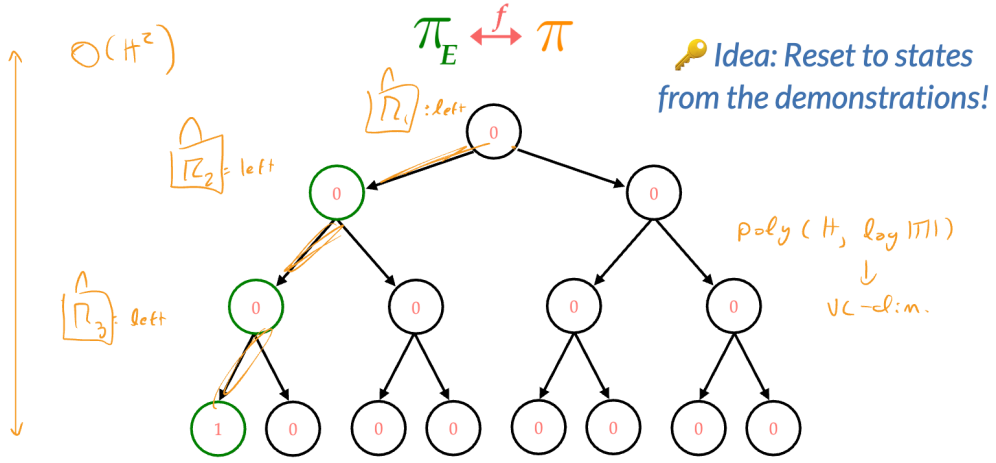


Figure 19.3: Let us consider our tree-structured MDP again with the additional information that the expert takes the leftward, green path. We proceed via backwards in time induction. First, at $h = H - 1$, we have to figure out whether we go left or right from the expert's s_{H-1}. We would pick to go left to get reward 1, which we then lock in to compute π_3. We then back up a timestep to $h = H - 2$ and again pick a single-step decision conditioned on the future policies we've computed. If we go left, we'd go left again and get reward 1. If we go right, we'd then go left and get reward 0. So, we'd choose to go left, giving us π_2. We can do this all the way up the tree. Observe that we only need $\mathcal{O}(H)$ interaction at each step of the algorithm and there are H steps, which gives us an overall interaction complexity of $\mathcal{O}(H^2)$, an *exponential* speedup.

In words, we're reducing interaction complexity by performing a *local search*. For longer horizon problems where we'd like a single, stationary policy, we can instead pick a random timestep rather than doing backwards-in-time induction and achieve similar guarantees. Practically, this algorithm corresponds to doing resets to states from the expert demonstrations during the inner loop of IRL, rather than to the true start-state distribution. Of course, it is hard to do resets in the real world, so one can instead learn a model (fitting it on a *hybrid* distribution of learner and expert data) and do resets inside the model. Intuitively, fitting a model on just expert data can make it too optimistic, while fitting a model on just learner data can make it too pessimistic. Fitting on a hybrid mixture provably balances optimism and pessimism without computational intractability.

19.5.2 PSDP Performance Guarantees

For completeness, we now state the formal performance guarantee for PSDP:

Lemma 1 (PSDP Performance Guarantee) *Assume that at each time-step $h \in [H]$, we perform policy optimization up to ε-optimality on some baseline distribution μ:*

$$\mathbb{E}_{s_h \sim \mu}\left[\mathbb{E}_{a \sim \pi'_h(s_h)}\left[Q_t^{\pi_{h+1:H}}(s_h, a)\right] - \mathbb{E}_{a \sim \pi_h(s_h)}\left[Q_t^{\pi_{h+1:H}}(s_h, a)\right]\right] \leq \varepsilon H$$

Then, the performance of the learned policy satisfies:

$$J(\pi'_{1:H}, f_t) - J(\pi_{1:H}, f_t) \leq \mathcal{O}((\varepsilon + \mathbb{D}_{TV}(\mu, \rho_{\pi'}) \cdot H^2)$$

where μ is the reset distribution used for PSDP, $\rho_{\pi'}$ is the occupancy measure of the reference policy π', and $\mathrm{TV}(\mu, \rho_{\pi'})$ is the total variation distance between the two distributions.

Proof: We proceed via the Performance Difference Lemma (PDL):

$$\begin{aligned}
J(\pi'_{1:H}, f_t) - J(\pi_{1:H}, f_t) &= \sum_h^H \mathbb{E}_{s_h, a_h \sim \pi'_{1:h}}\left[Q^{\pi_{h+1:H}}(s_h, a_h) - \mathbb{E}_{a \sim \pi_h(s_h)}\left[Q^{\pi_{h+1:H}}(s_h, a_h)\right]\right] \\
&\leq \sum_h^H \mathbb{E}_{s_h, a_h \sim \mu}\left[Q^{\pi_{h+1:H}}(s_h, a_h) - \mathbb{E}_{a \sim \pi_h(s_h)} Q^{\pi_{h+1:H}}(s_h, a_h)\right] \\
&\quad + H \cdot \mathbb{D}_{TV}(\mu_h, \rho_h^{\pi'}) \quad \text{(by Hölder's inequality)} \\
&\leq \sum_h^H (\varepsilon + \mathbb{D}_{TV}(\mu_h, \rho_h^{\pi'})) \cdot 2(H - h) \\
&\leq (\varepsilon + \mathbb{D}_{TV}(\mu, \rho_{\pi'})) \cdot H^2
\end{aligned}$$

■

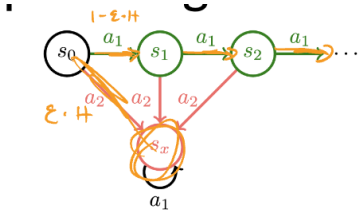
In the language we used above, this result means that $Reg_\pi(T) \leq (\varepsilon + \mathbb{D}_{TV}(\mu, \rho_{\pi'})) \cdot H^2 T$. While we don't discuss this in detail in the lecture, observe that we can use reset distributions other than the expert demonstrations and still have strong performance guarantees so long as μ (the reset distribution) is close to ρ_E (the expert's state distribution). This is a natural opportunity to use plentiful suboptimal data to speed up exploration – see the linked paper on the course website for more information.

19.5.3 Compounding Errors in PSDP

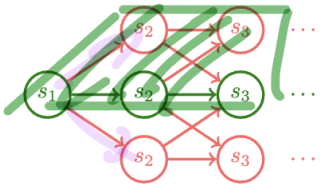
A natural question after seeing the $\mathcal{O}(H^2)$ term in the above bound might be whether the above bound is tight. Put differently, does PSDP suffer from compounding errors just like

behavior cloning does in some situations? Recall that the whole reason we're using an interactive algorithm for imitation is to avoid compounding errors.

It turns out that in the worst case (e.g. cliff-walking, irrecoverable problems), this is true. However, on less harsh problems, PDSP can recover from some errors, since it knows some information about what it's going to do in the future. We sketch an example of both below.



Unavoidable in general on cliff-like (irrecoverable) problems.



Interaction can still help us figure out which mistakes compound.

Figure 19.4: **Top:** We consider a cliff-walking problem where a single mistake can doom the learner for the rest of the horizon. Assume the policies for timesteps $[1, H]$ have been computed and stay on the top of the cliff. At $h = 0$, the learner picks a policy that makes a mistake with probability ϵH. While they have an average error of ϵ, they have a performance difference from the expert that scales with $\mathcal{O}(\epsilon H^2)$. Furthermore, an interactive algorithm like PSDP wouldn't be able to fix such issues any better than behavioral cloning as during the computation of policies $[1, H]$, we pretend the learner is going to start from the top of the cliff rather than having already fallen off. **Bottom:** We now consider a recoverable problem, where the expert takes the middle path but either of the top two rows give accessible reward. Similar to before, assume we perfectly imitate the expert for steps $[1, H]$ by going straight. At $h = 0$, we give the learner the ability to choose between going up or down. Observe that PSDP would know to choose to go up because it observes the consequences of its future actions, while BC would be unable to tie-break correctly.

Thus, while in the worst case, resetting to expert states can introduce compounding errors, it is less likely to do so on problems with recoverability. See papers linked on course website for a more formal statement of the above. In practice, one can also start by only looking at expert state distributions, but then gradually interpolate with or anneal towards the ground-truth initial state distribution ρ_0 to reduce compounding errors.

19.5.4 Additional Algorithms

While we do not discuss this idea in detail, one can also run HyQ as part of the above reduction and have similar guarantees. In practice, this corresponds to running some off-policy RL algorithm (e.g., SAC) with a hybrid replay buffer (e.g., RLPD). Intuitively, this helps speed up policy search as we're picking a reward function that makes the expert look good, so training on expert data is likely to provide positive signal.