ECE 232E (Spring 2018)

Project 3: Reinforcement Learning and

Inverse Reinforcement Learning

Group Members:

Yunhao Ba (705032832)

Shuangyu Li (805035359)

Jingchi Ma (705027270)

Chenguang Yuan (005030313)

Table of Contents

1. Introduction

This project was developed on Python to study and implement reinforcement learning & inverse reinforcement learning. In part 2 and 3, we learned the optimal policy of an agent navigating in a 2-D environment and implemented the value iteration algorithm to learn the optimal policy. In part 4, we explored the application of IRL in the context of apprenticeship learning.

2. Reinforcement Learning (RL)

The two main objects in Reinforcement Learning are: Agent and Environment. In this project, we will learn the optimal policy of a single agent navigating in a 2-D environment.

2.1 Environment

In this project, we assume that the environment of the agent is modeled by a Markov Decision Process (MDP). In a MDP, agents occupy a state of the environment and perform actions to change the state they are in. After taking an action, they are given some representation of the new state and some reward value associated with the new state.

2.1.1 State Space

In this project, we consider the state space to be a 2-D square grid with 100 states. The 2-D square grid along with the numbering of the states is shown in the figure below:

	0	1	2	3	4	5	6	7	8	9
0	0.0	10.0	20.0	30.0	40.0	50.0	60.0	70.0	80.0	90.0
1	1.0	11.0	21.0	31.0	41.0	51.0	61.0	71.0	81.0	91.0
2	2.0	12.0	22.0	32.0	42.0	52.0	62.0	72.0	82.0	92.0
3	3.0	13.0	23.0	33.0	43.0	53.0	63.0	73.0	83.0	93.0
4	4.0	14.0	24.0	34.0	44.0	54.0	64.0	74.0	84.0	94.0
5	5.0	15.0	25.0	35.0	45.0	55.0	65.0	75.0	85.0	95.0
6	6.0	16.0	26.0	36.0	46.0	56.0	66.0	76.0	86.0	96.0
7	7.0	17.0	27.0	37.0	47.0	57.0	67.0	77.0	87.0	97.0
8	8.0	18.0	28.0	38.0	48.0	58.0	68.0	78.0	88.0	98.0
9	9.0	19.0	29.0	39.0	49.0	59.0	69.0	79.0	89.0	99.0

Fig: 2-D square grid with state numbering

2.1.2 Action Set

In this project, we will consider the action set(A) to contain the 4 following actions: move right, move left, move up, and move down.

2.1.3 Transition Probabilities

In this project, we define the transition probabilities in the following manner:

1. If state *s'* and *s* are not neighboring states in the 2-D grid, then

$$\mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a) = 0$$

s' and *s* are neighbors in the 2-D grid if you can move to *s'* from *s* by taking an action *a* from the action set A. We will consider a state *s* to be a neighbor of itself. For example, from figure 1 we can observe that states 1 and 11 are neighbors (we can transition from 1 to 11 by moving right), but states 1 and 12 are not neighbors.

2. Each action corresponds to a movement in the intended direction with probability 1 - *w*, but has a probability of *w* of moving in a random direction instead due to wind.

2.1.4 Reward Function

To simplify the project, we will assume that the reward function is independent of the current state (*s*) and the action that you take at the current state (*a*). To be specific, reward function only depends on the state that you transition to (*s'*). With this simplification, we have

$$\mathcal{R}^{a}_{ss'} = R(s')$$

In this project, we will learn the optimal policy of an agent for two different reward functions:

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Fig: Reward function 1

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	-100.0	-100.0	-100.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	-100.0	0.0	-100.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	-100.0	0.0	-100.0	-100.0	-100.0	0.0
4	0.0	0.0	0.0	0.0	-100.0	0.0	0.0	0.0	-100.0	0.0
5	0.0	0.0	0.0	0.0	-100.0	0.0	0.0	0.0	-100.0	0.0
6	0.0	0.0	0.0	0.0	-100.0	0.0	0.0	0.0	-100.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	-100.0	-100.0	-100.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	-100.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0

Fig: Reward function 2

Question 1:

For visualization purpose, we generated heat maps of Reward function 1 and Reward function 2, and we can observe that the Reward function 1 is symmetric according to the diagonal line. Heat map figures are shown below.

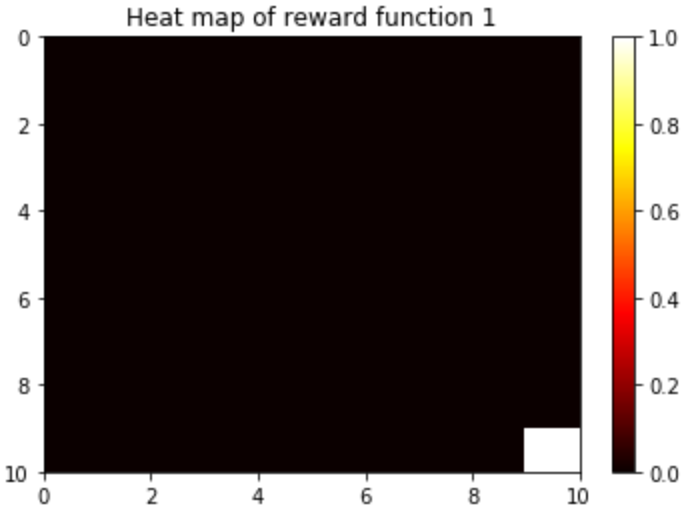

Fig: Heat map of reward function 1

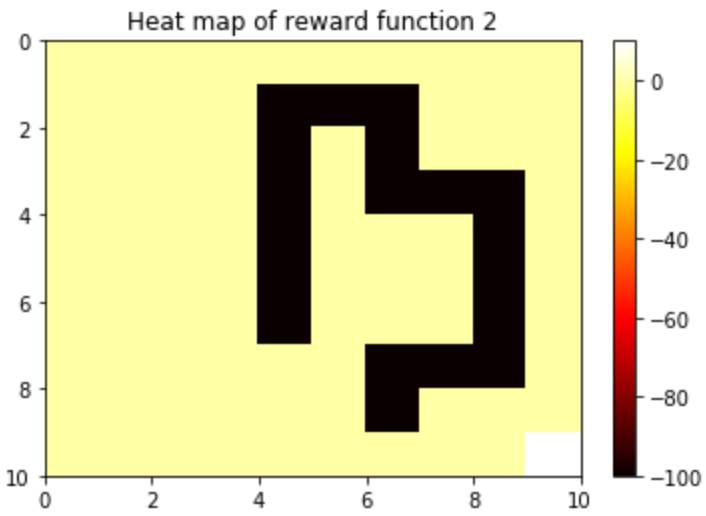

Fig: Heat map of reward function 2

3. Optimal Policy Learning Using RL Algorithms

In this part of the project, we use reinforcement learning (RL) algorithm to find the optimal policy. The main steps in RL algorithm are:

1. Find optimal state-value or action-value
2. Use the optimal state-value or action-value to determine the deterministic optimal policy

There are a couple of RL algorithms, but we will use the Value iteration algorithm since it was discussed in detail in the lecture.

Question 2:

In this part, we implement the value iteration algorithm according to the provided pseudocode, and the obtained state values are shown in the figure below. Note: When updating the state values, we choose to update the state values after each iteration, i.e. we update the state values when we finish calculation of all the 100 states. The state 99 tries to go back itself consistently and constraint to 4.7 and all other state values are varies depends on their distance to the state99.

	0	1	2	3	4	5	6	7	8	9
0	0.044	0.065	0.091	0.125	0.168	0.223	0.292	0.380	0.491	0.610
1	0.065	0.088	0.122	0.165	0.219	0.289	0.378	0.491	0.633	0.788
2	0.091	0.122	0.165	0.219	0.289	0.378	0.491	0.636	0.818	1.019
3	0.125	0.165	0.219	0.289	0.378	0.491	0.636	0.820	1.052	1.315
4	0.168	0.219	0.289	0.378	0.491	0.636	0.820	1.054	1.352	1.695
5	0.223	0.289	0.378	0.491	0.636	0.820	1.054	1.353	1.733	2.182
6	0.292	0.378	0.491	0.636	0.820	1.054	1.354	1.735	2.220	2.807
7	0.380	0.491	0.636	0.820	1.054	1.353	1.735	2.220	2.839	3.608
8	0.491	0.633	0.818	1.052	1.352	1.733	2.220	2.839	3.629	4.635
9	0.610	0.788	1.019	1.315	1.695	2.182	2.807	3.608	4.635	4.702

Fig: The optimal values of states using reward function 1

Question 3:

We then generated a heat map of the optimal state values across the 2-D grid as shown below. Highest value appears at bottom right corner and lowest shows on the top left and behave symmetry along the diagonal elements.

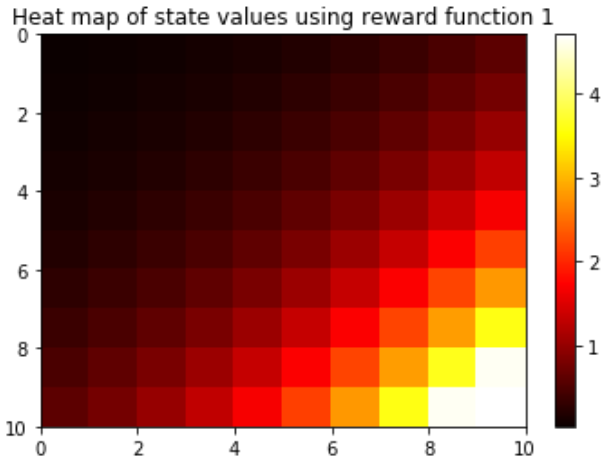

Fig: Heat map of state values using reward function 1

Question 4:

As observed, the state value distribution is similar to the shape of Gaussian distribution with the center at the bottom-right corner (the state with reward = 1). When a state is away from the reward state (State 99), the its state value will drop according to its distance to the reward state. Besides, all the state values are positive for reward function 1, since there is no state with negative reward values. This kind of distribution is intuitive. Since if a state is far from the reward state, it will take more steps for the agent to get to the reward state from that state. Due to the existence of the discount factor, the state value of that state will decrease according to the number of steps, which results in this reward function in the Gaussian shape.

Question 5:

The optimal policy is shown in the figure below, and we find that this optimal policy of the agent does match our intuition. Our intuition is that the agent will go to reward state (State 99) as quickly as possible, and in the figure below, the agent will go to the State 99 either by moving down or by moving right, which is in accordance with our intuition. Besides, since we can observe that the state values for reward function 1 is symmetric as shown in the Question 3 and Question 4. So, the optimal policy is not unique in this case. The agent can either goes down or goes right to reach State 99 (the scores for going down and going right are the same). Due to the implementation, we get the policy as shown below, however, this policy is just one of the optimal policies.

We think it is possible for the agent to compute the optimal action to take at each state by observing the optimal values of its neighboring states in this environment. First, in this environment, the agent can only move to the neighbor states in each step, and due to the existence of the discount factor. The optimal values of the neighboring states will dominate the total reward, and thus the agent will try to avoid the neighboring states with small state values

and head to the state with large state value. This conclusion is true for this simple reward function 1 under this simple environment, and may not be true for other environment with different reward functions.

	0	1	2	3	4	5	6	7	8	9
0	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇩	⇩
1	⇩	⇩	⇨	⇨	⇨	⇨	⇩	⇩	⇩	⇩
2	⇩	⇩	⇨	⇨	⇨	⇩	⇩	⇩	⇩	⇩
3	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩
4	⇩	⇩	⇩	⇨	⇨	⇩	⇩	⇩	⇩	⇩
5	⇩	⇩	⇨	⇨	⇨	⇩	⇩	⇩	⇩	⇩
6	⇩	⇨	⇨	⇨	⇨	⇨	⇩	⇩	⇩	⇩
7	⇩	⇨	⇨	⇨	⇨	⇨	⇨	⇩	⇩	⇩
8	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇩	⇩
9	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇩

Fig: The optimal policy of the agent using reward function 1

Question 6:

The state values using reward function 2 is shown below.

	0	1	2	3	4	5	6	7	8	9
0	0.647	0.791	0.821	0.525	-2.386	-4.237	-1.923	1.128	1.591	2.035
1	0.828	1.018	1.062	-1.879	-6.755	-8.684	-6.373	-1.298	1.925	2.607
2	1.061	1.313	1.446	-1.635	-6.758	-13.917	-9.653	-5.515	-0.135	3.355
3	1.358	1.689	1.944	-1.243	-6.339	-7.983	-7.947	-9.434	-1.918	4.387
4	1.734	2.168	2.586	-0.736	-5.847	-3.258	-3.241	-7.434	1.715	9.160

5	2.211	2.778	3.413	-0.038	-5.114	-0.553	-0.488	-2.984	6.583	15.354
6	2.816	3.553	4.479	3.024	2.480	2.880	-0.466	-4.911	12.688	23.296
7	3.584	4.539	5.793	7.288	6.719	7.241	0.931	12.366	21.159	33.483
8	4.558	5.795	7.397	9.439	12.008	12.889	17.097	23.014	33.778	46.529
9	5.727	7.316	9.388	12.045	15.452	19.824	25.498	36.158	46.583	47.311

Fig: The optimal values of states using reward function 2

Question 7:

Heat map of the optimal state values using reward function 2 is shown below.

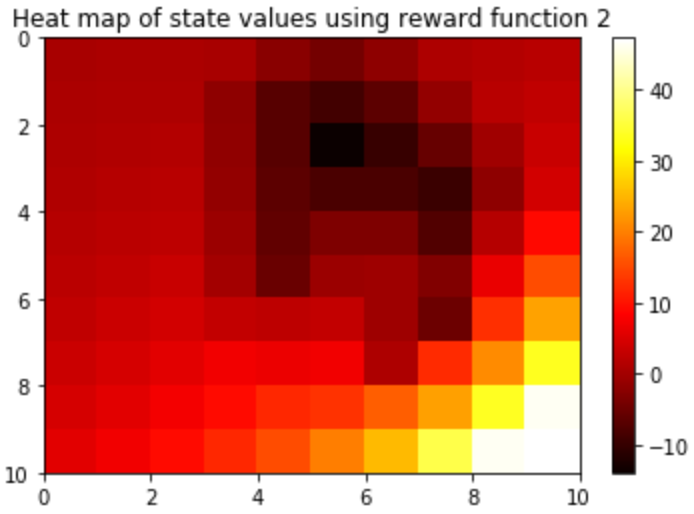

Fig: Heat map of state values using reward function 2

Question 8:

From the figure of question 7, we could observe the distribution is somewhat similar to the state value generated by reward function 1 due to the positive reward at the bottom right corner. The corner part is still in gaussian distribution and decrease with longer distance from the corner. However, the state values near the negative reward states are significantly affected. Especially the most “inside” state, which is almost surrounded by the states with negative rewards and needs much more steps to reach the corner reward. The states inside negatives rewards are

mostly negative and form a nearly negative Gaussian shape with center at State 52. The negative part is not exactly Gasussain, because State 52 is not fully surrounded by the states with negative rewards (Sate 53 is with zero reward).

Question 9:

From the figure above, we find that the optimal policy of the agent does match our intuition. The optimal policy follows the intuition quite well. Following the arrows, all the states will point to the the bottom right corner at last except the bottom right states itself. Since the problem defines the environment in a way that once a states pointing out of boundary, the action probability will point to itself instead. The bottom right state is the only positive state in the environment, therefore it also makes sense to return back to that state. What's more, all the optimal actions of negative rewards states are pointing out of the negative area, which means the agent prefers to take a detour to avoid stronger negative reward, which is also in accordance with our intuition. In general, the agent tends to avoid the state with negative values to reach the bottom right corner under the optimal policy.

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇦	⇦	⇨	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇩
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇩
4	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩	⇨	⇩
5	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇦	⇨	⇩
7	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇩	⇩	⇩
8	⇨	⇨	⇨	⇩	⇩	⇩	⇩	⇩	⇩	⇩

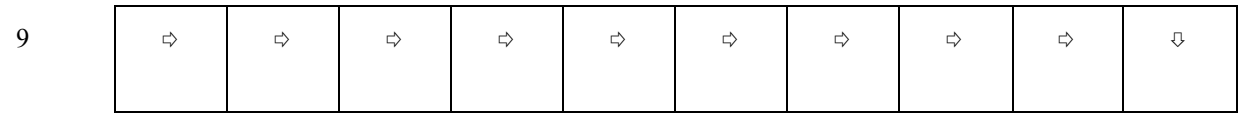

Fig: The optimal policy of the agent using reward function 2

4. Inverse Reinforcement Learning (IRL)

Inverse Reinforcement learning (IRL) is the task of learning an expert's reward function by observing the optimal behavior of the expert. The motivation for IRL comes from apprenticeship learning. In apprenticeship learning, the goal of the agent is to learn a policy by observing the behavior of an expert. This task can be accomplished in two ways:

1. Learn the policy directly from expert behavior
2. Learn the expert's reward function and use it to generate the optimal policy

The second way is preferred because the reward function provides a much more parsimonious description of behavior. Reward function, rather than the policy, is the most succinct, robust, and transferable definition of the task. Therefore, extracting the reward function of an expert would help design more robust agents.

In this part of the project, we will use IRL algorithm to extract the reward function. We will use the optimal policy computed in the previous section as the expert behavior and use the algorithm to extract the reward function of the expert. Then, we will use the extracted reward function to compute the optimal policy of the agent. We will compare the optimal policy of the agent to the optimal policy of the expert and use some similarity metric between the two to measure the performance of the IRL algorithm.

4.1 IRL Algorithm

For finite state spaces, there are a couple of IRL algorithms for extracting the reward function:

1. Linear Programming (LP) formulation
2. Maximum Entropy formulation

Since we covered LP formulation in the lecture and it is the simplest IRL algorithm, so we will use the LP formulation in this project. The LP formulation of the IRL is given by the following equation:

$$\begin{aligned} \underset{\mathbf{R},t_i,u_i}{\text{maximize}} \quad & \textstyle\sum_{i=1}^{|\mathcal{S}|}(t_i - \lambda u_i) \\ \text{subject to} \quad & [(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i))(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R}] \geq t_i, \;\; \forall a \in \mathcal{A} \setminus a_1, \forall i \\ & (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma\mathbf{P}_{a_1})^{-1}\mathbf{R} \succeq 0, \;\; \forall a \in \mathcal{A} \setminus a_1 \\ & -\mathbf{u} \preceq \mathbf{R} \preceq \mathbf{u} \\ & |\mathbf{R}_i| \leq R_{max}, \;\; i = 1, 2, \cdots, |\mathcal{S}| \end{aligned} \tag{1}$$

In the LP given by the above equation, **R** is the reward vector (**R**(i) = **R**(si)), **P**a is the transition probability matrix, λ is the adjustable penalty coecient, and ti's and ui's are the extra optimization variables (please note that **u**(i) = u_i). Use the maximum absolute value of the ground truth reward as R_{max}. For the ease of implementation, we can recast the LP in the above equation into an equivalent form given by the following equation using block matrices:

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{D}\mathbf{x} \preceq 0, \quad \forall a \in \mathcal{A} \setminus a_1 \end{aligned} \tag{2}$$

Question 10:

Here, instead of using the Dx ≤ 0 form, we use the Dx ≤ b form for simplicity as mentioned in the office hour. The dimension of D is 1000×300, and x and c are vectors with 300 elements. b is a vector with 1000 elements (800 zeros and 200 Rmax). The optimization problem can be formulated as follows.

$$maximize \begin{bmatrix} \mathbf{0}_{1\times n} & \mathbf{1}_{1\times n} & -\boldsymbol{\lambda}_{1\times n} \end{bmatrix} \times \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{t} \\ \boldsymbol{u} \end{bmatrix}$$

$$
\begin{bmatrix}
-\left((\boldsymbol{P}_{a_1}(1)-\boldsymbol{P}_{a_2}(1))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(1)\right)^{-1}\right) & \boldsymbol{e_1} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(1)-\boldsymbol{P}_{a_3}(1))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(1)\right)^{-1}\right) & \boldsymbol{e_1} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(1)-\boldsymbol{P}_{a_4}(1))\left(\boldsymbol{I}-\boldsymbol{\gamma}\boldsymbol{P}_{a_1}(1)\right)^{-1}\right) & \boldsymbol{e_1} & \boldsymbol{0}\\
\vdots & \vdots & \vdots\\
-\left((\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_{a_2}(i))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(i)\right)^{-1}\right) & \boldsymbol{e_i} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_{a_3}(i))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(i)\right)^{-1}\right) & \boldsymbol{e_i} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_{a_4}(i))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(i)\right)^{-1}\right) & \boldsymbol{e_i} & \boldsymbol{0}\\
\vdots & \vdots & \vdots\\
-\left((\boldsymbol{P}_{a_1}(n)-\boldsymbol{P}_{a_2}(n))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(n)\right)^{-1}\right) & \boldsymbol{e_n} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(n)-\boldsymbol{P}_{a_3}(n))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(n)\right)^{-1}\right) & \boldsymbol{e_n} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(n)-\boldsymbol{P}_{a_4}(n))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(n)\right)^{-1}\right) & \boldsymbol{e_n} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(1)-\boldsymbol{P}_{a_2}(1))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(1)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(1)-\boldsymbol{P}_{a_3}(1))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(1)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(1)-\boldsymbol{P}_{a_4}(1))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(1)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
\vdots & \vdots & \vdots\\
-\left((\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_{a_2}(i))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(i)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_{a_3}(i))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(i)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_{a_4}(i))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(i)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
\vdots & \vdots & \vdots\\
-\left((\boldsymbol{P}_{a_1}(n)-\boldsymbol{P}_{a_2}(n))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(n)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(n)-\boldsymbol{P}_{a_3}(n))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(n)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
-\left((\boldsymbol{P}_{a_1}(n)-\boldsymbol{P}_{a_4}(n))\left(\boldsymbol{I}-\gamma\boldsymbol{P}_{a_1}(n)\right)^{-1}\right) & \boldsymbol{0} & \boldsymbol{0}\\
-\boldsymbol{I}_{n\times n} & \boldsymbol{0} & -\boldsymbol{I}_{n\times n}\\
-\boldsymbol{I}_{n\times n} & \boldsymbol{0} & \boldsymbol{I}_{n\times n}\\
\boldsymbol{I}_{n\times n} & \boldsymbol{0} & \boldsymbol{0}\\
-\boldsymbol{I}_{n\times n} & \boldsymbol{0} & \boldsymbol{0}
\end{bmatrix}
\begin{bmatrix}\boldsymbol{R}\\ \boldsymbol{t}\\ \boldsymbol{u}\end{bmatrix} \leq R_{max}\begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{1}_{2n\times 1}\end{bmatrix}
$$

where $\mathbf{P_{a1}(i)}$ represents the transition matrix under the optimal action for each state i, $\mathbf{P_{a2}}$, $\mathbf{P_{a3}}$, $\mathbf{P_{a4}}$ denotes transition matrix with actions other than the optimal one, and $\boldsymbol{ei}$ = (0, 0 … 0, 1, 0 … 0) with ith element being one and all other elements being zeros (ith row of the identity matrix).

4.2 Performance Measure

In this project, we use a very simple measure to evaluate the performance of the IRL algorithm. Before we state the performance measure, let's introduce some notation:

$O_A(s)$: Optimal action of the agent at state s

$O_E(s)$: Optimal action of the expert at state s

$$m(s) = \begin{cases} 1, O_A(s) = O_E(s) \\ 0, \text{else} \end{cases}$$

Then with the above notation, accuracy is given by equation :

$$Accuracy = \frac{\sum_{s \in \mathcal{S}} m(s)}{|\mathcal{S}|} \tag{3}$$

Since we are using the optimal policy found in the previous section as the expert behavior, so we will use the optimal policy found in the previous section to fill the $O_E(s)$ values. Please note that these values will be different depending on whether we used Reward Function 1 or Reward Function 2 to create the environment.

To compute $O_A(s)$, we will solve the linear program given by equation 2 to extract the reward function of the expert. For solving the linear program we will use the LP solver in python. Then, we will use the extracted reward function to compute the optimal policy of the agent using the value iteration algorithm we implemented in the previous section. The optimal policy of the agent found in this manner will be used to fill the $O_A(s)$ values. Please note that these values will depend on the adjustable penalty coefficient λ . We will tune λ to maximize the accuracy.

Question 11:

We use the previously extracted optimal policy from reward function 1 as the $O_E(s)$, and the accuracy plot is shown below.

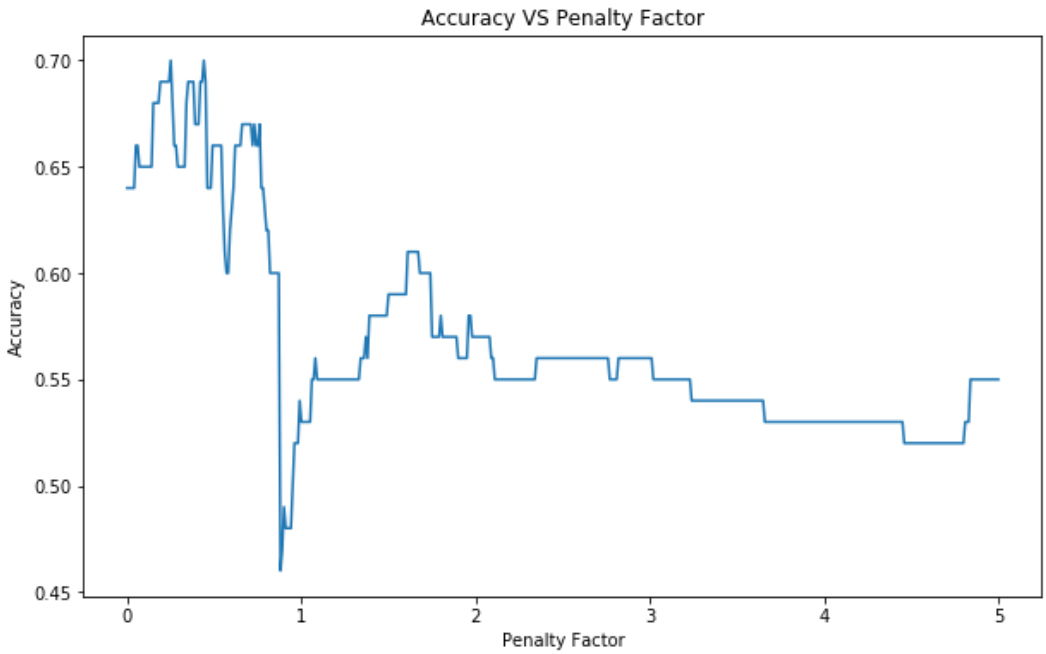

Fig: Penalty Factor λ vs Accuracy

Question 12:

Penalty factor λ with best accuracy: λmax=0.25

Corresponding accuracy: 0.7

Question 13:

The ground truth reward and the recovered reward by IRL are shown below.

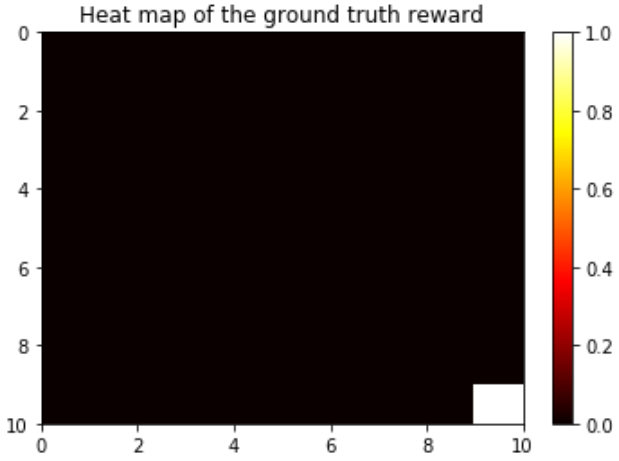

Fig: Heat map of the ground truth reward

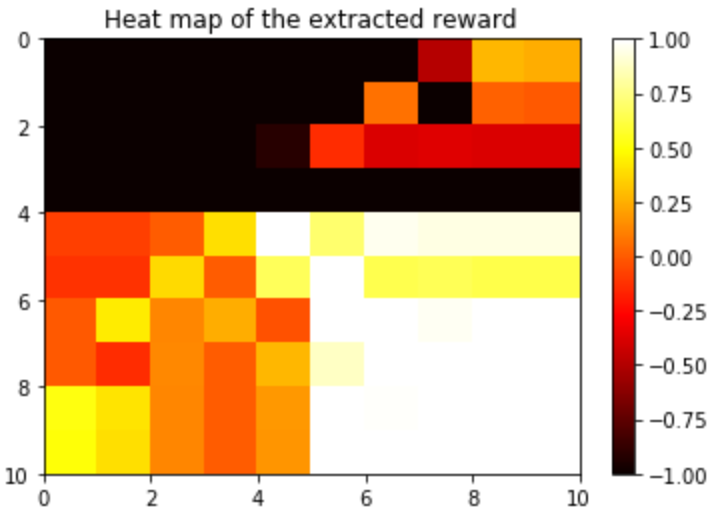

Fig: Heat map of the extracted reward

Question 14:

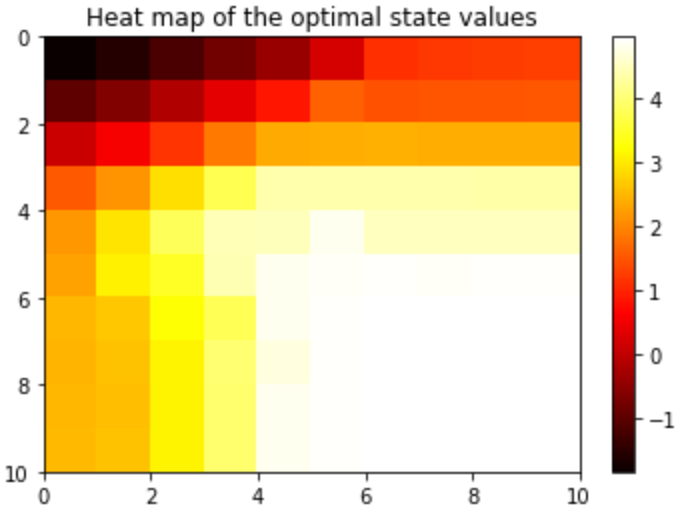

Fig: Heat map of the optimal state values using the extracted reward function

Question 15:

The heat map of question 14 looks similar to the heat map of question 3 to some extent (both in Gaussian shape). The reward function from IRL is still nearly symmetry and holds the trends in general that the up left corner is the smallest one and the bottom right corner is the largest one, and the distance to the bottom corner decides the state values, which is similar to the original optimal state values. However, there are some minor differences as well. The heat map of question 14 has some negative values but the values in the heat map of question 3 are all positive. The scale of these two maps are the same in general. The other main difference is the "spread out" of the reward to larger area. We could barely to see the difference of the large white

area. The state values become flatter after the IRL. It may caused by the limitation of the used IRL algorithm which is only a linear model [1]. Therefore, the rewards are constrained to have a more smooth gradient as shown in the figure above.

Question 16:

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇨	⇧	⇦
1	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩
2	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩
3	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩
4	⇨	⇨	⇨	⇨	⇨	⇩	⇩	⇦	⇩	⇩
5	⇨	⇨	⇨	⇨	⇨	⇩	⇩	⇩	⇩	⇩
6	⇨	⇨	⇨	⇨	⇨	⇨	⇩	⇩	⇩	⇩
7	⇩	⇨	⇨	⇨	⇨	⇨	⇨	⇩	⇩	⇩
8	⇦	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇦	⇦
9	⇧	⇨	⇨	⇨	⇨	⇨	⇨	⇧	⇧	⇦

Fig: The optimal policy of the agent using the extracted reward function

Question 17:

In the general the policy are in the same trend with some minor differences to the original policy. The agent will head to the states in the bottom right region from the upper left region for both policies. However, there are some states on the boundaries with actions pointing of the stage for the IRL policy, which is different from the original policy. It is may caused by the local maximum generated from the LP model. The agent under the policy will try to loop back to itself for a higher reward by the moving out actions. What's more, in the high value area, the policy flows finally goes to State 78 and 88 because of they pointing to each other and both holding a relative high state value. Therefore, it will form a loop here and increase the state value to a large value.

Question 18:

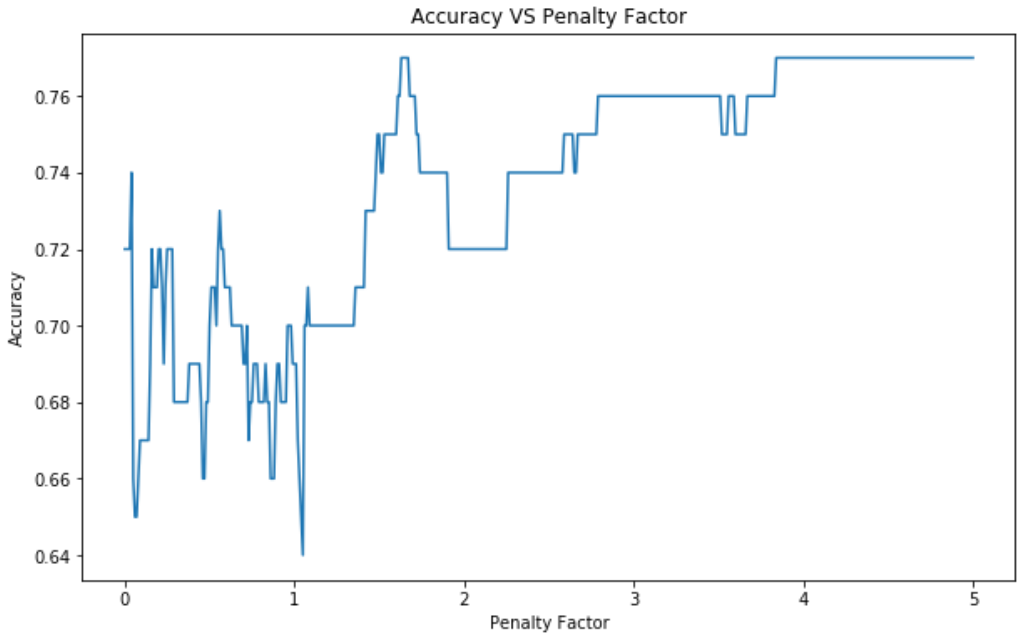

Fig: Penalty Factor λ against Accuracy

Question 19:

Penalty factor λ with best accuracy: λmax=1.63

Corresponding accuracy: 0.77

Question 20:

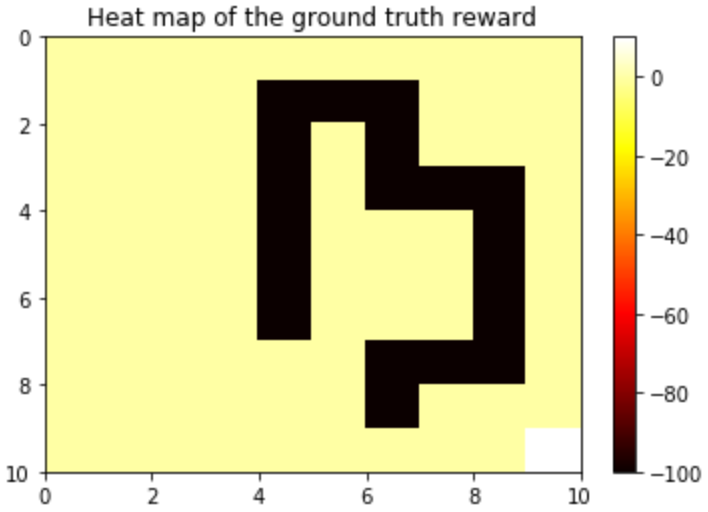

Fig: Heat map of the ground truth reward

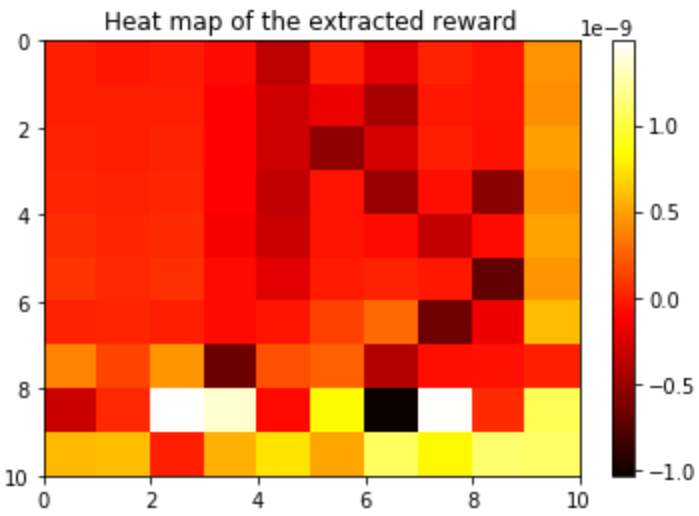

Fig: Heat map of the extracted reward

Question 21:

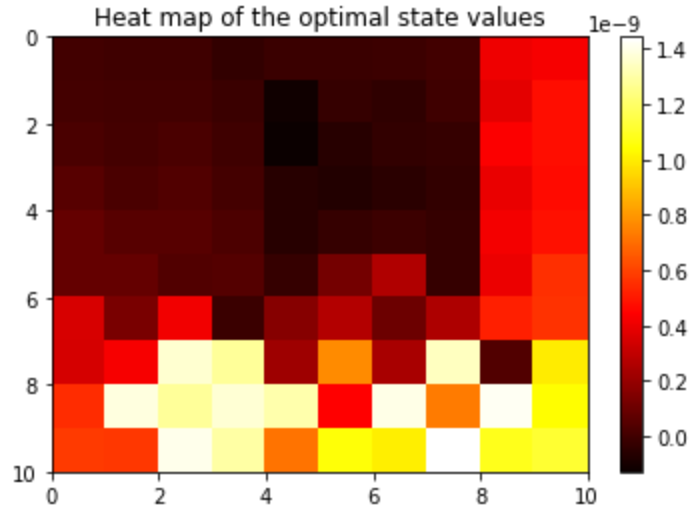

Fig: Heat map of the optimal state values using the extracted reward function

Question 22:

Different from the reward function 1, the IRL optimal state values from reward function 2 is more complicated and different with original in many ways. For the similarity, we can observe states with large state values for both the figures. For the difference, the scale of the IRL state values is in 1e-9, which is quite different from the original state values of reward function 2, though this scale doesn't matter during the decision of optimal policy (To decide the optimal policy, we only need to compare the relative relationship among states). The second difference is that there is another peak (with high state value) which does not show in the bottom right corner but shows in State 29. Besides, same with the IRL state value of optimal policy 1, state value of

reward function 2 in IRL tends to spread out its value to larger area. As we saw in the figure, the upper left part and upper right part are almost flat because the nature of linear model to have a "smooth" result.

Question 23:

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇦	⇩	⇦	⇨	⇧	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇨
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇩
4	⇩	⇦	⇩	⇦	⇨	⇩	⇩	⇨	⇨	⇩
5	⇩	⇦	⇩	⇦	⇩	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇩	⇨	⇨
7	⇦	⇨	⇩	⇩	⇩	⇩	⇨	⇩	⇩	⇩
8	⇨	⇨	⇨	⇦	⇦	⇩	⇨	⇩	⇦	⇩
9	⇨	⇧	⇧	⇧	⇦	⇨	⇨	⇧	⇨	⇩

Fig: The optimal policy of the agent using the extracted reward function

Question 24:

Different from the optimal policy 1, this one is more complicated due to the drastic variance in a relative small environment. Compared with the original policy, the IRL policy will also try to avoid states with small state values and head to the states with large reward. The difference is that in IRL policy, the agent will end in different local maximum states, where in the original

policy, the agent will end in State 99 anyway. The second difference is that there are to states with optimal actions pointing toward each other (State 28 and 38). This kind of behavior is not observed in the original policy.

Question 25:

To pinpoint the discrepancy, we plot the difference map of these two policies. As shown in table below, all the differences are shown with red marks below.

	0	1	2	3	4	5	6	7	8	9
0	⇩	□	⇩	⇦	□	□	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	□
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇩
4	⇩	□	⇩	⇦	□	⇩	⇩	□	⇨	⇩
5	⇩	□	⇩	⇦	□	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	□	⇨	□
7	□	□	⇩	⇩	⇩	⇩	□	⇩	⇩	⇩
8	⇨	⇨	⇨	□	□	⇩	□	⇩	□	⇩
9	⇨	□	□	□	□	⇨	⇨	□	⇨	⇩

Table: Highlighted difference between two policies

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇦	⇦	⇨	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇩
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇩
4	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩	⇨	⇩

5	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇦	⇨	⇩
7	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇩	⇩	⇩
8	⇨	⇨	⇨	⇩	⇩	⇩	⇩	⇩	⇩	⇩
9	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇩

Table: Original optimal policy

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇦	⇩	⇦	⇨	⇧	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇨
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇩
4	⇩	⇦	⇩	⇦	⇨	⇩	⇩	⇨	⇨	⇩
5	⇩	⇦	⇩	⇦	⇩	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇩	⇨	⇨
7	⇦	⇨	⇩	⇩	⇩	⇩	⇨	⇩	⇩	⇩
8	⇨	⇨	⇨	⇦	⇦	⇩	⇨	⇩	⇦	⇩
9	⇨	⇧	⇧	⇧	⇦	⇨	⇨	⇧	⇨	⇩

Table: IRL policy

Discrepancy:

We summarize the two discrepancies with the table above. The discrepancies are mainly about the loop/contradictory behavior and actions pointing out of the stage.

Discrepancy 1: Out of the stage actions

For the IRL policy, there are many moving out actions at the boundaries, however, the only moving out action is in State 99 for the original policy. This can be viewed as the second discrepancy. This is discrepancy is because the local maximum in the IRL rewards. With IRL

rewards, the agent prefers to stay in some boundary states, rather than moving through some states with negative rewards to reach a common ending state.

Discrepancy 2: Contradictory actions/Loop (e.g. ⇨⇦)
This discrepancy is about the actions in State 28 and 38. This kind of loop/contradictory behavior is from the peak behavior of the state values of these two states. In the state value map, we can observe that the reward of State 28 and 38 are higher than their neighboring states. Since, these two states are not boundary states, the agent has to travel between these two states for a higher reward, which results in these contradictory actions. (For the boundary cases, the agent can use the moving out actions to stay in the same state, while for the non-boundary cases, it is impossible, since the agent has to move to another state.)

Solution:
1. Boundary action restriction (for discrepancy 1)

To fix the first problem, one easy way is to restrict the actions for boundary states. The actions pointing out the stage will be ignored when calculating the optima policy. For example, for State 9 (with coordinate (9, 0)), we ignore action 1 (going down) and action 2 (going left), because the agent will move out of the stage with these two actions. After this modification, the new results are as follows.

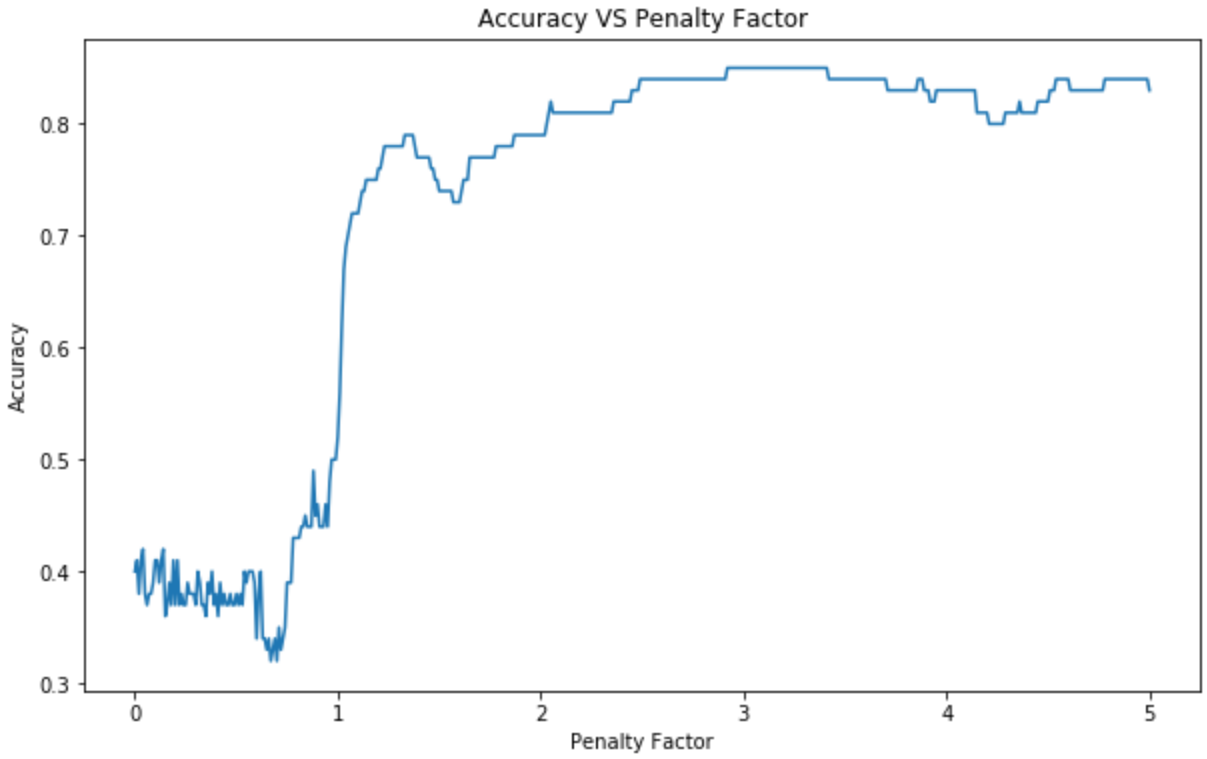

Fig: Penalty Factor λ against Accuracy

Penalty factor with best accuracy: 2.92
Corresponding accuracy: 0.85
Corresponding IRL reward function and state values are shown below.

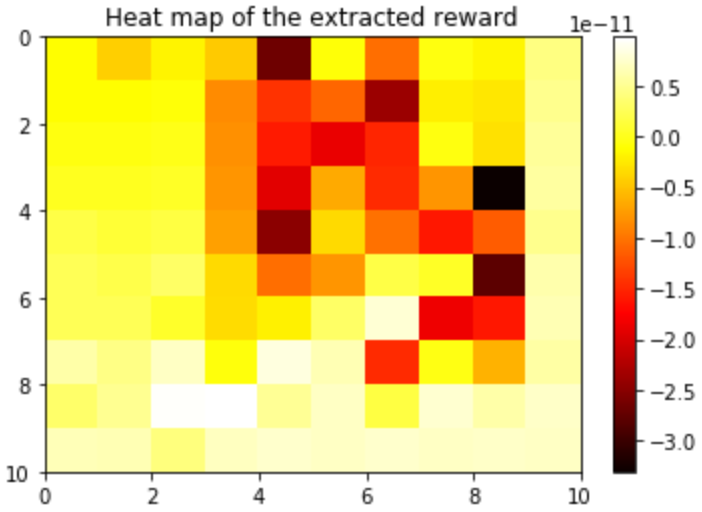

Fig: Reward Function for IRL

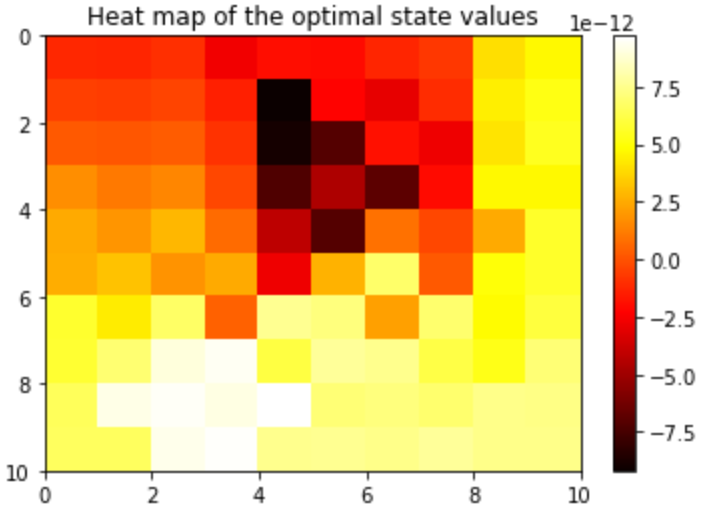

Fig: Optimal State Values for IRL

Then the new policies with this modification are shown below.

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇦	⇦	⇨	⇨	⇨	⇨	⇩

1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇩
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇩
4	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩	⇨	⇩
5	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇦	⇨	⇩
7	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇩	⇩	⇩
8	⇨	⇨	⇨	⇩	⇩	⇩	⇩	⇩	⇩	⇩
9	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇦

Table: Optimal Policy after value iteration modification

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇦	⇨	⇨	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇩
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇧
4	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩	⇨	⇩
5	⇩	⇩	⇩	⇦	⇩	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇦	⇨	⇩
7	⇨	⇨	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩
8	⇨	⇨	⇨	⇦	⇦	⇩	⇩	⇩	⇩	⇩
9	⇨	⇧	⇧	⇧	⇦	⇦	⇨	⇦	⇦	⇦

Table: Optimal Policy for IRL after value iteration modification

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇦	□	⇨	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇩
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	□
4	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩	⇨	⇩
5	⇩	⇩	⇩	⇦	□	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇦	⇨	⇩
7	□	□	⇩	⇩	⇩	□	⇦	⇩	⇩	⇩
8	⇨	⇨	⇨	□	□	⇩	⇩	⇩	⇩	⇩
9	⇨	□	□	□	□	□	⇨	□	□	⇦

Table: Comparison between RL optimal policy and IRL optimal policy with modification on the value iteration algorithm

It can be seen that the accuracy has increased from 0.77 to 0.85. More importantly, the mismatches near boundaries have reduced a lot (except those near lower boundary). Also, the number of mismatches in the center area also decreases. This is because after removing out-of-boundary actions, we are reducing the possibility of staying in the same state so the transition would be more smooth than before. Also, it can be seen that actions stepping out of boundary are easier to cause mismatches, so removing them will help to increase the accuracy. Based on the two reasons, the optimal policy will be easier to recover. And the reason for fewer changes near lower boundary is probably that the optimal action for those states are not the ones leading to out-of-boundary, and thus no significant improvement after eliminating the pointing out actions for those states.

2. Reward function normalization (for discrepancy 1) [Alternative]

Due to the most obvious difference between original policy and IRL policy is its scale, we conceive normalized the reward back to original scale (Rmax - Rmin) which may help to solve some discrepancy. That is because the original reward is too small, and the program may calculate those numbers in an unstable manner(the limitation of floating point). Small

approximation may lead to a large difference, and normalize the reward to the original scale may alleviate the influence of floating point approximation.

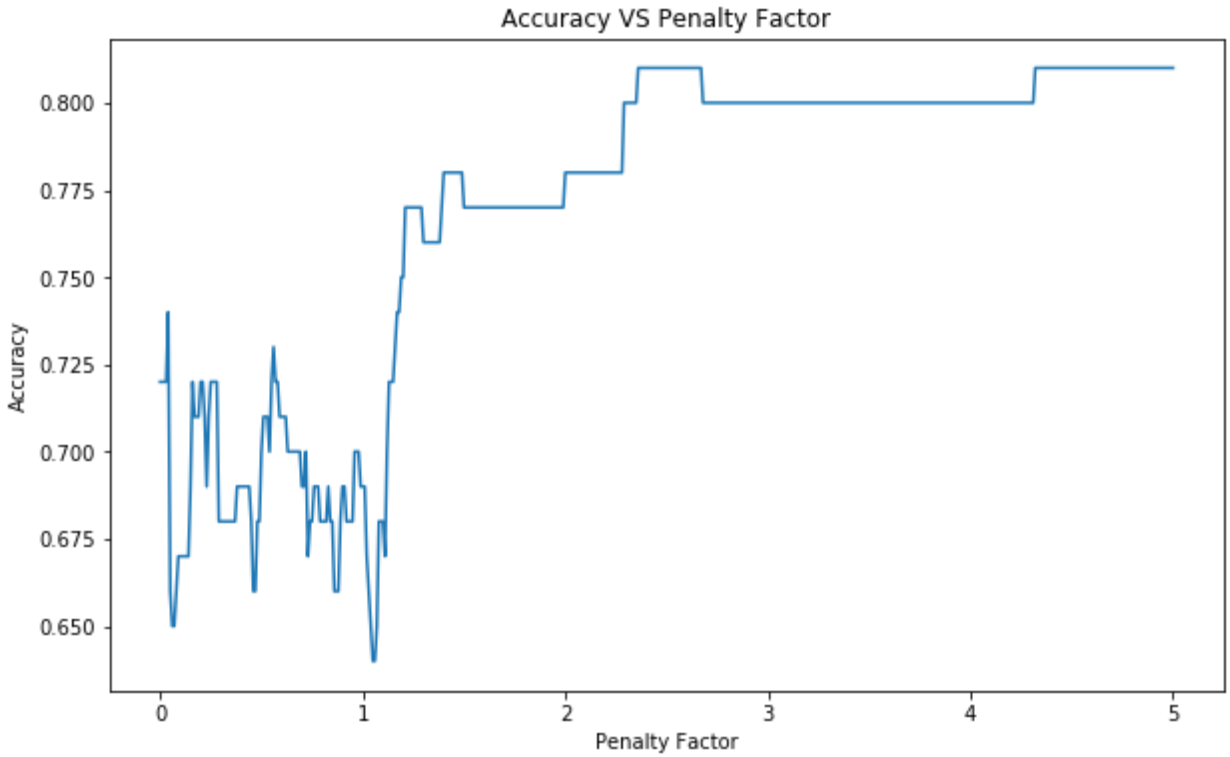

Fig: Penalty Factor λ against Accuracy

Penalty factor with best accuracy: 2.36
Corresponding accuracy: 0.81
Corresponding IRL reward function and state values are shown below.

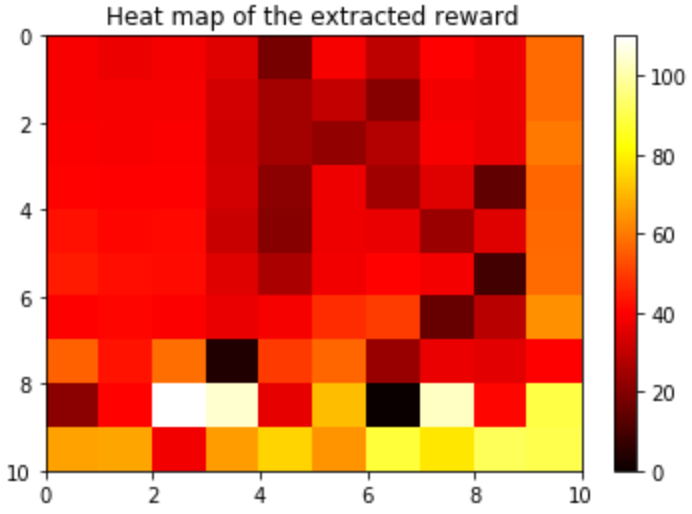

Fig: Reward Function for IRL

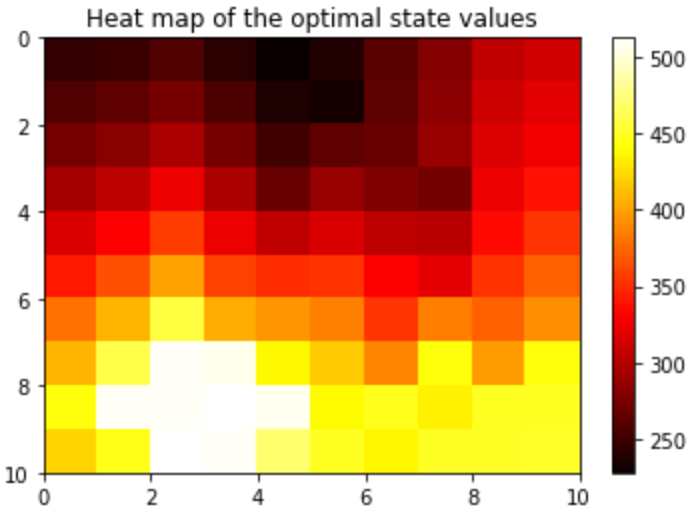

Fig: Optimal State Values for IRL

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇦	⇦	⇨	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇧	⇨	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇩	⇨	⇨	⇨	⇩
3	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇧	⇨	⇩
4	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩	⇨	⇩
5	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇦	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇦	⇨	⇩
7	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇩	⇩	⇩
8	⇨	⇨	⇨	⇩	⇩	⇩	⇩	⇩	⇩	⇩
9	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇨	⇩

Table: Optimal Policy for RL after Reward Normalization

	0	1	2	3	4	5	6	7	8	9
0	⇩	⇩	⇩	⇦	⇦	⇦	⇨	⇨	⇨	⇩
1	⇩	⇩	⇩	⇦	⇦	⇦	⇩	⇨	⇨	⇩
2	⇩	⇩	⇩	⇦	⇦	⇦	⇩	⇨	⇨	⇩
3	⇩	⇩	⇩	⇦	⇦	⇦	⇩	⇨	⇨	⇩
4	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇨	⇨	⇩
5	⇩	⇩	⇩	⇦	⇦	⇩	⇩	⇩	⇨	⇩
6	⇩	⇩	⇩	⇩	⇩	⇩	⇦	⇩	⇨	⇩
7	⇨	⇨	⇩	⇩	⇩	⇩	⇦	⇩	⇩	⇩
8	⇨	⇨	⇨	⇦	⇦	⇦	⇨	⇩	⇩	⇩
9	⇨	⇨	⇧	⇧	⇦	⇦	⇨	⇨	⇩	⇦

Table: Optimal Policy for IRL after Reward Normalization

We can observe that after the normalization on the reward function, there is only one state (State 89) with action pointing out the state space. The first discrepancy (actions pointing out the state space) is thus solved. The accuracy also increases from 0.77 to 0.81.

3. Design a new IRL algorithm (for discrepancy 2)
To solve the second discrepancy , we need to design a new IRL algorithm. Due to the time limitation, we can not provide a solution to this discrepancy.

References

[1] M. Alger, ‘Deep Inverse Reinforcement Learning’, The Australian National University, Canberra, 2016