Inverse Reward Design - Learning and Exploration

Huiwen You, huiwen.you@mail.mcgill.ca
Zilun Peng, zilun.peng@mail.mcgill.ca

1 Introduction

As a common practice in reinforcement learning, a model designer encodes its belief about the environment into the reward function of an autonomous agents; however it is always difficult to design a reward function that actually captures all intentions of the designer. Inevitably, agents may encounter new scenarios that are not covered by the training environments when using the trained model to perform tasks. As a result, the agent often performs very badly on those new states in the testing environment.

Suppose the model designer uses a set of weights to approximate the reward. What are the optimal weights? Hadfield-Menell et al.[1] formulate this problem as Inverse Reward Design (IRD). IRD is the problem of inferring the true objective based on the designed reward in the training environment. To solve IRD, Hadfield-Menell et al.[1] introduce approximate methods to infer optimal weights of the reward function, then use the solution to plan risk-averse policy in the testing environments. Based on empirical results, they show that this approach can help alleviate negative side effects of misspecified reward functions and mitigate reward hacking.

We would like to highlight the difference between IRD and Inverse Reinforcement Learning (IRL) [2]. IRL takes model designer's labeled data as its input and outputs optimal parameters of the reward function. The difference is that IRD takes parameters provided by the model designer.

We consider this is quite an exciting discovery because if the proposed solution works as expected, it means that agents are able to perform as what the model designer expects when encountering a new environment. As a result, agents are able to avoid bad behaviors. In this project, we are inspired to implement the proposed IRD solution and use it to reproduce the lavaland experiment included in the Hadfield-Menell et al's paper [1] to test its effectiveness. Furthermore, we are curious to explore the generalizability of this approach in other experimental settings.

Few required reinforcement learning techniques and terminologies are listed in Section 2. We addressed details of the IRD algorithm in the Section 3. The set of experiments and corresponding results can be found in Section 4. We conclude the project findings Section 5.

2 Preliminary

2.1 Markov Decision Process(MDP)

In this project, we refer to the training environment as training MDP. It denotes the environments anticipated by human-designers. Whereas, we refer to the testing environment as testing MDP. It denotes the environments encountered by robot agents, and it contains unobserved states.

2.2 Policy Iteration

To solve an MDP problem, the aim is to compute an optimal MDP policy and its value function. The optimal policy guides agents through its way to the goal state. In our project, policy iteration is used as the baseline in the experiments described in Section 4, and it is also used to compute expected feature vector for the trajectories sampled from rewarded behavior.

2.3 reward function

In our project, the reward function is assumed to be a linear function of weights and trajectory's feature vector.

2.4 rewarded behavior

In IRD, Hadfield-Menell et al.[1] introduced "rewarded behavior" terminology. From our understanding, it stands for a distribution over trajectories that depends on the proxy reward and the MDP that the agent behaves in.

3 Model

The overall IRD solution can be summarized by the following two steps.

- Step 1: Compute IRD posterior
- Step 2: Risk averse planning

3.1 IRD posterior

The IRD posterior gives a probability distribution on various candidate weights guessed by agents. According to the original paper[1], what is specified by the system designer is actually proxy reward's weight rather than proxy reward. Here we call "proxy reward's weight" as "proxy weight" for simplicity. Similarly, we refer to the "true reward's weight" as "true weight" or "optimal weight" for simplicity.

Given the proxy weight $\tilde{w}$, we want to compute $P(w^*|\tilde{w})$. Here w^* is the true weight of the reward function. We know that $P(w^*|\tilde{w}) = \frac{P(\tilde{w}|w^*)P(w^*)}{P(\tilde{w})}$ by Bayes rule. In order to compute $P(\tilde{w}|w^*)$, Hadfield-Menell et al.[1] learnt from Ziebart et al[5] and used maximum entropy trajectory distribution[5] to model $P(\tilde{w}|w^*)$, written as

$$P(\tilde{w}|w^*) \propto exp\big(\beta\,\mathbb{E}\big[w^{*\intercal}\phi(\xi)\big]\big) \propto exp\big(\beta w^{*\intercal}\,\mathbb{E}\big[\phi(\xi)\big]\big)$$

ξ is the trajectory generated by using the proxy reward $\tilde{w}$. Considering the only uncertainty in the above equation is associated with ξ, we can pull out w^*. $\phi(\xi)$ is the feature vector of ξ, represented by the number of times each state is visited. If $\tilde{w}$ is close to w^*, $\phi(\xi)$ would have large values on those states which are also frequently visited by w^*; as a result, $w^{*\intercal}\,\mathbb{E}\big[\phi(\xi)\big]$ would be large as expected. $P(\tilde{w})$ is difficult to compute, because it involves integrating over all possible w^*. Hadfield-Menell et al[1] approximates this term by following Sunnaker et al's approach[3], where $P(\tilde{w})$ is approximated by $Z(\hat{w})$.

$$Z(\hat{w}) \coloneqq exp(w^\intercal\,\mathbb{E}[\phi]) + \sum_{i=0}^{N-1} exp(\beta w^\intercal\,\mathbb{E}[\phi_i]).$$

In the above equation, $\mathbb{E}[\phi]$ is associated with w and $\mathbb{E}[\phi_i]$ is associated with w_i. $\mathbb{E}\big[\phi(\xi)\big] = \sum_\xi P(\xi)\phi(\xi)$. Computing $\sum_\xi P(\xi)\phi(\xi)$ involves summing over all possible trajectories that the agent could take given $\tilde{w}$ and this is clearly not feasible. Since $\sum_\xi P(\xi)\phi(\xi)$ is equivalent to $\sum_s P(s)\phi(s)$, we could compute $\mathbb{E}\big[\phi(\xi)\big]$ by using the dynamic programming approach introduced in [5]. The complete algorithm for estimating the IRD posterior is not given in the original paper and will be given in Algorithm 1. Note that we assume the prior distribution is uniform.

Algorithm 1 Estimating $P(w^*|\tilde{w})$

```
Input $w_1 ... w_n, \tilde{w}$
Output $P(w_1|\tilde{w}) ... P(w_n|\tilde{w})$
1: $E\tilde{\phi} \leftarrow computeExpectedPhi(\tilde{w})$
2: for $w_i \in w_1 ... w_n$ do
3:     $E\phi_i \leftarrow computeExpectedPhi(w_i)$
4: for $w_i \in w_1 ... w_n$ do
5:     $denominator \leftarrow exp(w_i^\intercal E\phi_i) + \sum_{j \neq i} exp(w_i^\intercal E\phi_j)$
6:     $P(w_i|\tilde{w}) \leftarrow \frac{exp(w_i^\intercal E\tilde{\phi})}{denominator}$
7: return $P(w_1|\tilde{w}) ... P(w_n|\tilde{w})$
```

3.2 Risk Averse Planning

By taking samples from $P(w^*|\tilde{w})$, we obtain weights that best match to system designer's belief about the environment, which is encoded in $\tilde{w}$. However, samples will have high variances on states that are not present in the training environment. Suppose the testing environment contains state j which is never seen in the training environment. This means that $\phi(\xi)_j$ will always be 0. Therefore, w_j could be any number and it will not affect $P(w|\tilde{w})$. This causes high variances in samples obtained from $P(w^*|\tilde{w})$. Suppose the system designer wants the robot to avoid grass, stay on the dirt and reach the terminal. This will be represented by the first 3 elements of $\tilde{w}$. Samples from $P(w^*|\tilde{w})$ will carry this belief. If the fourth term of w represents lava, there will be large variances on this term across all samples. Hadfield-Menell et al[1] exploit this observation and let agent to plan in a risk averse way, so that the robot will still behave as what the system designer expects at test time. In the meanwhile, robot will try to avoid hitting states with large variances (in this case, lava). Given a set of weights w_i, sampled from the posterior. Being risk averse means that the agent finds ξ such that the reward obtained on the worst w_i is maximized. More concretely, this means that

$$\xi^* = \underset{\xi}{\operatorname{argmax}} \min_{w \in w_i} w^\intercal \phi(\xi)$$

Hadfield-Menell et al[1] follow Syed et al[4]'s linear programming approach to solve for ξ^*. The complete linear program is not given by Hadfield-Menell et al [1] and based on our understanding, the linear program is summarize as below:

$$\begin{aligned} \text{maximize} \quad & B \\ \text{subject to} \quad & B \le \sum_{s,a} w^i_{s,a} x_{s,a}, \forall i \\ & \sum_a x_{s,a} = \alpha_s + \gamma \sum_{s',a} x_{s',a}, \forall s \\ & 0 \le x_{s,a} \forall s, a \end{aligned}$$

$w^i_{s,a}$ is the reward obtained by performing action a at state s if we use w_i. α_s is the initial probability of state s. γ is the discount factor. Note that we intend to learn a deterministic policy, so there is no transition probabilities in the above linear program.

4 Experiment and Result

To prove the correctness of the IRD implementation and compare the effectiveness between IRD and original baseline learning algorithm. We decided to experiment on the toy example included in the original paper. Here we refer to this example as the lavaland experiment.

MDP is used in this project to define the environments with which agents are going to interact. The set of states is gridworld cells; the set of actions is "up","down", "left" and "right", encoded as integer numbers from 0 to 3 respectively; the transition matrix reflects the probability of agents going from one state s to the other s' by taking action a; the reward function is a mapping from states to rewards. In our problem, we are given the proxy weight w̃, and our objective is to infer the true reward weight based on w̃ and the observed environment.

There are two MDPs involved, namely training MDP and testing MDP[1]. The training MDP contains three kinds of terrains, dirt, grass and gold. Imagine that designer's intention is to encourage routing on dirt rather than grass and find the gold as fast as possible. Therefore, we randomly and uniformly sampled the proxy weight reflecting the designers intention. Contrarily, the testing MDP contains a new terrain type called lava, which is not anticipated by designer during the training time. Proxy weight for lava is generated randomly. If the agent does not run IRD, it trusts the proxy reward given by the designer and follows the instruction. If the randomly generated weight for lava is greater than dirt and grass, agents would run across lava cells to reach the gold. We used the policy learned from policy iteration as the baseline. The discount factor is set to 0.9 and threshold to terminate the linear program is set to 0.01 throughout our project. If agent uses IRD to plan its trajectory, it should be able to avoid traversing the lava. We performed experiment with the following 4 different settings.

4.1 Experiment with intentional proxy weight

Considering that IRD is proposed based on the fact that the proxy weights are able to reflect system designers intentions, so we first run experiments with randomly sampled intentional proxy weights that encourage agents to route on dirt rather than grass and to find the gold as fast as possible. To be more specific, we randomly sampled dirt weight from 1 to 6, grass weight from -10 to 0, gold/terminal weight from 6 to 10. For the unknown state, we give the agent the freedom to choose arbitrary garbage value from -100 to 100 to mimic the real world scenario. With this setting, we received the following result.

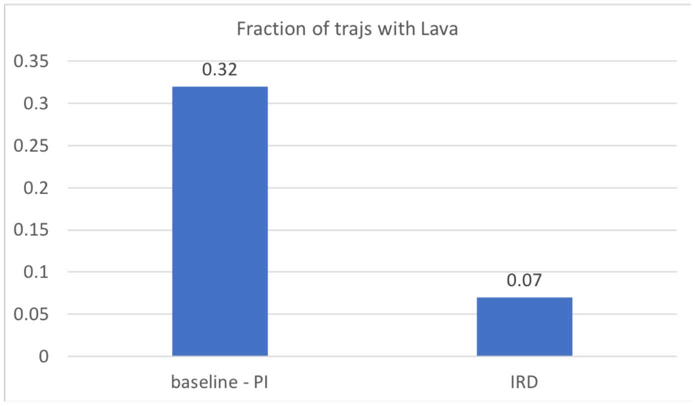

Figure 1: Performances between IRD and the baseline algorithm if sampling proxy weights from specified ranges

As it is shown in figure 1, we have received promising result. Agent with IRD is able to successfully avoid 93% of the occurrence of lava; whereas agent with policy iteration could not get away with 32% of the experiments.

4.2 Experiment with random proxy weight

In this experiment, we would like to evaluate how much does IRD model's reliability depend on intentional proxy weights. In theory, the performance should not be affected too much because the objective of IRD model is to plan a risk averse trajectory for the agent in order to avoid unobserved state. This means that the agent may not attempt to approach the gold in this setting, but it should still avoid hitting the lava, because sampled weights have large variances on lava's proxy weight. To conduct this experiment, we randomly assigned proxy weight for every land type. The result is shown in Figure 2.

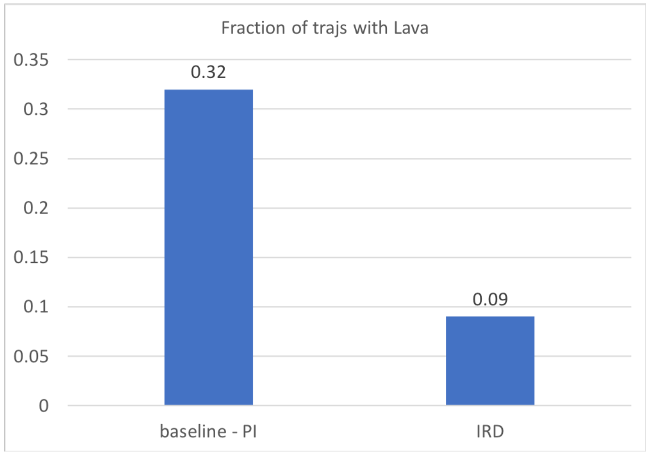

Figure 2: Performances between IRD and the baseline algorithm if sampling proxy weights completely at random

It is exciting to notice that the experiment result proves our understanding of the IRD model. This result is actually very close to the previous one which utilized intentional proxy weight. This is expected because variances on lava does not depend on values of proxy weights.

4.3 Performance comparison between IRD and baseline algorithm

Table 1 shows performance differences between IRD and the baseline algorithm. Hadfield-Menell et al [1] claim that rewards should be compared in the same scale in order to achieve good performances. This is done by subtracting $w^\intercal \phi(\xi)$ from $\tilde{w}^\intercal \phi(\xi)$ in constraints of the linear program. Results in Table 1 verify their claim. The reason is intuitive because sampled weights are randomly generated, so some weights may influence the linear program to be biased toward selecting certain actions.

Method	fraction of ξ with lava	number of w approximating $P(w^*\|\tilde{w})$	number of samples from $P(w^*\|\tilde{w})$	comparing to training feature counts
IRD	0.225	100	7	no
IRD	0.095	50	7	no
IRD	0.04	100	7	yes
IRD	0.055	50	7	yes
baseline	0.295	100	7	n/a
baseline	0.32	50	7	n/a
baseline	0.34	100	7	n/a
baseline	0.29	50	7	n/a

Table 1: Performances between IRD and the baseline algorithm under different experimental settings

4.4 Testing risk averse planner

All the experiments in previous sections use randomly generated proxy rewards. This means that we should only expect the agent to avoid the lava but not traveling to the gold. In this experiment, we manually select inputs to our risk averse planner, i.e. the linear program introduced in Section 3. We can think of those manually selected inputs as samples from the posterior, given our sample space is perfect. This means that there are some weights that are very similar to model designer's proxy weights, i.e. $\tilde{w}$ in the sample space.

Our selected inputs to the planner are $\begin{bmatrix} 0.1 \\ -10 \\ 10 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0.1 \\ -10 \\ 10 \\ -5 \end{bmatrix}$, $\begin{bmatrix} 0.1 \\ -10 \\ 10 \\ 10 \end{bmatrix}$, and $\begin{bmatrix} 0.1 \\ -10 \\ 10 \\ -10 \end{bmatrix}$. The four elements are proxy weights on dirt, grass, gold and lava respectively. This means that we want the robot to avoid grass and find the gold. There are a lot of variances on the proxy weight for lava, because lava is never seen in the training environment, so all possible values for lava's proxy weight could be sampled from the posterior. By using these values as inputs to our planner, we are able to get a trajectory that is exactly the same as what is shown in Figure 4 in the Appendix.

Obviously, it is not possible to get samples that are as perfect as we designed in this experiment. We showed that the risk averse planner is able to avoid choosing states with high variances through this experiment.

5 Conclusion

5.1 Correctness

In terms of model correctness, we can conclude from the experiments that IRD model is able to mitigate the risk from misspecified objectives. From our experiments, it shows that IRD is capable of helping agents avoid 94% of the unobserved new states. Besides, the IRD model with its best setting is 5 times less likely to traverse on the new state than baseline. Also notice that agents are still able to avoid 90% of the unintended consequences without providing intentional proxy weight.

5.2 Efficiency

The running time of our implemented IRD model per experiment is stably around 1 hour. Note that by "one experiment" here we mean one instance of experiment we conducted in section 4. If agent is only provided with one proxy weight, one training MDP and one testing MDP, our model will cost around 40 minutes to produce the planned trajectory. The reason why our experiment can run as fast as 1 hour to deal with 100 different proxy weights is that we set aside 50 or 100 candidate weights as a "candidate weight set" and this set is used for each one of the 100 model computations. Considering that this set is fixed, we pre-computed the related terms and then directly apply the resulted values to the subsequent model computation. Therefore, only the first proxy weight model computation is a bit time-consuming.

5.3 Stability

Numerical stability of IRD depends on $\mathbb{E}[\phi(\xi)]$. If $\mathbb{E}[\phi(\xi)]$ is too large or too small, this will make the likelihood either extremely large or extremely small because our likelihood term is defined as $exp\big(\beta w^{*\intercal}\,\mathbb{E}\big[\phi(\xi)\big]\big)$. To solve this problem, we only learn deterministic policies from proxy weights. In this way, $\mathbb{E}[\phi(\xi)]$ becomes more controlled.

The stability of IRD's solution depends on the quality of the sample space. If no weights in the sample space are similar to the proxy weights, our posterior distribution will become more uniform. Samples from the posterior will not be a good indicator of the proxy weights, so solution is not good. This problem should be addressed by the prior, which we did not exploit in this project.

5.4 Generalizability

Even though our IRD model produces promising result in our toy lavaland experiment, it does not generalize well with respect to the following aspects. First, risk-averse planning avoids good risk. As we learn from the IRD model, it's objective is to avoid unintended behaviour no matter it is good or not. For example, if the new state in our lavaland example is gold rather than lava, agents are actually encouraged to traverse through these new states according to designer's intention. Unfortunately, the current IRD model is not able to reactive differently for different circumstances. Second, problem scope does not scale. Current IRD model cannot handle large scale problem because number of variables to the linear program is number of states times number of actions. Third, reward function is restricted to be linear

6 Challenges

We have encountered some challenges in this project. The first challenge is to understand the model, especially the formulas to compute posterior and risk averse planning. We struggled to understand some notations in the formula, for example $\tilde{w}$ is used to refer to both the proxy reward and it weight in the original paper. When it comes to understand the formula, it is at first very confusing to differentiate which $\tilde{w}$ this notation refers to. The other challenge we met is to learn the risk averse planning algorithm. Neither of us has prior knowledge or experience with risk-averse planning method, and the original paper did not provide their implementation detail on this topic; therefore, we learned linear programming approach from Syed et al[4] and implemented this approach based on our understanding.

7 Project Contribution

Zilun Peng: contributed effort in risk-averse planning, code implementation, report and presentation.
Huiwen You: contributed effort in code implementation, report and slides.

References

[1] Pieter Abbeel Stuart Russell Anca Dragan Dylan Hadfield-Menell Smitha Milli. *Inverse Reward Design*. URL: https://arxiv.org/abs/1711.02827.

[2] Andrew Y Ng and Stuart J Russell. *Algorithms for Inverse Reinforcement Learning*. URL: http://www.cs.cmu.edu/~jeanoh/16-785/papers/ng-icml2000-irl.pdf.

[3] Busetto Alberto Giovanni Numminen Elina Corander Jukka Foll Matthieu Sunnaker Mikael and Christophe Dessimoz. *Approximate Bayesian Computation*. URL: http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002803.

[4] Bowling Michael Syed Umar and Robert E. Schapire. *Apprenticeship Learning Using Linear Programming*. URL: https://www.cs.princeton.edu/~schapire/papers/SyedBowlingSchapireICML2008.pdf.

[5] Maas Andrew L Bagnell J Andrew Ziebart Brian D and Anind K. Dey. *Maximum Entropy Inverse Reinforcement Learning*. URL: https://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf.

Appendices

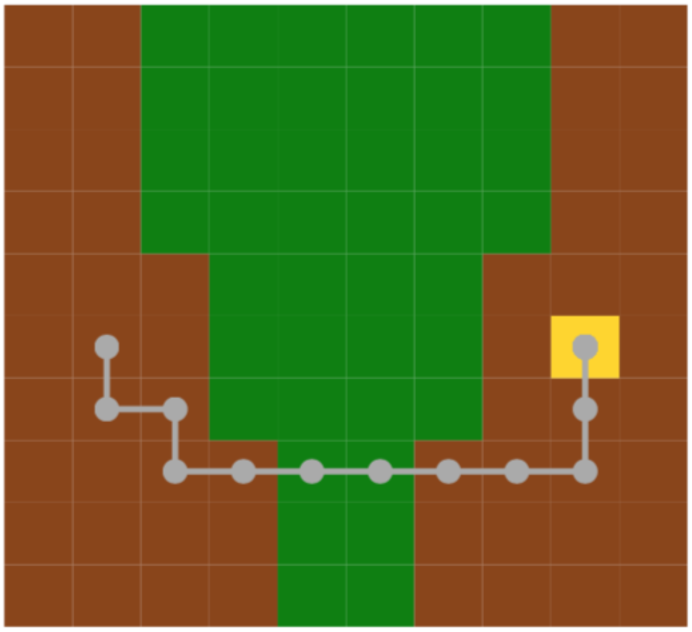

Figure 3: training MDP. image is taken from [1]

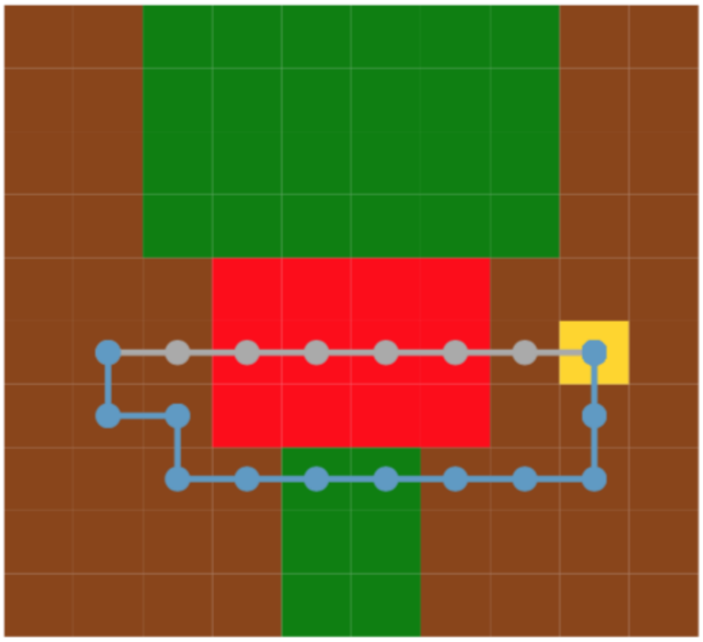

Figure 4: testing MDP. image is taken from [1]