EE232E Project 3

Reinforcement Learning and Inverse Reinforcement Learning

Hongyan Gu – 205025476

Xin Liu – 505037053

Aoxuan (Douglas) Li - 905027231

Jiawei Du - 404943853

Part I Reinforcement Learning

In this part, we used reinforcement learning (RL) algorithm to find the optimal policy for an agent running in the maze. Specifically, we studied Value Iteration algorithm and calculated estimation of rewards and corresponding deterministic optimal policy.

<u>Question 1</u> Plotting Heatmap

In this question, we plotted the heatmap of reward 1 and reward 2 given in the problem statement with coloring scale, which are shown in Fig.1.

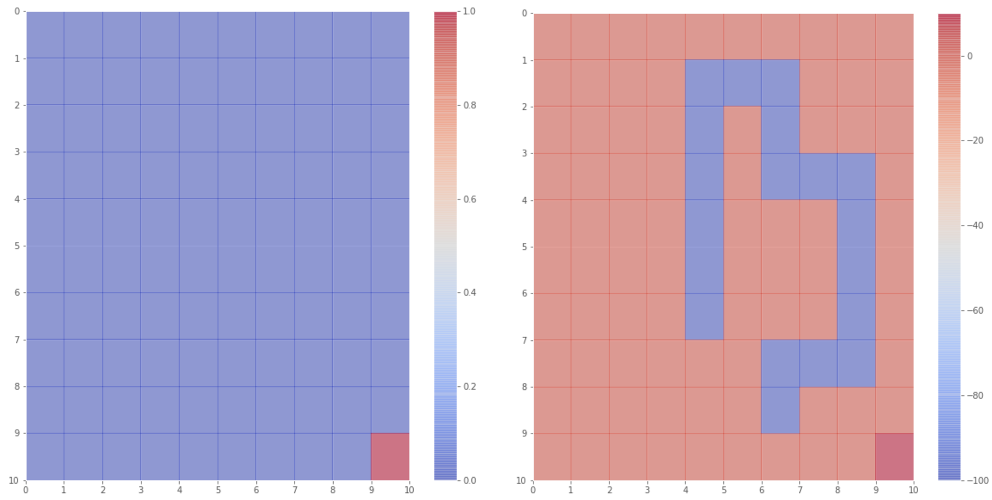


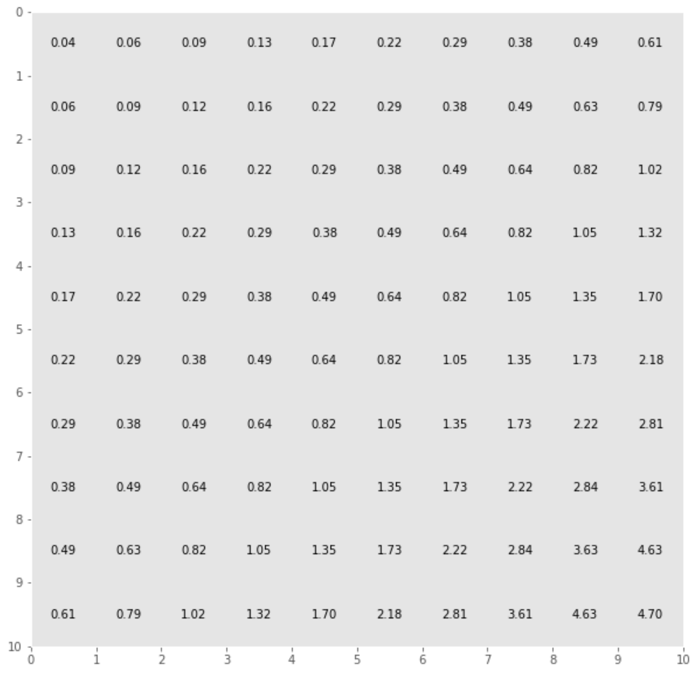
Fig. 1 Heatmap of reward function 1 (left) and reward function 2 (right), with coloring scale

As is clearly illustrated in the figure above, the red grid represents higher rewards and the blue grid indicates lower rewards.

Question 2 Optimal policy learning using RL algorithms

In this question, we introduced MDP() class (see in code) to calculate the optimal reward with Value iteration algorithm based on hints of problem statement and set Number of states = 100, Number of actions = 4, w = 0.1, Discount factor = 0.8 and ε= 0.01.

Fig. 2 shows the optimal value from reward 1 calculated from Value iteration algorithm. We can easily see that the optimal value reward matrix is symmetric.



0.04	0.06	0.09	0.13	0.17	0.22	0.29	0.38	0.49	0.61
0.06	0.09	0.12	0.16	0.22	0.29	0.38	0.49	0.63	0.79
0.09	0.12	0.16	0.22	0.29	0.38	0.49	0.64	0.82	1.02
0.13	0.16	0.22	0.29	0.38	0.49	0.64	0.82	1.05	1.32
0.17	0.22	0.29	0.38	0.49	0.64	0.82	1.05	1.35	1.70
0.22	0.29	0.38	0.49	0.64	0.82	1.05	1.35	1.73	2.18
0.29	0.38	0.49	0.64	0.82	1.05	1.35	1.73	2.22	2.81
0.38	0.49	0.64	0.82	1.05	1.35	1.73	2.22	2.84	3.61
0.49	0.63	0.82	1.05	1.35	1.73	2.22	2.84	3.63	4.63
0.61	0.79	1.02	1.32	1.70	2.18	2.81	3.61	4.63	4.70

Fig. 2 Optimal value adopting **reward function 1**

Question 3 Heatmap of optimal State-Value

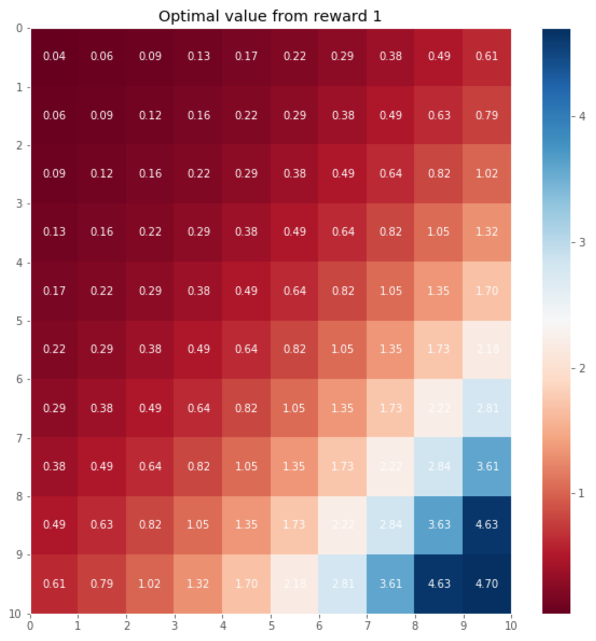


Fig. 3 Heatmap of computed optimal state-value adopting **reward function 1**

Fig. 3 shows the visualized heatmap of the optimal value of each state in question 2. From the plot we can see that, the bottom right area still has the largest reward, which is corresponding to the initial reward setting in Fig. 1, and the positive reward has been propagated to spanning of the matrix alongside the diagonal direction.

Question 4

From Fig. 1, we know that there is only 1 positive reward of V[9][9] = 1 from the initial reward matrix. When Value Iteration algorithm is computed, V'[8][9] and V'[9][8] are updated first. If the algorithm are executed recursively, the positive value would propagate along the diagonal, and that explains why the optimal reward matrix is symmetric. Another observation of the matrix is that, the top-left state values are much smaller than the bottom-right ones. This can be explained by the introduction of the discount factor γ. In this class we have discussed that γ is actually a hypermeter that how we eliminate the effect of the future. In this problem, γ=0.8, so each time the algorithm updates from a state value to its neighbors, the increment would be smaller.

Question 5 Optimal actions

In this section we implemented the computation step according to problem statement to obtain the final optimal policy of the agent navigating the 2D-grid. In the problem statement, the optimal action is selected according to the largest reward from 4 directions. In this part we do not consider the constraints on the boundary and if there exist multiple optimal values in distinct directions, we just let the algorithm randomly choose one as the optimal action. Fig. 4 shows the optimal actions based on the state-values acquired in Question 4.

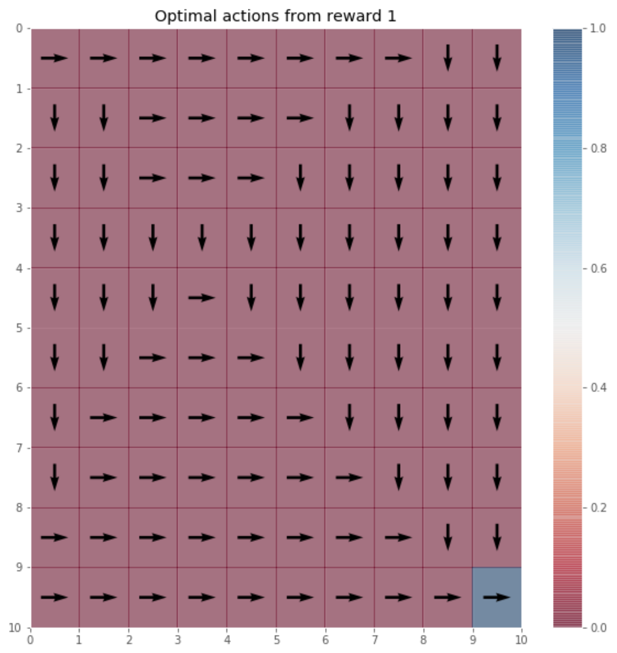
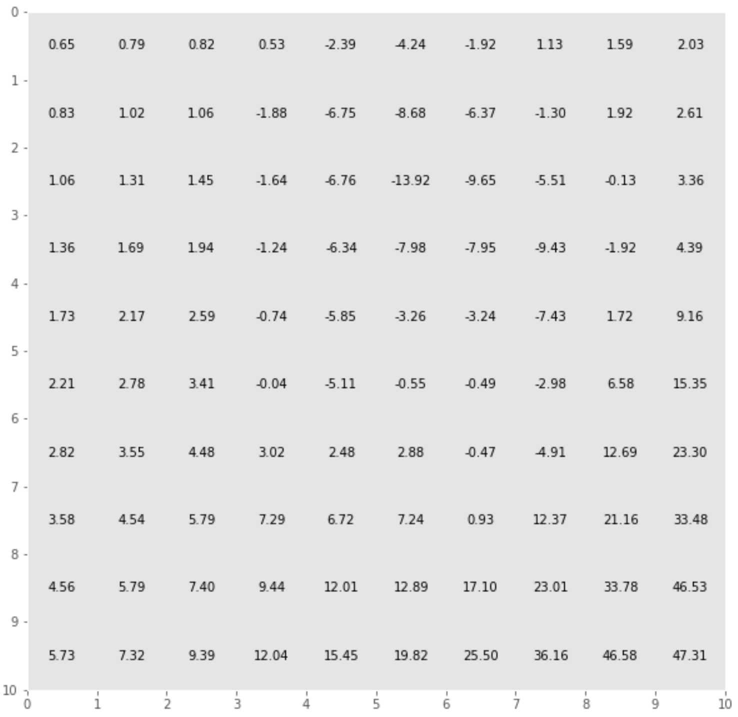


Fig. 4 Optimal policy adopting **reward function 1**

The optimal policy matches our intuition that the overall action tendency is to navigate towards the very bottom-right state because only that state owns the positive reward, which is extremely tempting compared with other plain states. Meanwhile, combining the result of Question 3 and Question 4, there are no conflict actions and the agent is not blown off the grid, and we can conclude that in this question, it is possible for the agent to determine the best action at each state by observing and pursuing the best optimal values of its neighboring states. As we have claimed in Question 4, the state located at right and bottom always has the higher state-value than the states in the top and left direction, which coincides with the optimal policy generated.

<u>Question 6</u>

Fig. 5 shows the optimal value computed through reward function 2 by Value Iteration algorithm. As expected, we can easily see that the optimal value reward matrix is no longer symmetric, since **reward function 2** is not symmetric.



0.65	0.79	0.82	0.53	-2.39	-4.24	-1.92	1.13	1.59	2.03
0.83	1.02	1.06	-1.88	-6.75	-8.68	-6.37	-1.30	1.92	2.61
1.06	1.31	1.45	-1.64	-6.76	-13.92	-9.65	-5.51	-0.13	3.36
1.36	1.69	1.94	-1.24	-6.34	-7.98	-7.95	-9.43	-1.92	4.39
1.73	2.17	2.59	-0.74	-5.85	-3.26	-3.24	-7.43	1.72	9.16
2.21	2.78	3.41	-0.04	-5.11	-0.55	-0.49	-2.98	6.58	15.35
2.82	3.55	4.48	3.02	2.48	2.88	-0.47	-4.91	12.69	23.30
3.58	4.54	5.79	7.29	6.72	7.24	0.93	12.37	21.16	33.48
4.56	5.79	7.40	9.44	12.01	12.89	17.10	23.01	33.78	46.53
5.73	7.32	9.39	12.04	15.45	19.82	25.50	36.16	46.58	47.31

Fig. 5 Optimal value adopting **reward function2**

<u>Question 7</u> Heatmap of optimal State-Value

Fig. 6 shows the visualized heatmap of the optimal value of each state in question 6.

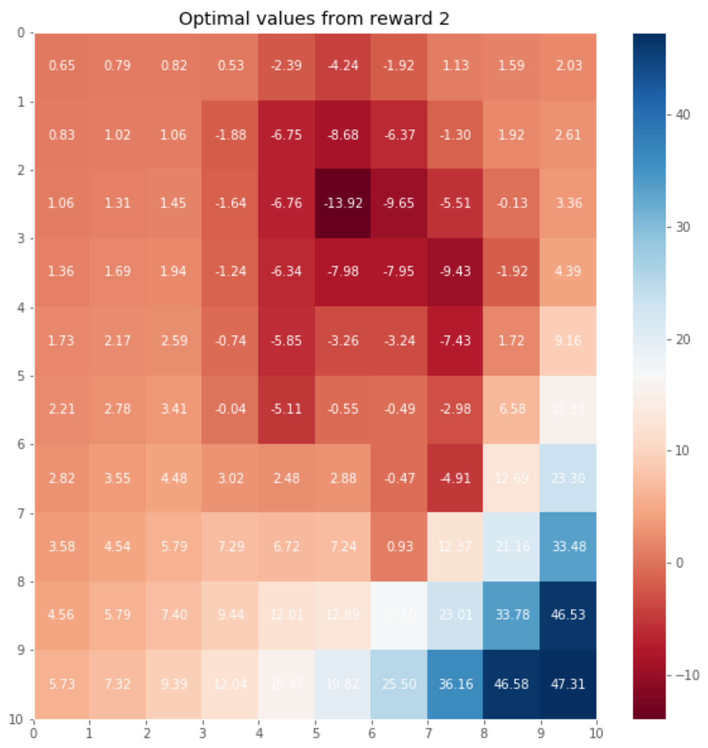


Fig. 6 Heatmap of computed optimal state-value adopting **reward function 2**

<u>Question 8</u>

From Fig. 6, the states in proximity to states having negative rewards tend to own lower state-values than those that are a little far from states having negative rewards. It is obvious that the state with negative reward disseminates its negative influence among its neighboring states, with some effects of the discount factor γ. In contrast, the states near state 99 are prone to have high state-values because they benefit from the solely positive reward of state 99. In addition, we can notice that the distribution of state-value is not sysmetric any more owing to effect of negative-reward region of irregular shape. In particular, states nearer to negative-reward region suffer more.

<u>Question 9</u>

In this section we implemented the computation step according to problem statement to obtain the final optimal policy of the agent navigating the 2D-grid. We used the exactly same method as in Question 5 and Fig. 7 shows the optimal actions based on the state-values acquired in Question 7.

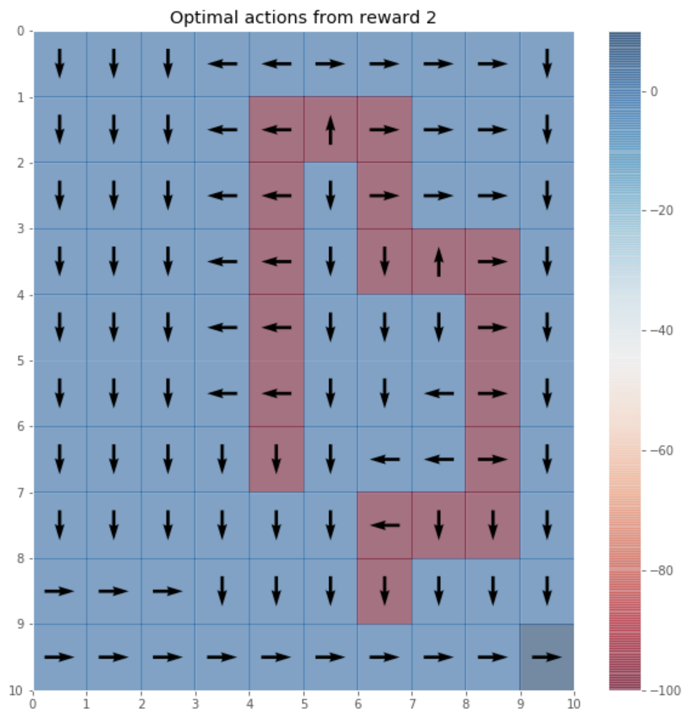


Fig. 7 Optimal policy adopting **reward function 2**

The optimal policy matches our intuition that running away from “negative reward zone” towards bottom-right state is a kind of reasonable policy. Thus, “Go right” or “Go down” is no longer a panacea in this maze, which is different from the strategy we may use in Question 5. However, the overall goal is same, which is destining to State 99, for it is the only success place characterized by its positive reward. Meanwhile, unlike the situation described in Question 5, now it is impossible for the agent to determine the best action at each state by merely observing and pursuing the best optimal values of its neighboring states because of the non-uniform distribution of rewards across states. In other words, we also need to take the impact of reward-function into consideration now.

Part I Inverse Reinforcement Learning

Question 10

In this project, we have 100 states for each block. We arrange the constraints and constants and form the following LP.

$$\underset{t,u,R}{\text{maximize}} \quad (1^T \ -\lambda \cdot 1^T \ \ 0^T)\begin{pmatrix} t \\ u \\ R \end{pmatrix}$$

$$\text{subject to} \quad \begin{bmatrix} I & 0 & -M_2 \\ 0 & 0 & -M_2 \\ I & 0 & -M_3 \\ 0 & 0 & -M_3 \\ I & 0 & -M_4 \\ 0 & 0 & -M_4 \\ 0 & -I & -I \\ 0 & -I & I \\ 0 & 0 & I \\ 0 & 0 & -I \end{bmatrix} \begin{pmatrix} t \\ u \\ R \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ R_{max} \cdot 1 \\ R_{max} \cdot 1 \end{pmatrix}$$

$$\text{where } \mathrm{M}_i = \left(P_{a_1} - P_{a_i}\right)\left(I - \gamma P_{a_1}\right)^{-1} \quad i \in \{2,3,4\}$$

With our variables $\mathrm{t} \in R^{100}, u \in R^{100}, R \in R^{100}$. For the constants part, we have $M_i \in R^{100\times100}\ for\ i \in \{2,3,4\}$, $R_{max} \in R^{100}$, and I is an identity matrix $\in R^{100\times100}$. We could make the LP in standard form:

$$\underset{x}{\text{minimize}} \quad c^T x$$

$$\text{subject to} \quad \mathrm{Dx} \leq \mathrm{b}$$

$$\text{where } \mathrm{c} = \begin{bmatrix} -\mathbf{1} \\ \lambda \cdot \mathbf{1} \\ \mathbf{0} \end{bmatrix} \in R^{300}, \ \mathrm{D} = \begin{bmatrix} I & 0 & -M_2 \\ 0 & 0 & -M_2 \\ I & 0 & -M_3 \\ 0 & 0 & -M_3 \\ I & 0 & -M_4 \\ 0 & 0 & -M_4 \\ 0 & -I & -I \\ 0 & -I & I \\ 0 & 0 & I \\ 0 & 0 & -I \end{bmatrix} \in R^{1000\times300}, \ \mathrm{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ R_{max} \cdot 1 \\ R_{max} \cdot 1 \end{bmatrix} \in R^{1000}$$

$$\mathrm{x} = \begin{pmatrix} t \\ u \\ R \end{pmatrix} \in R^{300}, \text{and } \mathrm{M}_i = \left(P_{a_1} - P_{a_i}\right)\left(I - \gamma P_{a_1}\right)^{-1} \quad i \in \{2,3,4\}$$

And, if we want to write it in the form as:

$$\underset{x}{\text{minimize}} \quad c^T x$$

$$\text{subject to} \quad \mathrm{Dx} \leq 0$$

we need to put R_{max} inside the variables matrix, where:

$$\mathrm{c} = \begin{bmatrix} -\mathbf{1} \\ \lambda \cdot \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \in R^{400}, \ \mathrm{D} = \begin{bmatrix} I & 0 & -M_2 & \mathbf{0} \\ 0 & 0 & -M_2 & \mathbf{0} \\ I & 0 & -M_3 & \mathbf{0} \\ 0 & 0 & -M_3 & \mathbf{0} \\ I & 0 & -M_4 & \mathbf{0} \\ 0 & 0 & -M_4 & \mathbf{0} \\ 0 & -I & -I & \mathbf{0} \\ 0 & -I & I & \mathbf{0} \\ 0 & 0 & I & -I \\ 0 & 0 & -I & -I \end{bmatrix} \in R^{1000\times400}, \ \mathrm{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \in R^{1000}$$

$$x = \begin{pmatrix} t \\ u \\ R \\ R_{max} \cdot \mathbf{1} \end{pmatrix} \in R^{400}, \text{ and } M_i = \left(P_{a_1} - P_{a_i}\right)\left(I - \gamma P_{a_1}\right)^{-1} \quad i \in \{2,3,4\}$$

Question 11

In this question, we swept the parameter λ from 0 to 500, to achieve a best parameter that maximize the IRL accuracy, the plot is given:

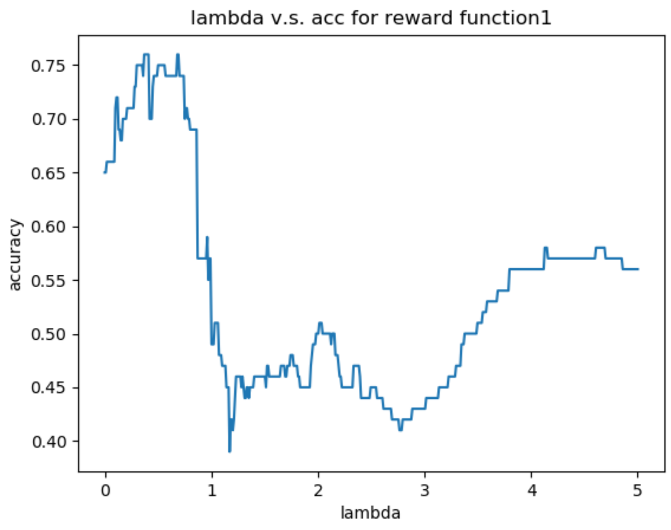


Fig. 8 plot with λ v.s. accuracy for reward function 1.

The accuracy started with a relatively low level and began to rise up. This does make sense. As we know, the l1 regularization can produce sparse matrices. When the penalty coefficient λ becomes larger, it means the solution for rewards are more prone to be zero. For our reward function 1, every state except the one at the right-down corner is zero, which means intuitively larger λ means more accurate. So that it does make sense the accuracy v.s. λ rises up at the early stage. To put it simple, the regularization term makes the reward not bumpy and become smoother, which is corresponding to the ground-truth reward map of reward function1. However, the accuracy drops at the tail because the accuracy here we used are the difference between actions. We noticed that when λ rises up to a high level, almost every state is with reward value like 1e-9, 1e-8, etc., which means they are equivalently small and close to zero. At this time, the actions choices are full with randomness, as we had argued in previous questions. So we could not ensure that every action chosen to be the same as the original map, and it may end at a stable stage.

Question 12

The maximum accuracy is reached when $\lambda_{max}^{(1)} = 0.3707$, and accuracy is 0.76.

Question 13

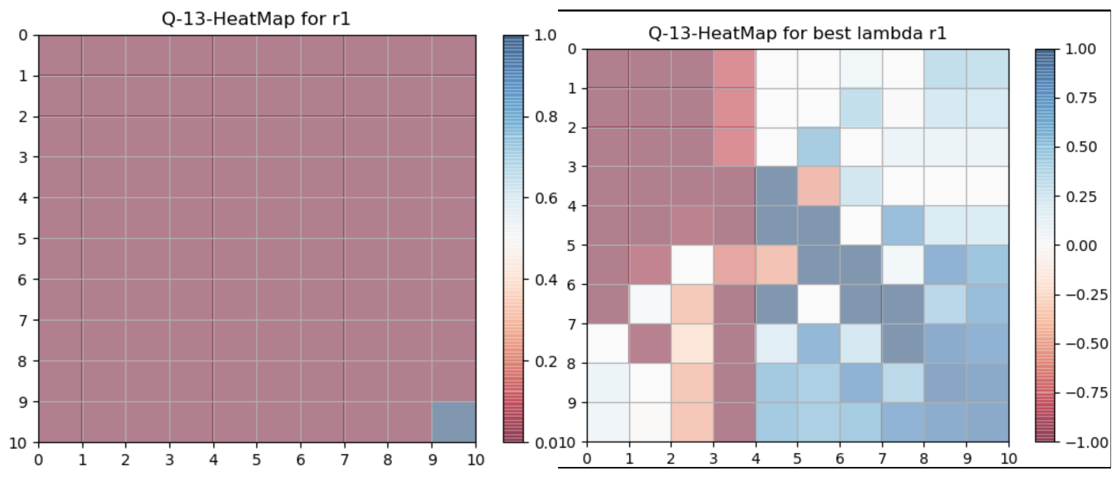


Fig. 9 (left) Ground truth heat map for reward function 1
(right) extracted reward heat map from best λ of reward function 1

From the comparison, we could see that our IRL algorithm heat map has the trend with lowest value on the upper-left corner and maximum value at right-down corner. The value increases along its diagonal.

Question 14

The value map for the extracted reward is given below:

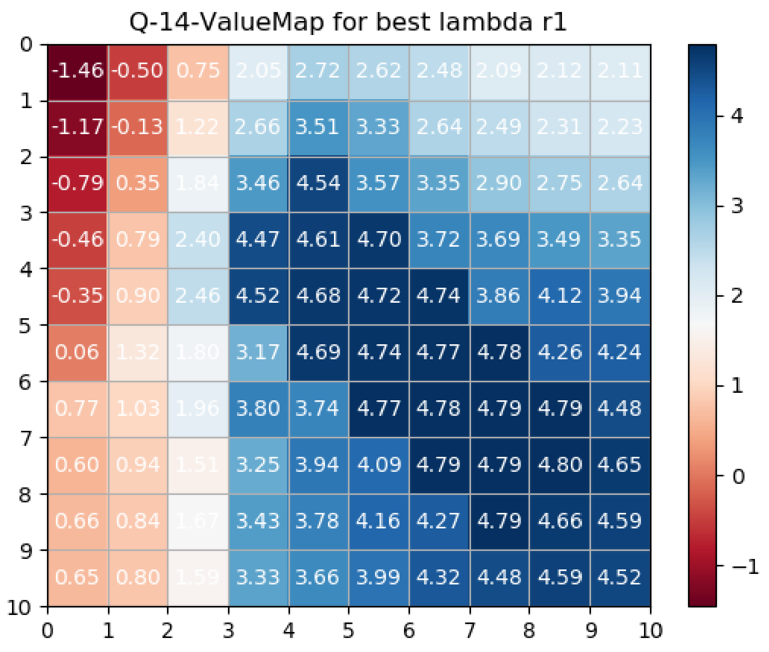


Fig. 10 Heat map of optimal value of extracted reward map from reward function 1.

Question 15

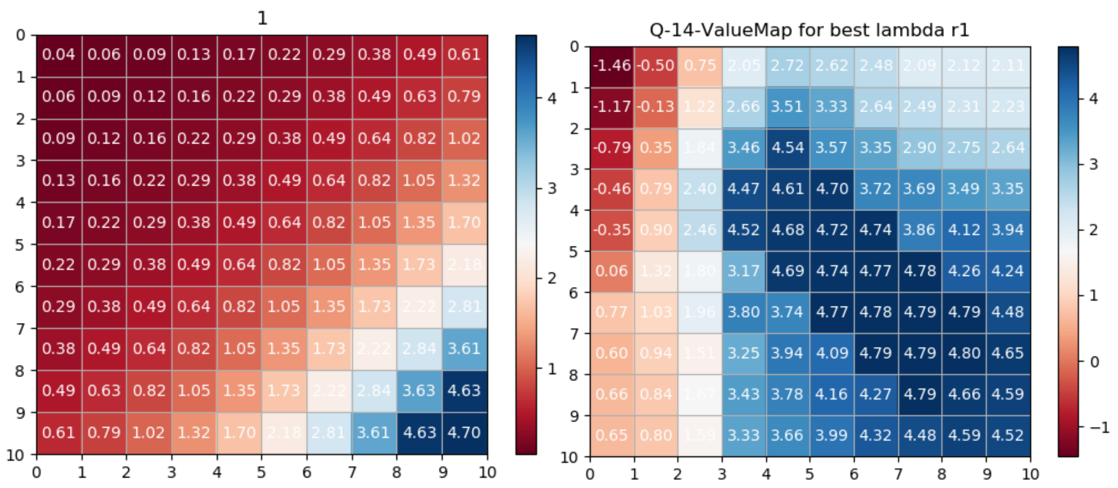


Fig. 11 (left) Optimal value map for original reward function 1
(right) Optimal value map for extracted reward function.

For similarity, both maps share the property of symmetry on its diagonal. And the best rewards value increase along their diagonals, with the upper-left corner to be the lowest and the down-right corner to be the maximum. The values are both between 0 and 5.

For differences:

- We observed that the extracted value map increases rapidly to a high value from a low one along its diagonal. This is because our penalty coefficient is not big.

- We observed that the right-down corner is not the highest value as what it is in ground-truth reward. This is because there is 'bumpiness' at the region of high values. Actually, they are very close to each other. We believe if the penalty coefficient is larger, the bumpiness would be eliminated.

- We observed that the symmetry of the extracted value map is not as perfect as the one of ground-truth one. This is because the IRL learned from our optimal action map in RL, which is intrinsically not symmetrical, so that the value map may not be perfectly symmetric.

- We observed that the reward map given by extracted reward function is smoother at high values in distribution of rewards. The value increases from the minimum to maximum with a less gradient than that of original reward value map. This may be attributed to the l1 regularization, which makes the rewards changes smoothly.

<u>Question 16</u>

The action map for extracted reward funciton is given below:

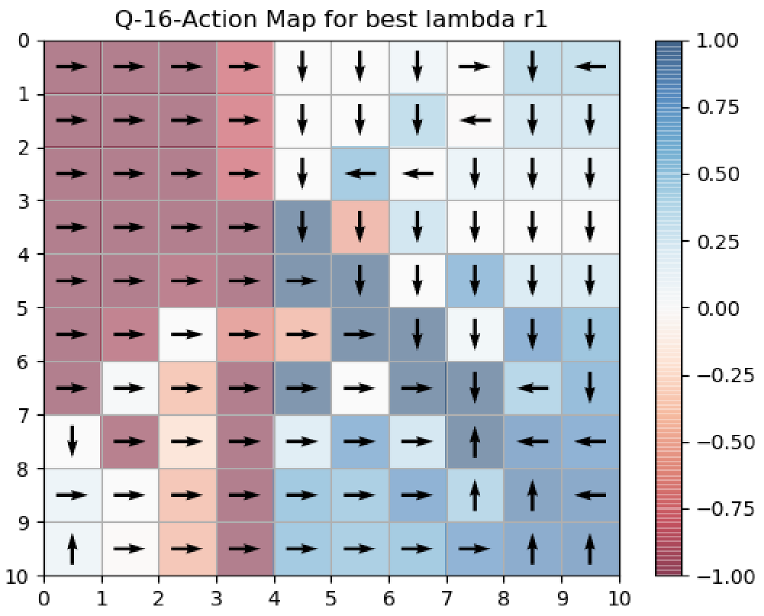


Fig. 12 Optimal policy of IRL of reward function 1

<u>Question 17</u>

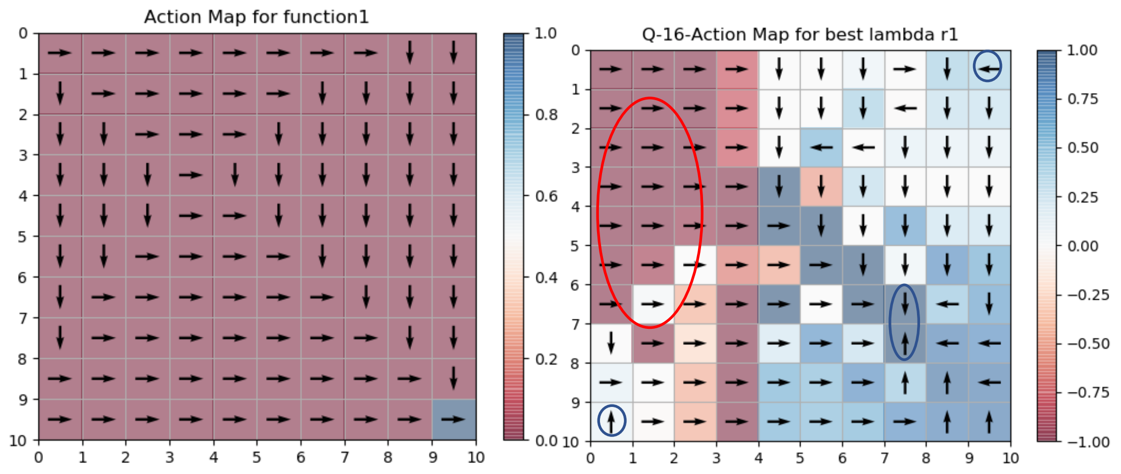


Fig. 13 (left) Action Map for reward funtion1,
(right) extracted action map 1

Similarities: The accuracy for the extracted map is 0.76, which means most of the actions are the same with the one in ground-truth reward. From the map, we could also see that arrows pointing to the rightwards and downwards occupy the most part of the map. And the overall trend is that it can go from the top-left corner to the down-right corner.

Differences:

- We observed some anomalies like some arrows go back to the start point at the left-upper corner, and some neighbor arrows are pointing to each other. These are the intrinsic problem in our IRL algorithm which we would improve it in Question 25.

- We also observed that the left part which should be down arrows are calculated as right arrows. These are because the bumpiness of calculate reward values, which make the reward not so flat as that in the ground-truth reward value map. The bumpiness is due to the penalty coefficient. Intuitively, large penalty coefficient would make the reward map smoother, but this does not mean more accuracy. This has been explained in question11.

Question 18

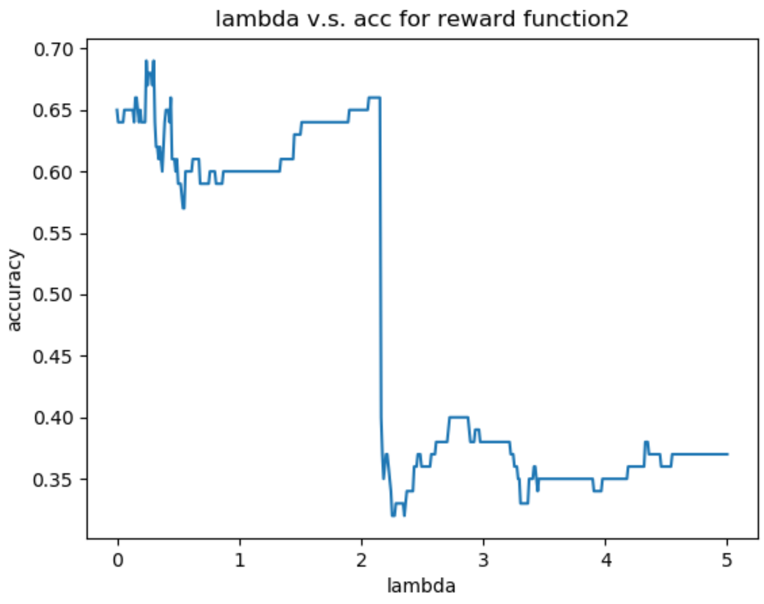


Fig. 14 plot with λ v.s. accuracy for reward function 2.

Obviously, there are two phases in the curve, and one gap between these two phases.

In the first phase, with the increasing of λ, the accuracy decreases and then increases. This is due to the structure of the action map (See question 9). For this particular map, the left-side is uniform, which requires a relatively big λ to make it smooth. However, the areas surround the barriers would require sharp changes, which require small λ. As a result, there is a trade-off in the first phase

There is a big gap. The following big gap can be interpreted as a phase transition at some points of λ_0, for $\lambda > \lambda_0$, a lot of blocks are set to be zero, and for $\lambda < \lambda_0$, there are some non-zero blocks. For this particular reward function 2, some of the values are -100 or 10, which are very far from zero. So that it does make sense if we increase the penalty term and make the matrix extremely sparse, the accuracy would drop to a very low level.

Question 19

The maximum accuracy is reached when $\lambda_{max}^{(2)} = 0.2404$, and accuracy is 0.69.

Question 20

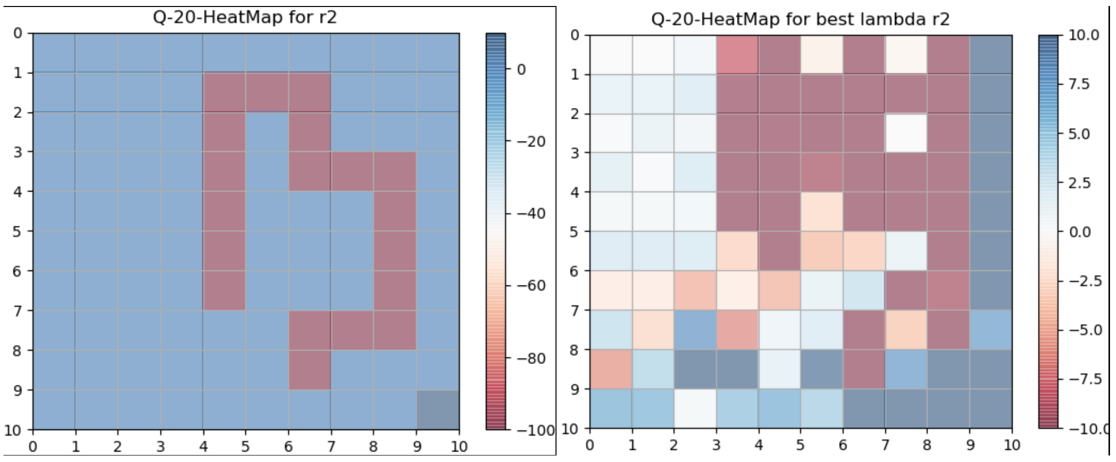


Fig. 15 (left) Ground truth heat map for reward function 2
(right) extracted reward heat map from best λ of reward function 2

The extracted reward map calculates the left-side of the map to be the low values. This does make sense because we could see in the original map that left part of the map would large possibly go through this route to the highest value. Intuitively, for IRL, the left part of the map should be recognized as low values.

The differences are also obvious:

- The barriers in the extracted reward functions are counted more than that in the ground-truth reward map. The rim is not easy to recognize. These two phenomena could be explained as the action map for the RL result, which IRL was calculated from has no differences in the red blocks and the 1-layer-out blocks of the red blocks. So that IRL would recognize them all together to be barriers with minimum values.

- The value for the last column should be all zero except the last element. However, in our IRL version, the last column is all with maximum values. From the ground-truth reward map, we just could see that the last column is a more important part of the reward map, since every block in the 2nd-last column would have a right arrow pointing to the last columns. Thus, the IRL would weigh the last column more.

<u>Question 21</u>

The value map for the extracted reward 2 is given below:

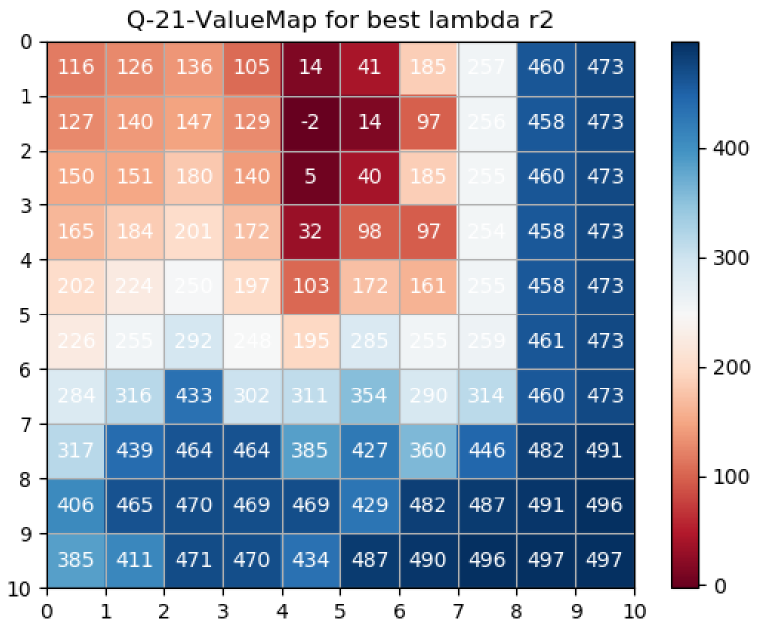


Fig. 16 Heat map of optimal value of extracted reward map from reward function 2.

<u>Question 22</u>

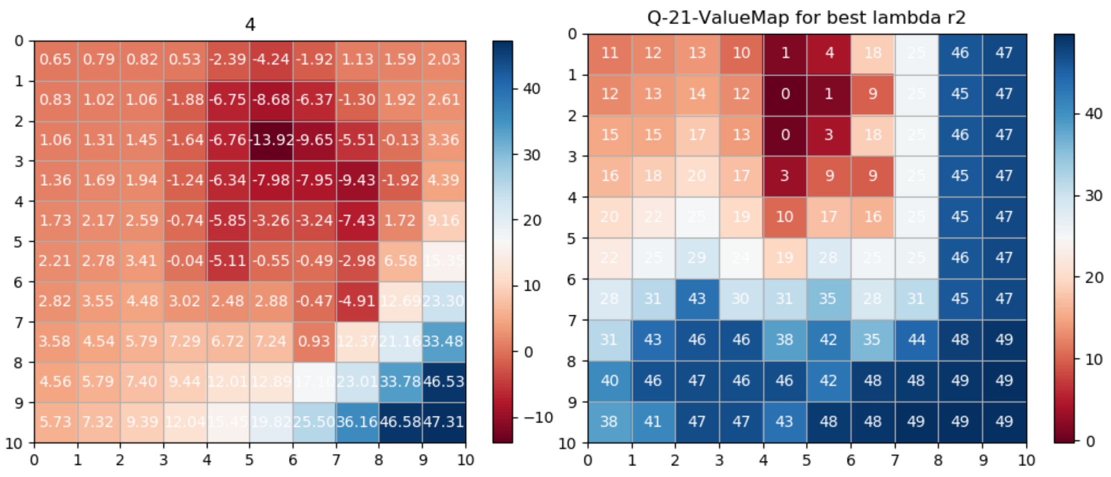


Fig. 17 (left) Optimal value map for original reward function 2
(right) Optimal value map for extracted reward function.

There are some similarities between the two reward maps:

- The extracted reward map recovered most part of the map. To see this, in the original map, there is a basin in the middle of the map, which is partially recovered for an IRL map. The right-corner part is also set to be the highest value among all of reward value, and the left-up corner is set to be the lowest value.

- Both two maps hold with symmetry along their diagonals if we ignore the barriers.

- The extracted reward map calculates the left-side of the map to be the low values. This does make sense because we could see in the original map that left part of the map would large possibly go through this route to the highest value. Intuitively, for IRL, the left part of the map should be recognized as low values.

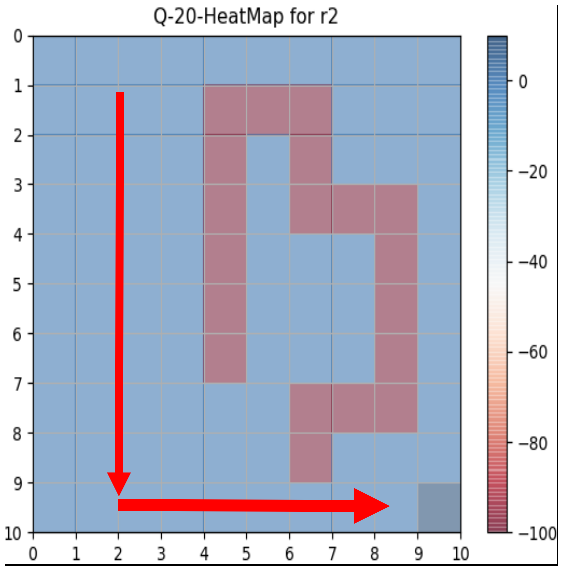


Fig. 18 Left part of the map would large possibly go through this route to the highest value

The differences are also obvious:

- The barriers in the extracted reward functions are counted more than that in the ground-truth reward map. The rim of the barriers is not easy to recognize. These two phenomena could be explained as the action map for the RL result, which IRL was calculated from has no differences in the red blocks and the 1-layer-out blocks of the red blocks. So that IRL would recognize them all together to be barriers with minimum values.

- The IRL recognized the bottom and the right-side of the map to be the maximum values. This is because in our action map, we could see that almost all the paths go directly to the last row/column and then go directly to the maximum value in the right-down corner.

- The high values in the IRL map are more than those in the RL map. This may due to l1 regularization, which makes the high-value part smoother.

Question 23

The best actions are given below:

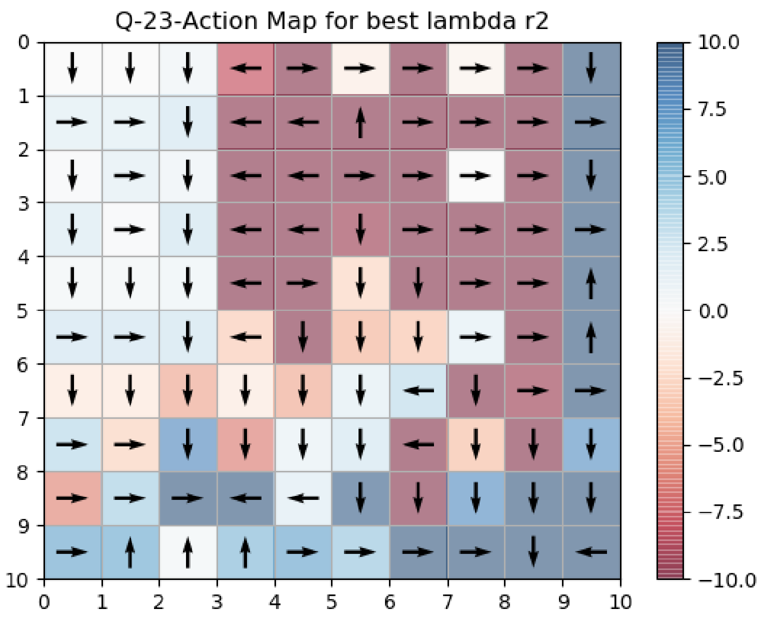
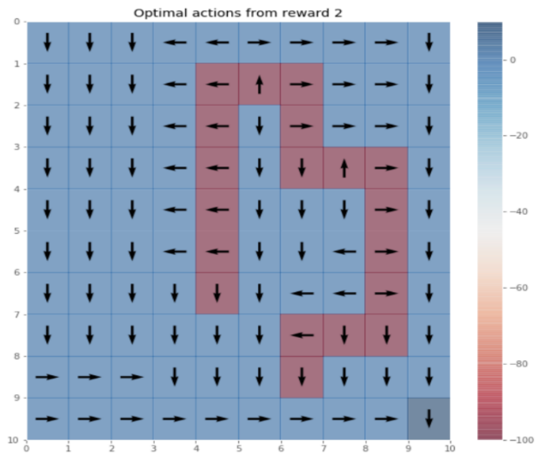


Fig. 19 Optimal policy of IRL of reward function 2

Question 24



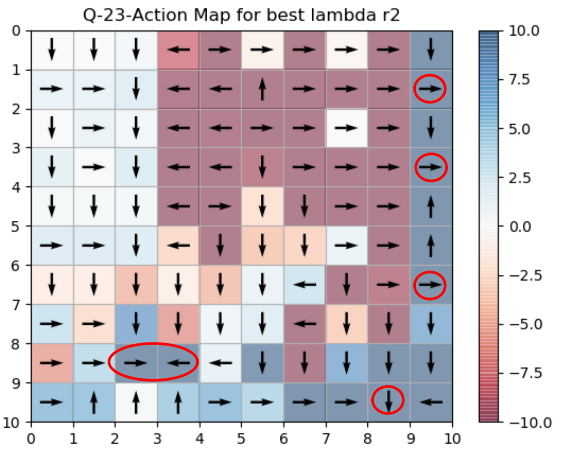


Fig. 20 (left) Action Map for reward funtion2,
(right) extracted action map 2

By comparison, the action map in part1 and the action map of the extracted values have some similarities and differences:

Similarities:

For the overall trends, the arrows that are not in barriers and barrier surroundings are pointing downwards and rightwards to the right-down corner of the maximum values in both maps, and in barriers and barrier surroundings, they tend to go outside this area to a higher-value area.

In the barriers and barrier surroundings (red blocks), the IRL arrows are almost the same with the original ones. This may be due to the l1 penalty coefficient is not large, so the map would allow more fierce direction-changes and bumpiness in the area.

Differences:

We observed that in some blocks, the decisions made are different but still make sense: like block[1][0], which is originally a down arrow in the left, and right arrow in the right. This is due to the bumpiness of the reward map calculated by IRL. Intuitively, larger l1 penalty coefficient would have smoother reward values on the map. For this particular map, the left side which is uniformly down in the original graph would perform better if we increase the penalty, but in the meantime, it would not have good performance around the barriers area. There is always a trade-off between smoothness and bumpiness.

We observed some anomalies, as marked in the circles. These are typically the two discrepancies mentioned in question 25.

- The first discrepancy, which is performed as arrows pointing out the map, is generated because the reward value at this particular block is higher than its neighbors. This can simply be modified by changing our value iteration algorithm.

- The second discrepancy with opposite-direction arrows is not easy to cope with. This discrepancy may be generated as the two blocks shares the same reward values, which means they together are the local high points of the map. To distinguish which block has 'higher' value, we need to improve our algorithm.

<u>Question 25</u>
From question 23, we observed two major discrepancies:
1. In block[1][9], block[3][9] and block[6][9], the arrows directed out the map, which is intuitively bad.
2. In block[8][2] and block[8][3], the directions of the two arrows are opposite. Thus, the path would get stuck over there.

The first discrepancy is generated because the reward value at this particular block is higher than its neighbors. This can simply be modified by changing our value iteration algorithm.

We added a judgement in our algorithm to cope with the blocks at the edges. If we noticed blocks at the edge would chosen to go out of the map, we let it re-choose from the remaining three directions the maximum reward values. By doing in this way, we would avoid the arrows to direct outside the map.

The second discrepancy with opposite-direction arrows is not easy to cope with. This discrepancy may be generated as the two blocks shares the same reward values, which means they together are the local high points of the map. To distinguish which block has 'higher' value, we need to improve our algorithm.

By doing this, our accuracy increases a little bit, from **0.69 -> 0.72,** the new action map is

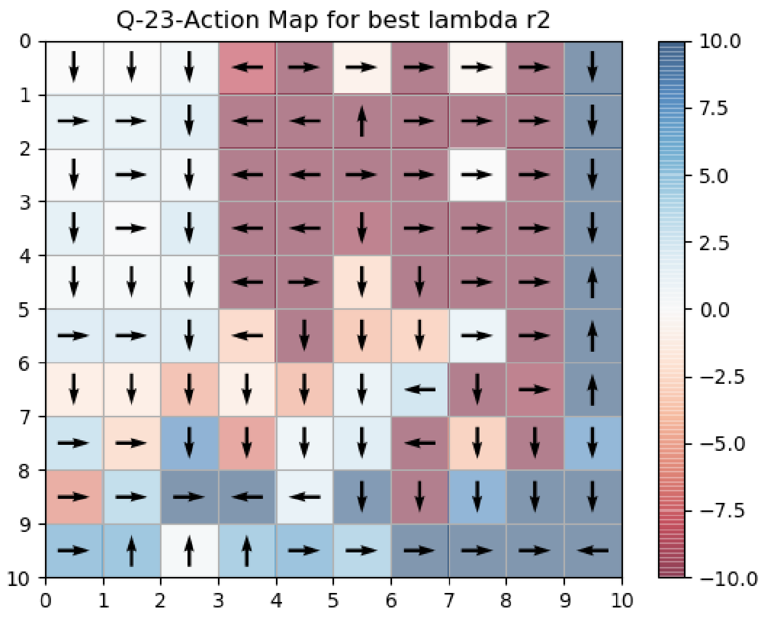


Fig. 21 Action map for reward function 2 after coping with discrepancy 1

However, discrepancy 2 still exists.

To deal with these two problems and make the accuracy higher, we used two new algorithms: **Maximum entropy algorithm** and **Deep Maximum entropy algorithm.**

I. Maximum entropy algorithm

The maximum entropy formulation is a method of matching feature expectations between observed paths and optimal paths for recovered reward functions. Intuitively, if we generate a policy which is optimal for our recovered reward function, we would expect that on average it generates the same paths as the optimal policy for the true reward function. This approach thus recovers reward functions that can be learned from using standard reinforcement learning methods to recover policies similar to the optimal policy. The observed feature expectations from our N observed trajectories is a simple average over feature counts.

To see more details about this algorithm, please refer to https://matthewja.com/pdfs/irl.pdf.

The algorithm is given by:

Algorithm 2: Dynamic programming algorithm for finding D[1].

```
Input: $\{\mathcal{P}^a_{ss'}\}, R(s), \mathcal{S}, \mathcal{A}$
Output: $D(s)$
$Z_{s_{\text{terminal}}} \leftarrow 1$;
loop $N$ times
    foreach $s \in \mathcal{S}$ do
        foreach $a \in \mathcal{A}$ do
            $Z_{sa} \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} Z_{s'} \exp R(s)$;
        end
    end
    foreach $s \in \mathcal{S}$ do
        $\sum_{a \in \mathcal{A}} Z_{sa} + \mathbf{1}_{s = s_{\text{terminal}}}$;
    end
endloop
foreach $s \in \mathcal{S}$ do
    foreach $a \in \mathcal{A}$ do
        $P(a \mid s) \leftarrow Z_{sa} / Z_s$;
    end
    $D_0(s) \leftarrow P(s = s_{\text{initial}})$;
end
foreach $t \in 1, \ldots, N$ do
    foreach $s \in \mathcal{S}$ do
        $D_t(s) \leftarrow \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} D_{t-1}(s') \mathcal{P}^a_{s's} P(a \mid s')$;
    end
end
foreach $s \in \mathcal{S}$ do
    $D(s) \leftarrow \sum_{t=0}^{N} D_t(s)$;
end
return $D(s)$
```

where D(s) is called the expected state visitation frequency of state s and represents the probability of being in a given state.

II. Deep Maximum entropy algorithm

The maximum entropy formulation can be simply extended to use deep learning. In the maximum entropy formulation, we assumed that the reward function is a linear combination of feature vectors.

$$\mathrm{R(s)} = \alpha \cdot \varphi(s)$$

Sometimes, the reward is not a linear combination of $\varphi(s)$. Instead, we could use the neural network with the activation function of sigmoid to modify this.

$$\mathrm{R(s)} = \alpha \cdot \phi(s)$$
$$\phi(s) = sigmoid\big(W \cdot \varphi(s)\big)$$

Hopefully, it would improve the maximum entropy algorithm.

*As deep maximum entropy algorithm runs for a very long time, we forfeited using it. But we believe that neutral network would perform better.

We used the paper https://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf as our reference, and improved Yiren's implementation https://github.com/stormmax/irl-imitation on maximum entropy algorithm for our project. What we did was mixing the algorithm that has dealt with discrepancy 1, and the MEA.

The major parameters are the number of trajectories and the length of trajectories. We adjusted the parameters to make it the best, but the running time is so long and we ended trying before deadline with the result in a low accuracy 0.56 (And with discrepancy 1), however, it eliminated the 2nd discrepancy perfectly. We believed that if we had more time to adjust the parameter, the performance (accuracy) would be better.

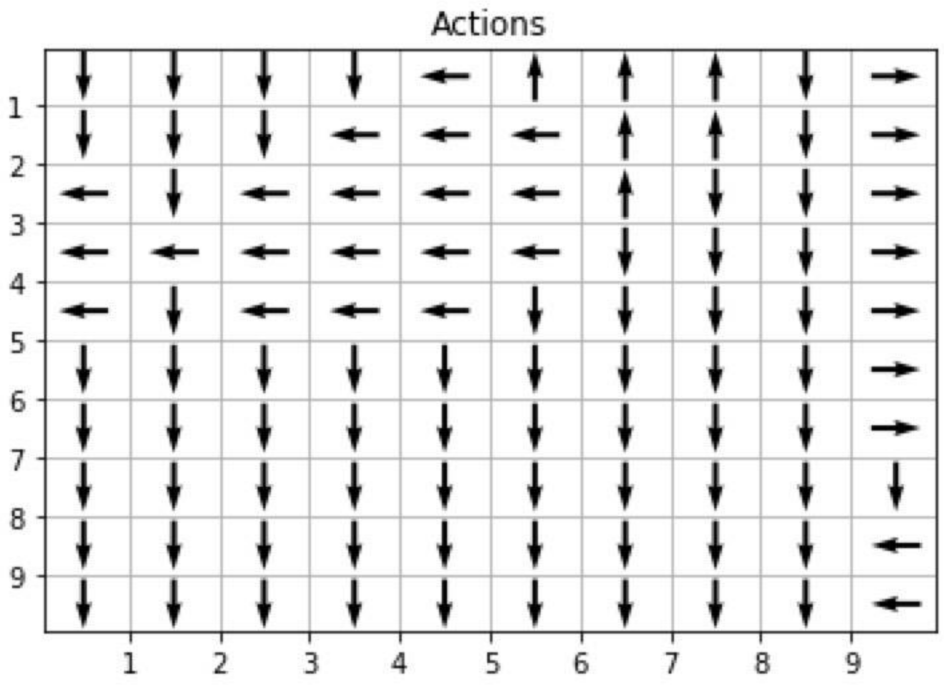


Fig. 22 Action map for reward function 2 after MEA.