HW4: Markov Decision Processes

Jiun-Yu Lee GTID: 903435223

1 Introduction

In this assignment, we will present a detailed analysis of two Markov Decision Process (MDP) problems, three reinforcement algorithm, that is, Policy Iteration, Value Iteration, and Q Learning, and how they perform on the two problems respectively. The following subsections will give a brief introduction about the problems, algorithms, and the implementation/experiment environment.

1.1 MDP Problem - Grid World

In this assignment, we chose Grid World to be the MDP problem to analyze. Grid World problem consists of a 2D grid map, with each cell in the map can be either 0 or 1, meaning that it is passable or not. As the agent, you will start at a location of the grid map, try to reach the goal location to get a reward. However, every time you take an action and move to the next cell, you will get a small reward deduction since you take more and more time to reach the goal. Additionally, there may be some penalty cell in the map that will give you a high negative reward if you move to it. We will define our grid world MDP problem as the following:

- **State:** Every passable cell (cell with 0) in the 2D grid map.
- **Action:** Up, Down, Left, Right
- **Transition:** Every time you take an action, you will have 0.8 chance to actually move toward that direction by one cell. For example, if you are in grid[1][1] and take an action Up. You will have a 0.8 probability of moving to grid[1][2], and a 0.2 probability of moving to other three adjacent cell grid[0][1], grid[2][1], grid[1][0], uniformly.
- **Reward:** A 2D array specifying the reward one will get if moving to it. It will be 100 in goal location, -1000 in penalty locations, and -1 for every other location. (ignore reward in the cell that is no passable.)
- **Discounted Rate:** Since discounted rate represents how important we treated to the future reward and how urgent we want to get to the goal, I prefer to consider this factor as a definition of the MDP rather than the hyper-parameter we can tune in the training process. Here we set the discounted rate to be 0.99.
- **Policy:** The direction you should go in each cell in order to maximize the total reward.

The reason why I choose this MDP is that it is highly relative to the natural process of human doing route planning. You can imagine a situation that you are leaving from Graduate Living Center to Clough Common to meet a friend. If you are in a rush, you may want to take the shortest path since it can lead you to the destination as soon as possible. However, if you have been bitten by a fox with Rabies on this shortest path, you will probably avoid this route no matter how fast it can bring you to the target. This problem can be simulated by a grid world MDP, where the urgentness can be represented by the discounted rate, and the location where I was bitten by a fox is the penalty location. Moreover, this problem can also be extended to the AI field such as robot and automobile motion planning. Therefore, I think this problem is interesting and want to explore more about how several reinforcement algorithms will behave on this MDP problem.

Since we want to analyze how different amount of states will affect the performance of each algorithm, we should run the algorithms on two MDP with a different number of states. To ensure the fairness of comparison, the first MDP of this assignment will be a simple grid world a small number of states, and the second MDP with be a hard grid world with larger and more complicated states and more penalty locations.

1.2 Value Iteration (VI)

If we know the transition and reward model before we are about to solve the MDP problem, we can use Value Iteration to find the optimal policy. Value Iteration try to estimate the Utility Function **U** when we are following the optimal policy based on **T, R,** and the **Bellman Equation**, and we can determine the optimal policy using this **U**. Unfortunately, since the original Bellman equation contains max operation, which makes it nonlinear, we can not solve it directly. Instead, we will use an iterative algorithm to update **U** until it converges. The algorithm is like the following:

1. Initialize U_0 with arbitrary values for all states.
2. Update U_t with the following formula:
$$U_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_t(s')$$
3. Repeat step 2 until the utility function converge.
4. Estimate the optimal policy π using the converged U_t

The reason that it will converge to the true, or approximate true utility is that we keep propagating true reward value in each iteration. This algorithm turns out to work well, and it is fast to compute for each iteration.

1.3 Policy Iteration (PI)

Like Value Iteration, if we know the transition and reward model of an MDP, we can solve the optimal policy by using Policy Iteration. However, instead of solving utility function **U**, Policy Iteration focuses on finding the optimal policy directly. The algorithm goes as the following:

1. Initialize π_0 with arbitrary policies for all states.
2. Evaluate current utility function $U_t = U^{\pi_t}$ as the following:
$$U_t(s) = R(s) + \gamma \sum_{s'} T(s, \pi_t(s), s') U_t(s')$$
3. Improve the policy π_t with the following equation:
$$\pi_{t+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') U_t(s')$$
4. Repeat step 2 and 3 until the policy does not change.

The trick we do in step 2 is that we fix the action when calculating the utility and thus remove the max operator from Bellman Equation. This makes the equation become linear and allows us to solve the utility function directly. The time complexity for one iteration is still more expensive to Value Iteration, but in practice, Policy Iteration takes fewer iterations to converge compared with Value Iteration.

1.4 Q Learning

In the real world when we are solving MDP problems, it is unlikely that we can get the transition and reward model in advance. Instead, we can only get experience tuples t(s, a, s', r) every time we perform an action in the world. In this situation, we are not able to use Value Iteration or Policy Iteration directly to solve the problem. We must either train models for the transition and reward function using the experience tuples before applying those two algorithms, or using a model-free reinforcement algorithm such as Q Learning to solve the problem. In this assignment, we will discuss how Q Learning works in dealing with these MDPs. Q Learning starts by defining a Q function as the following:

$$Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

This is very much like a Bellman Equation, except we ask the agent to take a specific action in the very beginning, and we can easily find the utility $U(s) = \max_a Q(s, a)$ and the policy $\pi(s) = \arg\max_a Q(s, a)$. We can estimate this Q function using the following Iterative algorithm:

1. Initialize the Q function with arbitrary values.
2. Given an experience tuple t(s, a, s', r), update the Q function as the following:
$$Q_{t+1}(s, a) = (1 - \alpha) Q_t(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a'))$$
3. Repeat step 2 until it converges, and use it to estimate the optimal policy.

Here, the parameter α is the learning rate, which stands for how much we should believe in the current experience than what we have learned before. Since Q Learning requires us to have a large enough experience tuple for every (s, a), it actually suffers from exploration-exploitation dilemma. Therefore, the exploration strategy we use plays an important factor to determine how good the trade-off we made between exploration and exploitation is. In this assignment, we experiment with two kinds of exploration strategy:

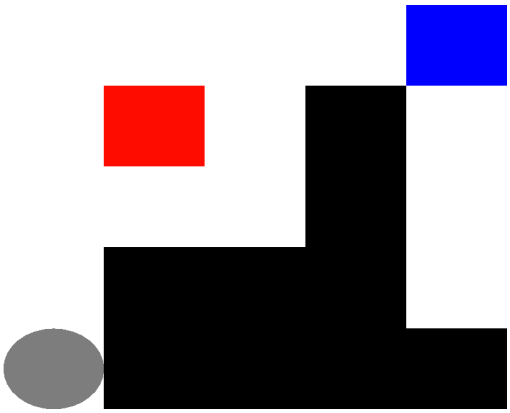
- Epsilon Greedy Strategy: This strategy will have a high probability ϵ to move with a random action at the beginning of training. ϵ will be decayed when the iteration increases and become very low eventually so that we are entirely exploitation.
- Boltzmann Strategy: This Strategy has a similar behavior as Epsilon Greedy Strategy, except that it models the probability using a Boltzmann Distribution. How this distribution looks like will be determined by the starting temperature.

1.5 Implementation / Experiment Environment

The implementation of this assignment is based on Juan J. San Emeterio and William Ma's Grid World Implementation using BURLAP, a java reinforcement learning library. The design of the two Grid World MDPs are also inspired by their implementation. For the following two grid world MDPs, only the States and Reward function are different. The Actions, Transition function, and Discounted Rate are remained the same to ensure a fair comparison between the two MDPs.

2 Simple Grid World

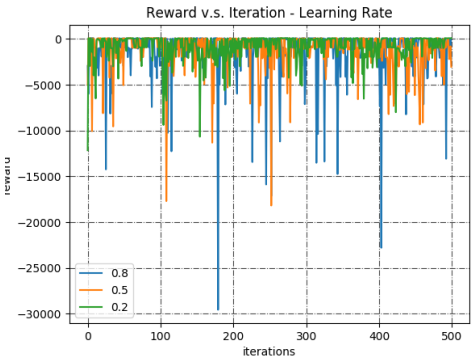
In this section, we will first discuss a relative simple Grid World like the right figure shows, with only 16 states and a naive map. The white and black cell means passable and non-passable block, respectively. The gray circle is the starting location of the grid world process. The blue rectangle shows the target location, which will give you a reward of 10 and terminate the process if moving to it. Finally, the red rectangle means a penalty location, which will give you a huge negative reward (-1000) while moving to it. This simple Grid World can simulate the real world situation that there is a danger zone in your way to the destination. You want to avoid this danger while still getting the optimal policy (the shortest path.) The following subsection will discuss how the three algorithms perform on this problem.

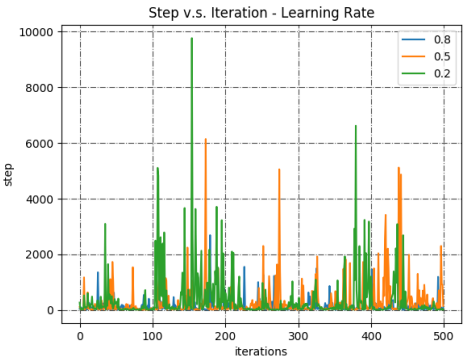


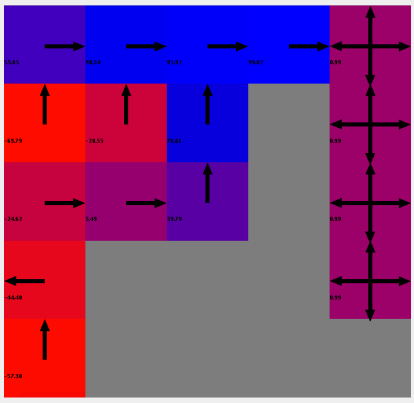
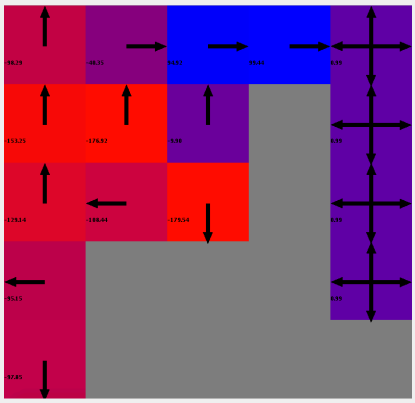
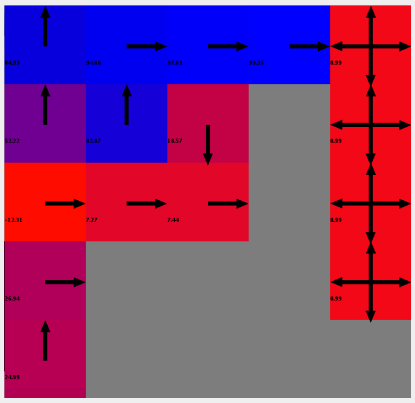
2.1 Q Learning Tuning

Since Policy Iteration and Value Iteration do not have any parameter to adjust (this two algorithms can compute the optimal policy by the MDP itself), this subsection will only focus on tuning the hyper-parameters of Q Learning, that is, the learning rate α, and the exploration strategy (epsilon-greedy and Boltzmann).

We will first try to adjust the learning rate of the Q Learning to get a good performance. The value we will try is 0.2, 0.5, and 0.8. The following figures show the results of our experiment:

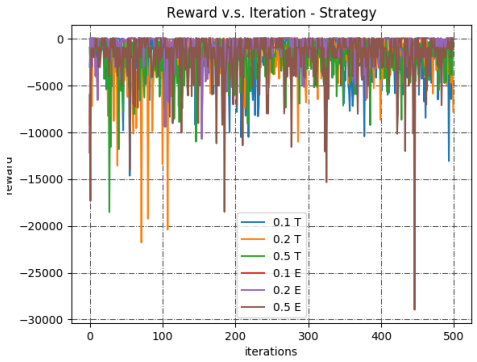


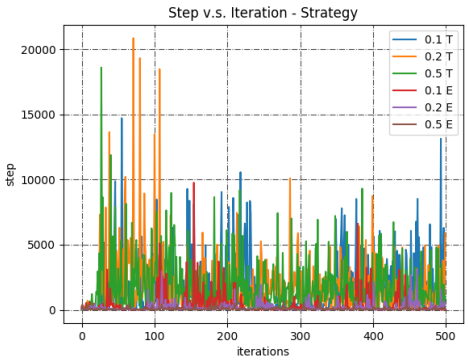


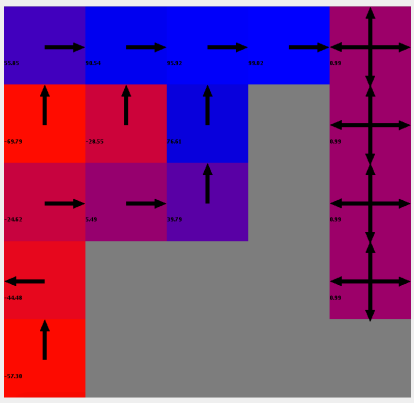
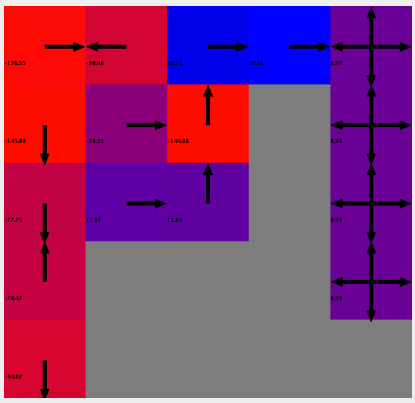
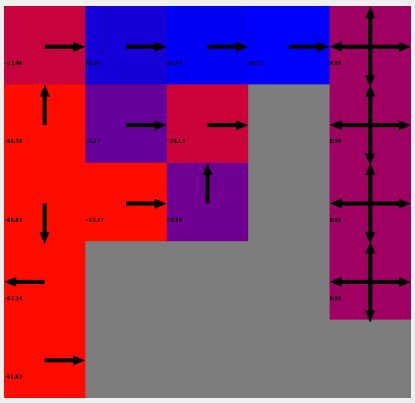
$\alpha = 0.2$	$\alpha = 0.5$	$\alpha = 0.8$
		

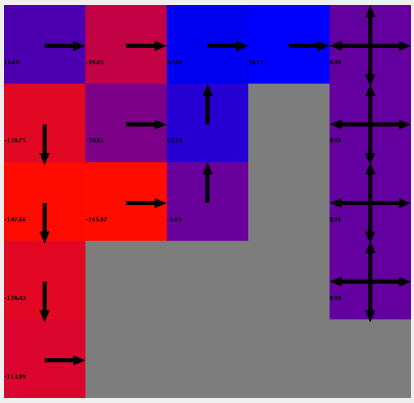
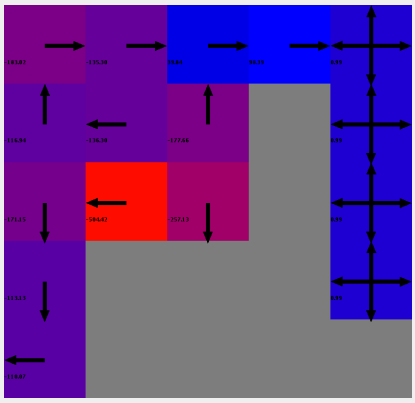
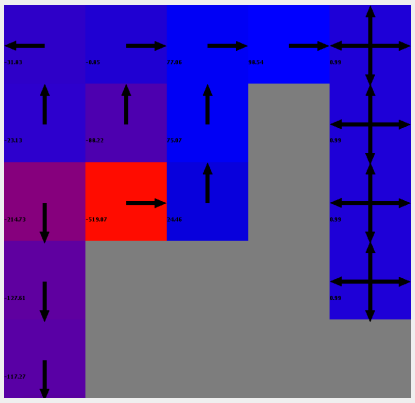
In the reward figure, we can see that although the value is fluctuating for all three alpha, alpha = 0.2 constantly gets a higher reward compared with other two alpha value. Although it seems that it doesn't get the best step number, we can see that in the policy visualization, alpha = 0.2 actually find a pretty well policy. This policy only makes one mistake at grid[0][1], while the other two policies make four to five mistakes throughout the map. Therefore, we pick alpha = 0.2 for this problem. This value is also reasonable because you always want to remember what you have learned before. If you set a high learning rate, You are probably forgetting too quickly. Considering a situation that you are trying to move toward a correct direction, while the transition function forces you to move to the penalty location, you will believe that this is a wrong move if you set an alpha that is too high. Over time, you will just fluctuate because you only believe what you see currently.

Next, we will experiment on how the two exploration methods will influence the learning algorithm. There is one parameter for epsilon-greedy (ϵ) and Boltzmann also has one (temperature). We will try 0.1, 0.2, and 0.5 for each parameter, and the following figures show the result:





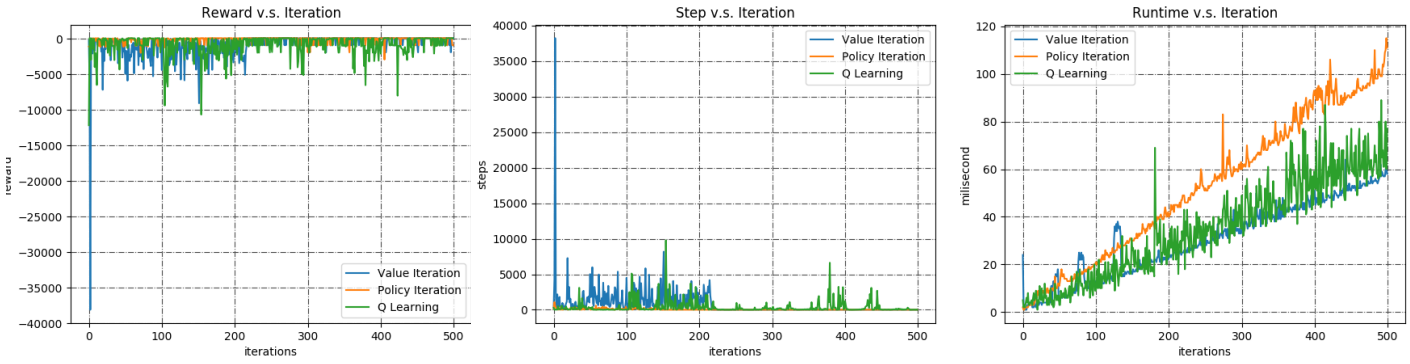
$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.5$
		

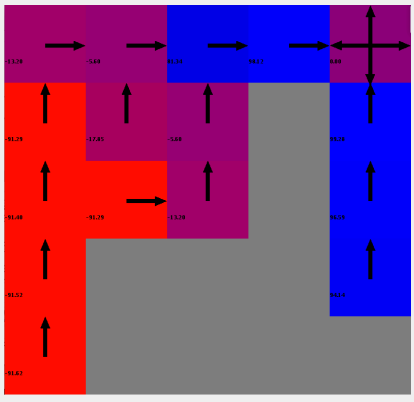
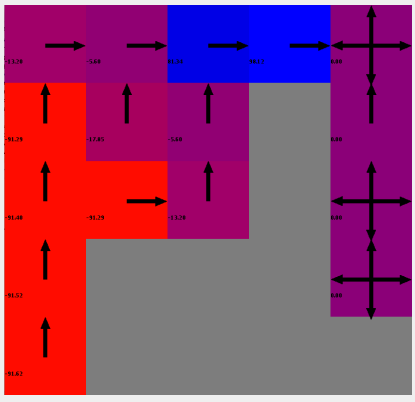
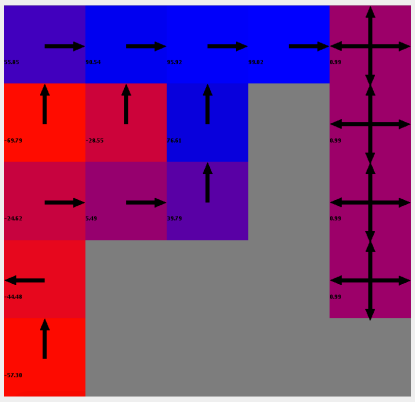
$temp = 0.1$	$temp = 0.2$	$temp = 0.5$
		

Here, we are not really able to get much information from the reward and step graph since they all fluctuate a lot. The reason for this may be that these two algorithms involve lots of randomnesses, and we are not running enough episode for their probability to decay. However, what we can still see here is that epsilon-greedy with epsilon = 0.1 can get a higher reward than others and the policy looks much more reasonable. We can also see that epsilon-greedy generally perform better than Boltzmann exploration, with higher total reward, fewer steps to termination, and slightly more reasonable policy. For this reason, I choose epsilon-greedy strategy with epsilon = 0.1 as my exploration strategy for this MDP problem.

2.2 Comparison

After tuning the hyper-parameter for Q Learning, we now run Policy Iteration, Value Iteration, and Q Learning with the parameter we find in the previous subsection on our simple Grid World MDP and present a comparison between these three algorithms. The following figures are the results we get:

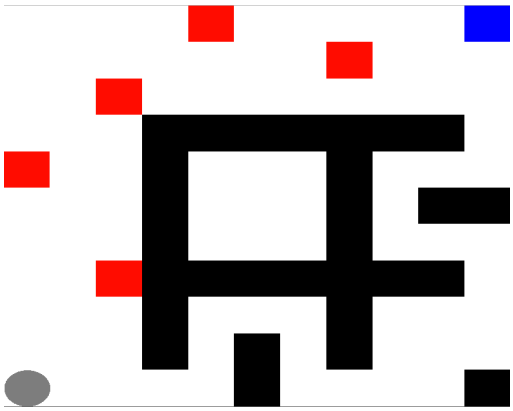


Value Iteration	Policy Iteration	Q Learning
		

In this result, we can immediately see that both policy iteration and value iteration converge to the same optimal policy, which turns out to be the global optimal in this MDP problem. However, the reward and step graph of Q Learning are still fluctuating after 500 iterations, and it is still struggling to get the policy of some of its states correct. This is under expectation since VI and PI explicitly make use of the transition and reward function, while QL needs to learn by experiencing the world, which will take much more iterations to converge. From the runtime graph, we can see that the runtime of PI is generally slower than VI, with QL stay in between. This is reasonable because one iteration of PI involves solving n linear equations for n variables, which is computationally more expensive than one iteration of VI. Nevertheless, you can see in the reward and step graph that VI converge at around the 220th iteration, while PI converge at almost the beginning. This is an evidence that PI is likely to converge much faster than VI in terms of iterations. For the total convergence time, I will say PI is faster than VI, and QL is the worst since it still did not converge after 500 iterations.

An interesting find is that there are still some differences between the policy of VI and PI. In the map, there is a side road in the right that cannot be reached by the agent. For VI, it still finds an optimal policy for these states, while in PI and QL they are simply doesn't care. This is because VI try to find the true utility function. Thus, these three states will get updated regardless it is reachable by agent or not. PI will not update these states because it only cares about policy. The utility function and policy here are definitely wrong, but it doesn't matter since the agent will never reach these states. QL will not update these states, either, since it simply won't get an experience tuple from these states. This finding also gives us a potential insight that we might be able to reuse the result of VI without rerun the algorithm even if we change the starting location to grid[5] [1].

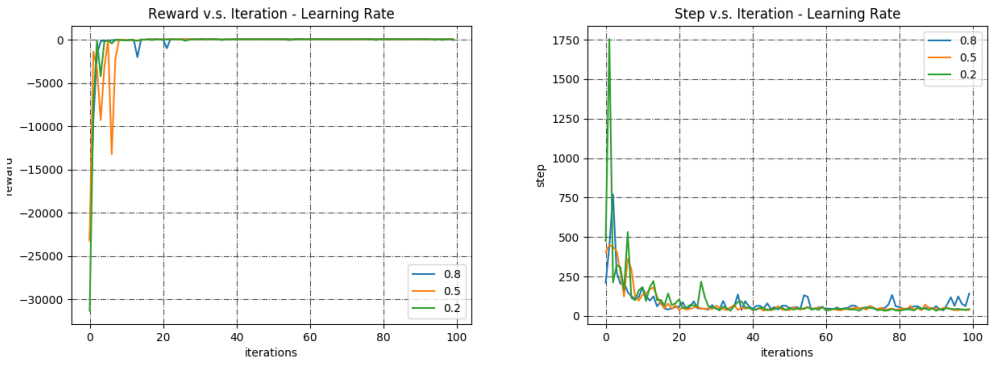
3 Hard Grid World

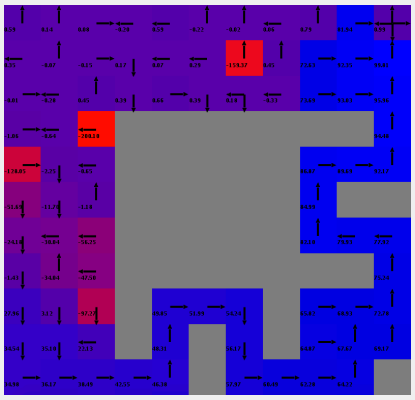
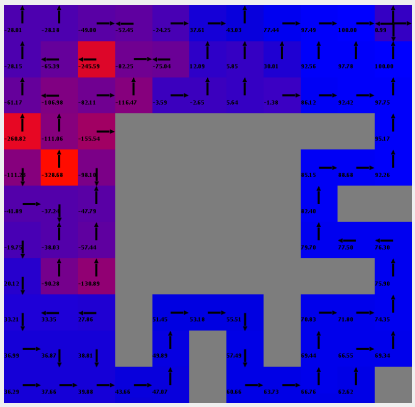
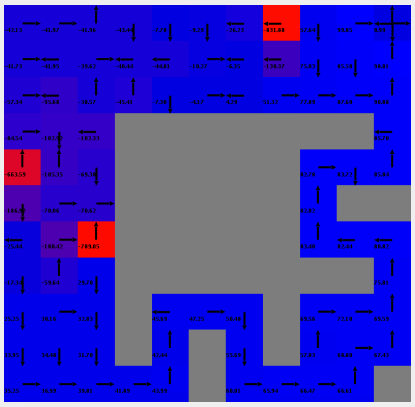


After discussing how the three algorithms perform on the simple Grid World MDP, we now run these three algorithms on a much harder Grid World as the right figure shows. This Grid World consists of 83 states with much more complicated map and more penalty locations. We can see there are two paths that can lead us to the target location, one is following the main road and go up and right. This can give us the shortest path, but there are also risks that we will move to one of the penalty locations. The other choice is taking the winding road on the right. This may lead to a longer path, but it is safer because there is no penalty location on this path. This is also a very common situation when human are planning their route to the destination, and I want to see what kinds of policies these three algorithms will generate. In the following subsections, we will have a discussion about the performance of the three algorithms on this problem.

3.1 Q Learning Tuning

As the same in the previous section, we must tune the hyper-parameter of Q Learning first before we make a comparison so that the result will be reasonable. In this subsection, we will conduct experiments to tune the learning rate α and exploration strategy. We will try the same value as we did in the previous section, that is, 0.2, 0.5, 0.8 for learning rate, two exploration strategies epsilon-greedy and Boltzmann, with ϵ = 0.1, 0.2, 0.5 and temperature = 0.1, 0.2, 0.5. The following figures show the experiment results regarding the learning rate:

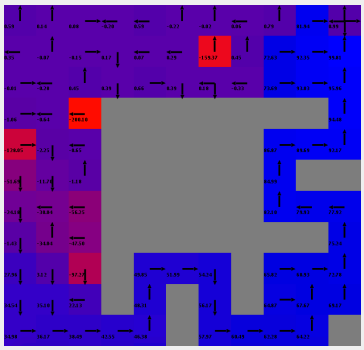
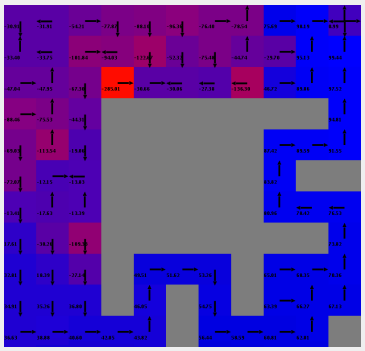
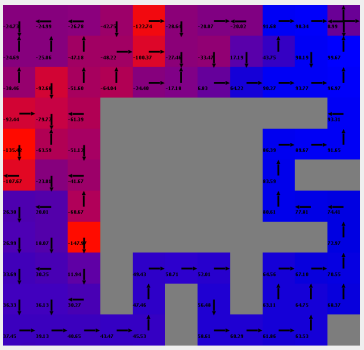
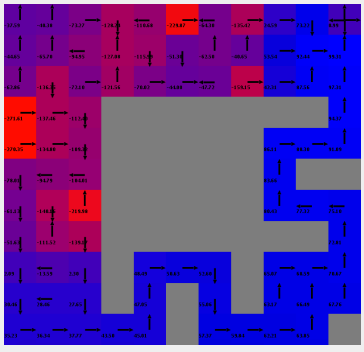
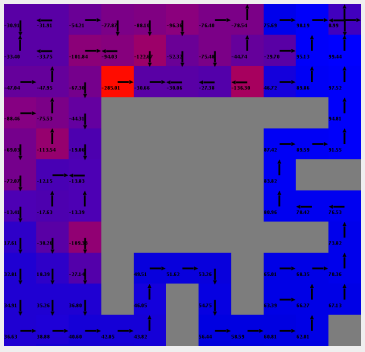
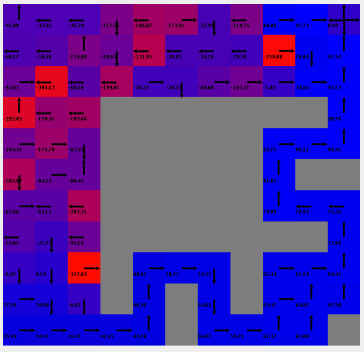


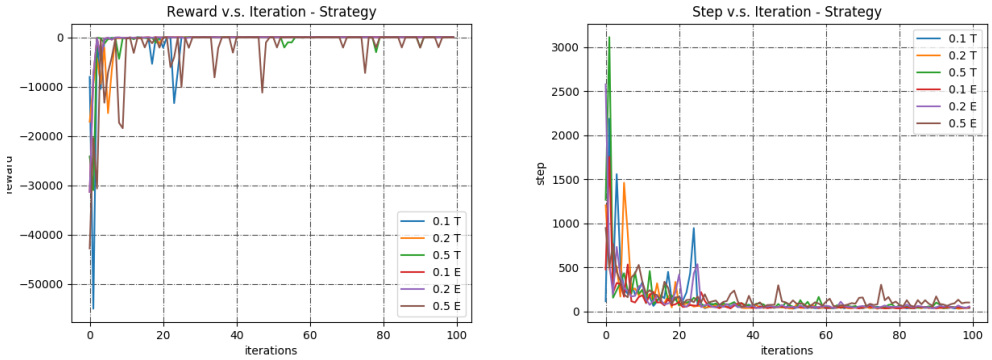
$\alpha = 0.2$	$\alpha = 0.5$	$\alpha = 0.8$
		

We can immediately see a difference is that the graph does not fluctuate so much compared with the result in section 2. The reason may be that in the previous MDP problem, we provide only one main path with a penalty location in the middle. No matter how our agent trying to avoid this penalty, there is still a chance we will accidentally move to this penalty location due to our transition model. Thus, the reward graph will fluctuate once this accident happens. However, in this problem, we provide another path which has no penalty location on it. It seems that Q Learning figures out pretty easily that it should avoid the main road and take the winding road instead. After Q Learning figured this out, there will be no fluctuation since there is no chance we will step into a penalty location.

In this experiment results, we can see that both alpha = 0.2 and alpha = 0.5 has a similar good policy result, while alpha = 0.8 is still trying to figure out the correct policy of walking through the winding road. If you take a look at the step graph, you will find out that alpha = 0.2 and alpha = 0.5 still perform similarly on the required step to termination. However, alpha = 0.2 converges faster in the reward graph than alpha = 0.5. As a result, we again pick learning rate alpha = 0.2 for this MDP problem.

After conducting the experiment to find the best learning rate, we now fix this value and conduct another experiment to find the most suitable exploration strategy for this MDP problem. The following figures show the result of the experiment:

$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.5$
		
$temp = 0.1$	$temp = 0.2$	$temp = 0.5$
		

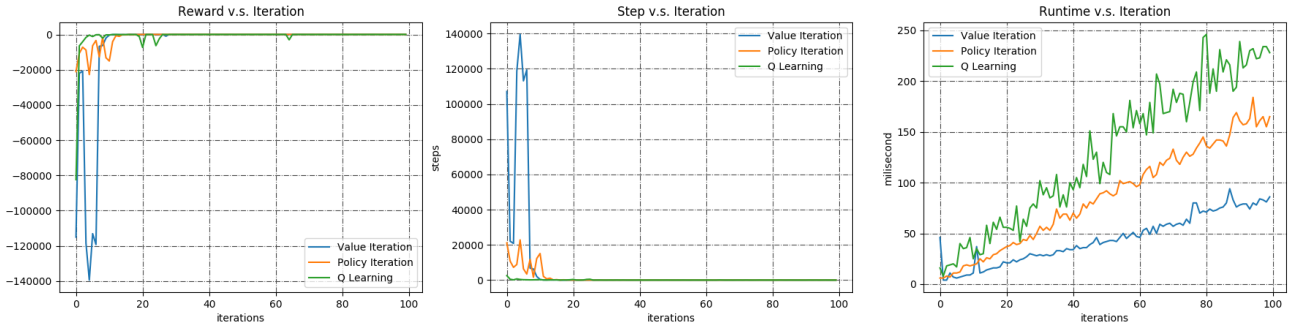


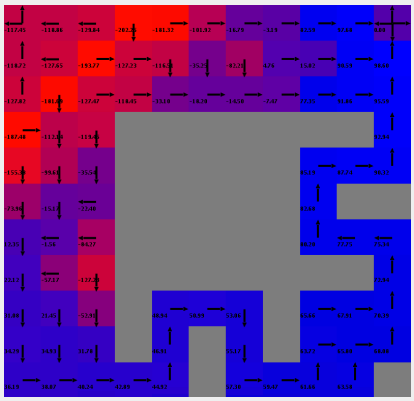
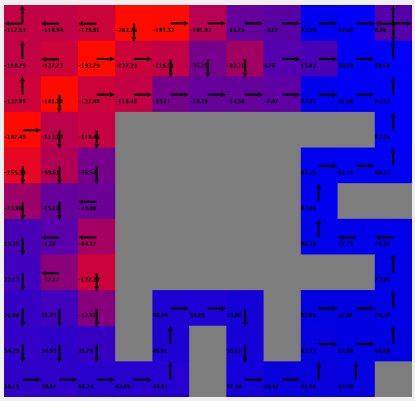
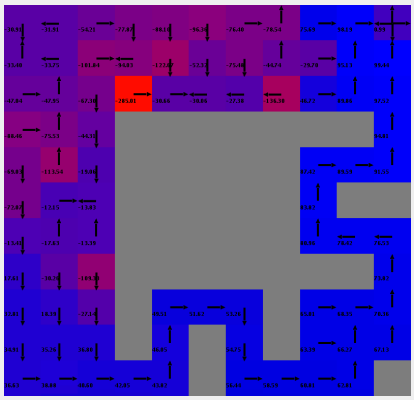
If you see the policy visualization, you will find another fact that all of the six exploration strategies seem to work well on walking in the winding road. Nevertheless, their policies on the main road are still a little bit nonsense. The reason may be that Q Learning figures out too quickly that it should avoid going along with the main road. After figuring this out, it just keeps walking through the winding path and seldom goes on the main road. As a result, we don't have enough experience tuples to update the policy of states on the main road. That is to say, we still do too much exploitation than exploration. While the result is not so bad because we expect the agent to take the winding road, we still have a chance to accidentally step on the main road due to our transition function. If that happens, we will have a bad reward since we don't have a good policy there.

Although these six exploration strategies perform similarly, we can still pick one out by looking at how fast it converges. As you can see in the reward and step graphs, epsilon-greedy strategy with epsilon = 0.1 and 0.2 seem to converge the fastest, with not much fluctuation after 20 iterations. Since we have addressed above that we might not do enough exploration for the main road. I pick the higher epsilon (0.2) value for this problem in order to enable more exploration.

3.2 Comparison

Here, we present a comparison of the performance among the three algorithms, Value Iteration, Policy Iteration, and Q Learning. The result is shown in the following figures:



Value Iteration	Policy Iteration	Q Learning
		

In the policy visualization, we can again see a similar result as section 2 that both PI and VI found the same, global optimal policy. They even figure out how to walk on the main road, such as how to avoid penalty locations, and at which state they need to proceed on the main road rather than turning back and taking the winding road. On the other hand, QL only figures out that it needs to avoid taking the main road and still have a bad policy in that region. Surprisingly, we can see that the order of convergence seems to reverse. QL in this problem turns out to converge the fastest, following by VI, and then PI. The reason, I argue, is the winding road we add here. Since we add the winding road here, QL can easily find out the optimal path to follow. Even though it still does not figure out how to walk on the main road, but it doesn't matter since we can get a decent amount of reward just by keeping following this optimal path. As long as QL figures out that it should avoid the main road, it will converge, which make it faster than VI and PI. Perhaps it is also the reason that VI can converge slightly faster than PI in terms of iterations. Adding a winding road just makes the problem easier. Another interesting finding is that increasing the number of states and complexity seems to have a much more significant influence on QL with respect to the runtime of each iteration. Here, an iteration in QL is actually an episode with a series of action and update. The episode ends when the agent reaches the target location, and the number of step during this process can grow exponentially as the number of state increase, especially when the action performed involve randomness from our transition function and exploration strategy. Nevertheless, every iteration of VI and PI only have a fixed number of updates, so they will not have an influence as significant as QL.

Here, we want to pick the best algorithm for this Hard Grid World MDP. Although QL converges faster than VI and PI, the policy it gives us is still a local optimal. VI gives us a global optimal policy, and its total convergence time is quite close to QL. As a result, we pick VI as the best algorithm to solve this MDP.

4 Conclusion

After discussing the simple and hard Grid World MDPs and the performance of Policy Iteration, Value Iteration, and Q Learning on these problems, we now know that not only the number of states but also how these states combine together to form the actual problem will influence the performance of each algorithm. Normally, we will think that adding more states will make algorithms much harder to converge, but the structure of the states also plays an important role. In fact, we can see in our analysis that all of the three algorithms converge much faster in the hard Grid World (within 20 iterations) than the simple one (VI 220 iterations, QL did not converge after 500 iterations.) I argue that this is because we add a winding road without any penalty location to the target, which makes the problem become easier. Despite this fact, we can still see that increasing the number of states does increase the runtime of each iteration for all three algorithms, which is under our expectation, and the effect is more significant on QL.

As for the performance of the policy, both VI and PI give us a global optimal policy within a small number of iterations. However, as stated in the very beginning, these two algorithms are not very practical since they make explicitly use of transition and reward function, which is often not accessible in the real world problem. Q Learning is a model-free reinforcement algorithm that makes no assumption on transition and reward function. Unfortunately, as we can see in our analysis, QL surfers from exploration-exploitation dilemma, which means we need both explore and exploit at the same time. In order to get a good approximation of the global policy, one should experience each (s, a) pair sufficient amount of time, but you will learn nothing if you just explore and not exploit. This dilemma makes QL take much more iterations to train and it is almost impossible to get the global optimal policy like we can do using VI and PI. As the MDP problem becomes more complex (more states, actions, even continuous states), what exploration strategy you use to balance the tradeoff between exploration and exploitation will become more and more important for QL to get a good policy result.

5 Reference

- Udacity Lecture - https://classroom.udacity.com/courses/ud262
- BURLAP - http://burlap.cs.brown.edu/
- burlap_examples - https://github.com/jmacglashan/burlap_examples
- Epsilon-Greedy Strategy - BURLAP - http://burlap.cs.brown.edu/doc/burlap/behavior/policy/EpsilonGreedy.html
- Boltzmann Strategy - BURLAP - http://burlap.cs.brown.edu/doc/burlap/behavior/policy/BoltzmannQPolicy.html

- Boltzmann Distribution - Wikipedia - https://en.wikipedia.org/wiki/Boltzmann_distribution
- Juan J. San Emeterio's Implementation of Grid World - https://github.com/juanjose49/omscs-cs7641-machine-learning-assignment-4
- William Ma's implementation of Grid World - https://github.com/willzma/CS4641-Machine-Learning/tree/master/4.%20MDPs%20and%20Reinforcement%20Learning