CS 7641 Assignment 4

Student: Yinglin Li (yli973) Email: allamli@gatech.edu

1. Markov Decision Processes Problems Description

When Professor Charles first mentioned Markov Decision Processes (MDPs) problems in the class, I was impressed by its precise and interesting definition. It is a model that defines a discrete, time stochastic control process, which means that the next state after taking an action from the current state is only related to the current state and has no relation to previous states before the current states.

In my understanding, the two most important stuffs of MDPs are its states and its transition model (in other words, the actions of each state). What we need is to find the optimal policy to ensure to get the best reward by taking an action from the current state. However, MDPs is not always based on the real-world but it can provide a framework for decision-making, especially useful when studying a wide range of optimization problems of reinforcement-learning or DP. As the result, there are many applications in different industrial and research area based on MDPs.

In the class, the example given by Professor Charles was based on the grid world MDPs model. In such a model, given a m*n matrix, which represent the states that an agent can access or not. The agent can be in the accessible block, which we call the current state and then it can take actions by moving up, down, right or left to get to the absorbed state, also called the terminal. Normally, there can be multiple terminals in the map, they can be with positive, negative or 0 reward and each action it takes will also have a reward (in most cases, it is negative or it will keep running all the time and the best total reward will be infinite). Therefore, the goal of the model is to find the best policy to make the agent get the best reward totally when it gets to the terminal.

In this paper, we used two MDPs problems based on this kind of grid world domain. But as homework descriptions said, one was in an easy grid world domain with only 6*6 = 36 states, the other was a more complex model with 40 * 40 = 1600 states, a much larger number compared to the easy one.

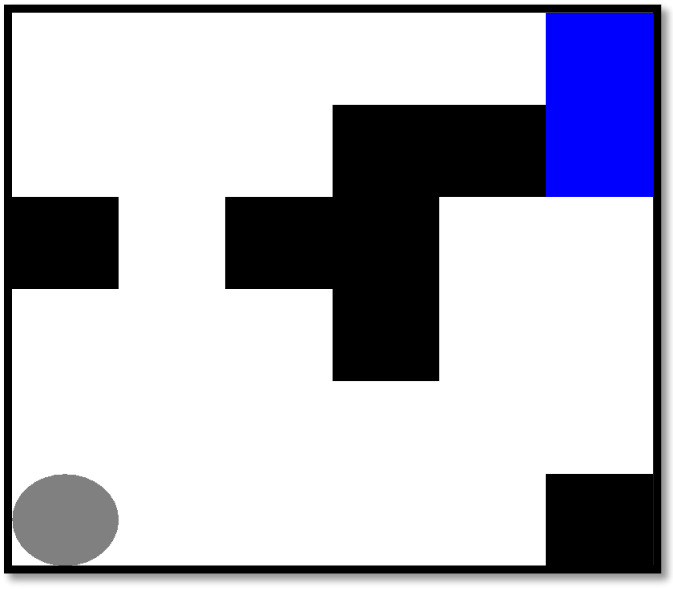

Fig 1.1. Easy Grid World Model

1.1. Easy Grid World Problem Description

The easy grid world problem was a relatively small map with 6*6 = 36 blocks of a matrix just like figure 1.1. It was nearly the same as Charles' grid world example in the class. The gray circle was the moving circle, the black blocks were

walls that could not be accessed and the blue block meant the terminal state which was the goal that the agent should move to.

Just as the Charles' example, there were two terminals in this problem, one was in the upper right corner with a reward of 10. The other was with a reward of -10, just below the terminal with positive terminal. Since both of them were blue, it might be difficult to point out their positions.

The transition model was as the class's example, which was not fully deterministic with a probability to move to other directions. Given a prob = 0.8, the probability of the agent's action that was effective to move directly by this direction. Otherwise, the agent would move to other two directions that were orthogonal to the original direction with a probability of (1-prob)/2 = 0.1 for each of them. The reward of each action was -1.0 which could easily to see as the steps of the agent that needed to walk to the terminal.

1.2. Complex Grid World Problem Description

The Complex Grid World Model was a 40*40 matrix with crossing blocks and some random reward in random locations with fixed random seed. In Figure 1.2, it seems like a normal grid world domain, but some of accessible blocks were with a reward of -20 and other accessible blocks were with -1 as reward, just like figure 1.3Different from the Easy Grid World, there was only 1 terminal state in this example and it was with no reward.

The transition model was also similar to the first one, but this time the prob = 0.95, which meant probability of moving forward with a given direction was 0.95, other two orthogonal action was 0.025.

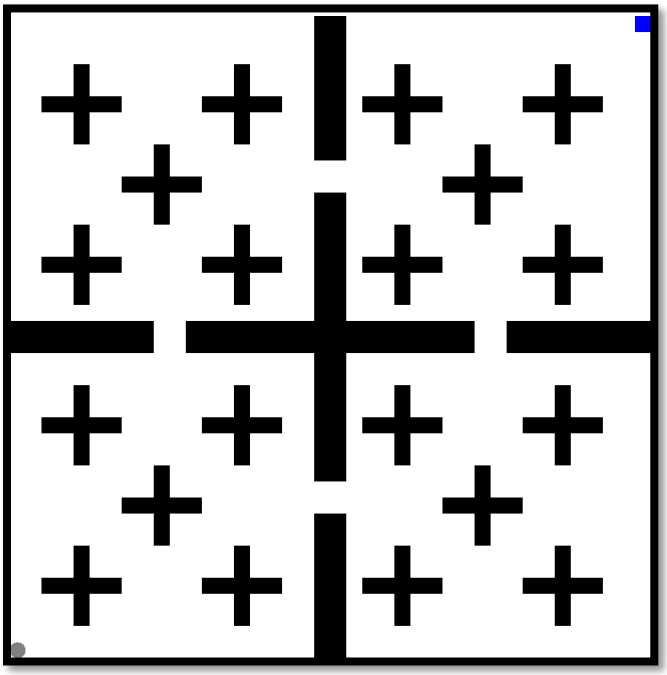

Fig 1.2. Complex Grid World Model

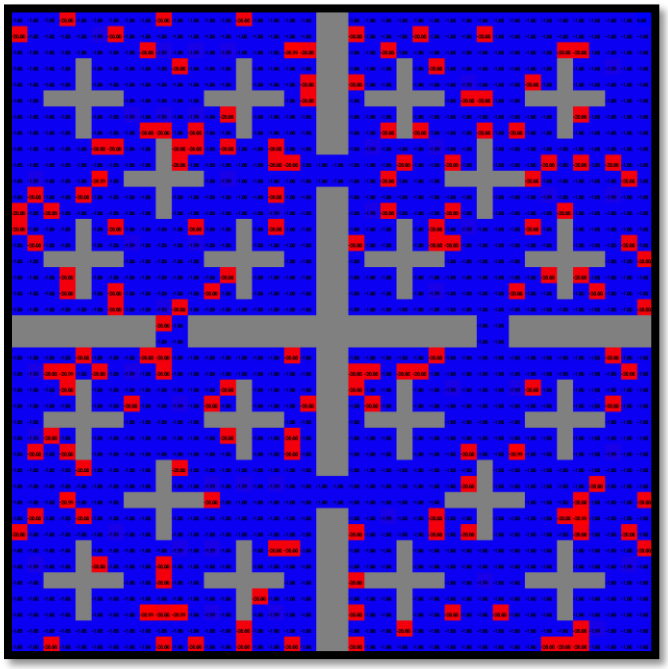

Fig 1.3. State reward of Complex Grid World

1.3. Why Are They Interesting?

At first, when Professor Charles first introduced the concept of MDPs and Reinforcement Learning, he mentioned the grid world example that is similar to the first easy grid problem so it's my first impression of how a MDPs looks like. However, it seems like the grid world domain does nothing helpful to the real-world problem because it may appear to be a trivial problem with discrete states and a too simple matrix.

But the interesting part of this kind of model is how to apply it to different kinds of real-world applications and problems. The more interesting thing is that, the process of how to transform a real-world problem to an existed grid world domain model or other MDPs model is very impressed and amazing!

Here are two interesting examples of how grid world domain applies to the real-world problem. First, the automated robotics navigation. Suppose a robot is surrounded by different kinds of buildings and obstacles, and we need the robot adapt itself to the new surroundings by navigating itself to a given path. This case can also be extended to a current heat research topic, self-auto driving system with a start point and a terminal point. But in this situation, states will be much different and we have to consider other effects such as velocity, acceleration and avoiding collision with other driving cars.

Second example is the race strategy and it seems like another extension of the first example. Given a racing path with start and end point, and the matrix represents the racing circuit. The goal of is to find the best policy that help racers reach the terminal with the least time or with the least punishment in other worlds. But as mentioned above, the agent must have velocity and acceleration that will make the action model much more complex.

In conclusion, gird world example is a simplified problem of MDPs with discussion of how to get the best policy when you are travelling in this 2D-matrix world. It's very difficult but interesting to how to transform our real-world to a world that the computer can recognize and how to design the transition model for it.

2. Value Iteration and Policy Iteration Analysis

2.1. Value Iteration Analysis

Problem	Algorithm	Converged Iterations	Used Time(ms)	Converged Value
Easy Grid World	Value Iteration	27	77.64	88
Complex Grid World	Value Iteration	99	1460.84	-118

Table 1. Value Iteration Data

Value Iteration is a method based on Dynamic Programming. It starts from the terminal point of the states and the works backward, refining and calculating an estimate of best value or policy. We must notice that the algorithm works by estimate instead of calculating the best value directly and this value is assumed to be dependent on future rewards

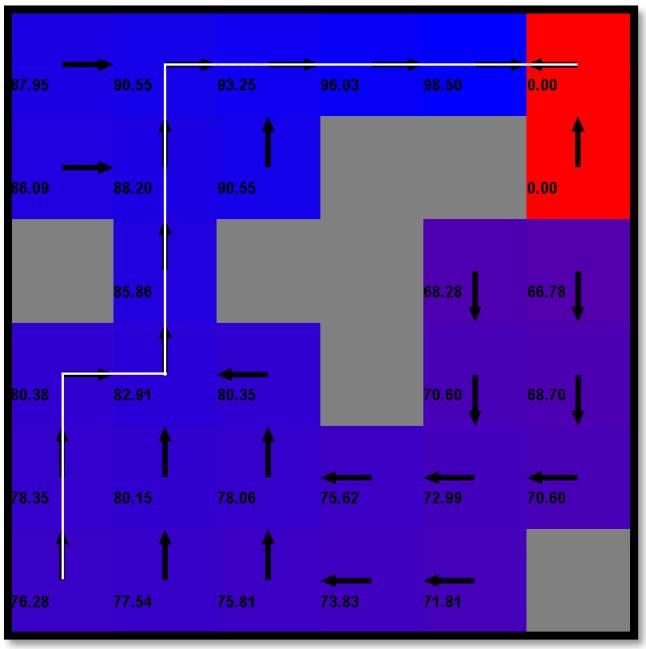

Fig 2.1. Easy Grid World Model VI Converged State Value

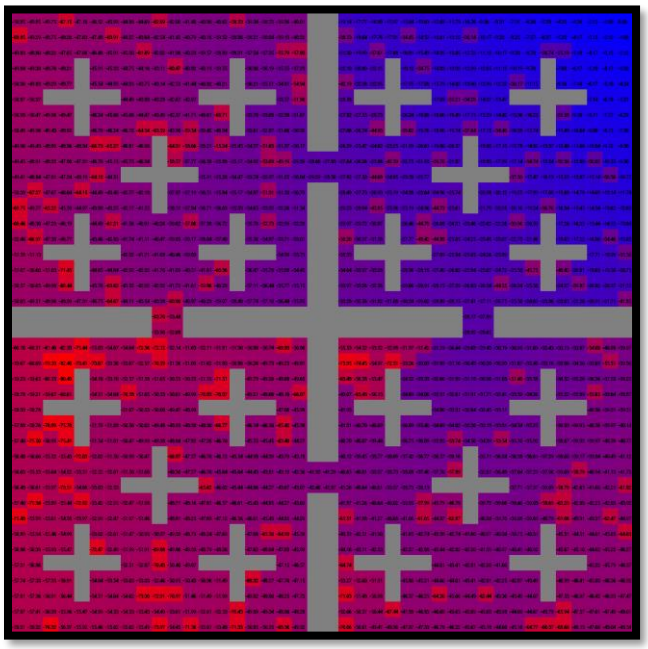

Fig 2.2. Complex Grid World Model VI Converged State Value

which satisfies with the MDPs definition so multiple iterations can help improve to estimate this value until all these future reward (estimated values) in every state converge. The experiments were done in BURLAP, with ValueIteration(gamma = 0.99, maxDelta = 0.001, maxIteartion = 1000)

In table 1, we could see that in Easy Grid World experiment, it took a very small time, just about 77ms to converged but it also needed 27 iterations to converge to the best policy with a value of 88.

Also see the figure 2.1, the number in the block was the expected future reward from the current state and the arrow meant the best policy that could choose from the current block. It was obvious to find the optimal policy calculated by Value Iteration was the route drawn in white marker.

For the Complex Grid World experiment, we could see that it took 1460.84ms to converge, with a converged value of -118.

However, it was interesting to see that the expected future value from the start state was far less than the terminal reward – total steps even though $\Upsilon = 0.99$. That was because this expected value was only considered to pick the optimal policy instead being chosen as the real reward. Moreover, because the possibility of each action the agent took was not deterministic, this expected future reward should not equal to the real reward by the best policy. However, they converged at last but it had to take a much larger iteration number to converge comparing to the deterministic actions.

2.2. Policy Iteration Analysis

Different from Value Iteration which aims to find the best value directly, Policy Iteration tries to manipulate the policy directly to find the best policy to get the best total reward. In policy Iteration, we try to estimate the reward of an action at a state instead of the state value updated by neighboring states. As the result, once we know the value of each state under a current policy, we can determine it by solving the linear functions by executing the policy to gain the value. Until there are no improvements for change of policy, it converges to guarantee the best policy with best reward.

Problem	Algorithm	Converged Iterations	Used Time(ms)	Converged Value
Easy Grid World	Policy Iteration	4	243.28	88
Complex Grid World	Policy Iteration	25	28821.16	-101

Table 2. Value Iteration Data

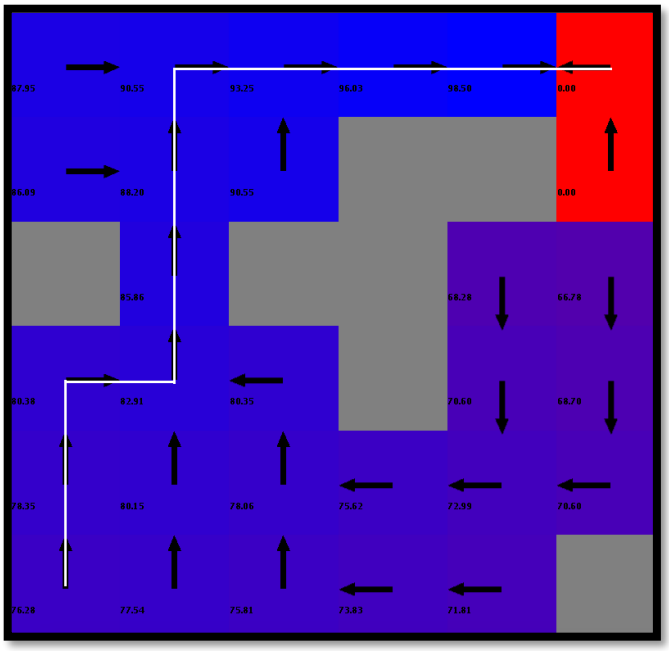

Fig 2.3. Easy Grid World Model PI Converged State Value

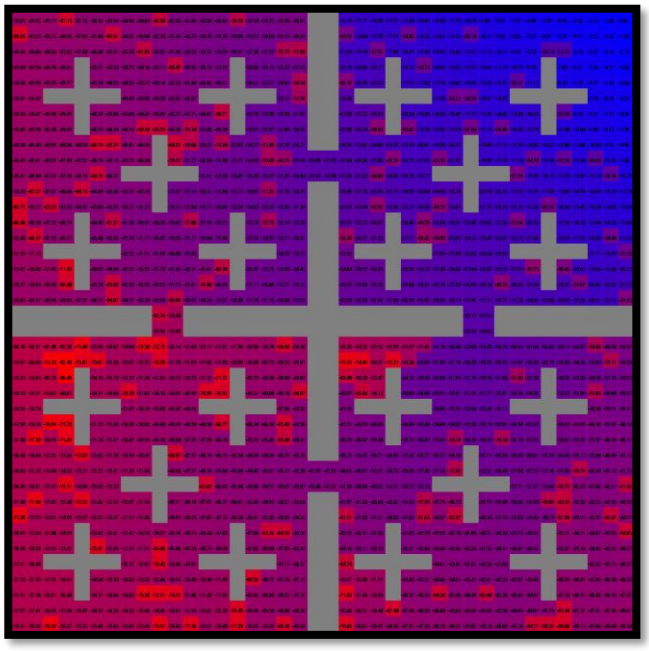

Fig 2.4. Complex Grid World Model PI Converged State Value

The experiment was done in BURLAP, with PolicyIteration(gamma = 0.99, maxDelta = 0.001, maxEvaluationIteartion = 100, maxPolicyIteration = 100)

In table 2, we could see that Policy Iteration only needed 4 iterations but it took about 243ms to converge in the first Easy Grid World example. For the Complex Grid World, it seems crazy that it took nearly 30000ms to converge! And it converged to a value of -101, much better than the VI.

In figure 2.3, Also the same as mentioned above, white marker meant the best policy by Policy Iteration. However, it was obvious that the result of Policy Iteration in Easy Grid World was identical as Value Iteration result with figure 2.1, with all the converged values in each state (each block) being the same.

But in figure 2.4 for Complex Grid World, it seems very similar to the figure 2.3 for Easy Grid World, but it was a bit different because they got a different converged value.

Detailed comparison and explanation will be in the next section.

2.3. Value Iteration vs. Policy Iteration

Based on table1 and table2, we could see that Value Iteration converged with a larger number of iterations but it could converge in a much faster speed. On the contrary, Policy Iteration only need a small number to converge, but however, each of the iteration would take a much longer time.

As mentioned above, the converged values of both VI and PI in Easy Grid World Domain were 90. They were the same because the model was easy with a small amount of states all of which could be accessed before converged. In the Complex Grid World Model, it was obvious that PI had a converged Value better than VI.

The reason was that Value Iteration is based on Dynamic Programming, which means all of the states of values are computed and stored. As the result, they are updated by such a DP equation:

$$V_{t+1}(s) = max_a(R(s) + \sum_{s'}(P(s'|s,a) \cdot V_t(s')))$$

Where $V_t(s)$ is the current best value of state s in time t, R(s) is the immediate reward in state s. So the best policy could be found according to the optimal value of $V_{t+1}(s)$:

$$\pi^*(s) = argmax_a(\sum_{s'}(P(s'|s,a) \cdot V_t(s')))$$

But in Policy Iteration, the different point was that the value equation was based on the policy instead of action. Therefore, the DP equation was:

$$V_{t+1}(s) = R(s) + \sum_{s'}(P(s'|s,\pi(s)) \cdot V_t(s'))$$

Notice the difference between ***π(s)*** and ***a***, and it means that ***π(s)*** needs another iteration to get its best action, called the optimal policy in current policy iteration:

$$\pi_{t+1}(s) = argmax_a(\sum_{s'}(P(s'|s,a) \cdot V_t(s')))$$

In this equation, we can see it as an inner direct “value iteration” to find the best action in the current state. As the result, Policy Iteration could converge with a small number of policy iterations but actually it needs a lot of inner iterations in each policy iteration. That was why it converges with a larger time compared to Value Iterations. However, the advantage is that since it is based on the optimal policy, it could usually get a better result than VI.

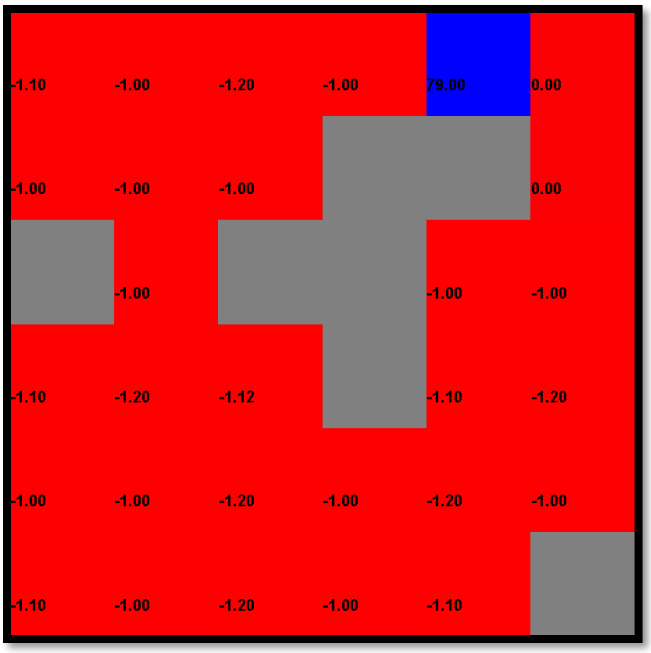

Fig 2.5. VI State Value (Iteration = 1) of Easy Grid World

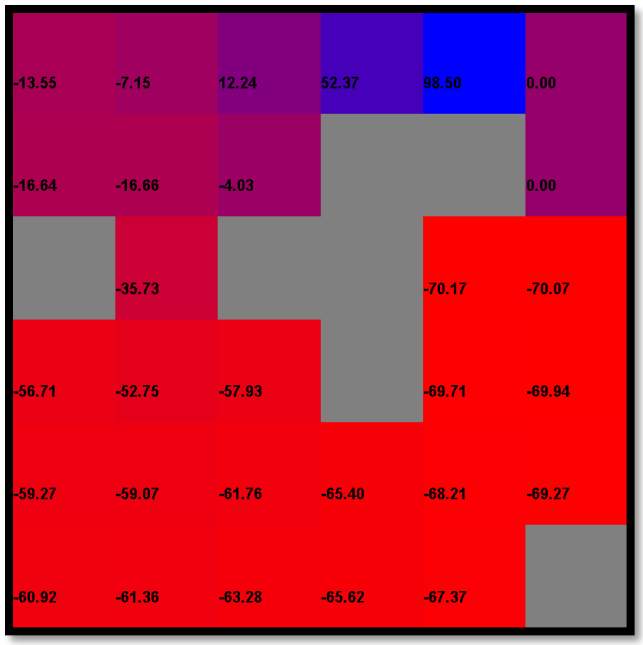

Fig 2.6. PI State Value (iteration = 1) of Easy Grid World

Here is an example of how the different equation makes a different result. Figure 2.5 and 2.6 are the state value after first iteration of VI and PI in Easy Grid World example. In VI, the state value only changed once according to its neighbors but in PI, all of the state values had changed to get the "local" optimal policy.

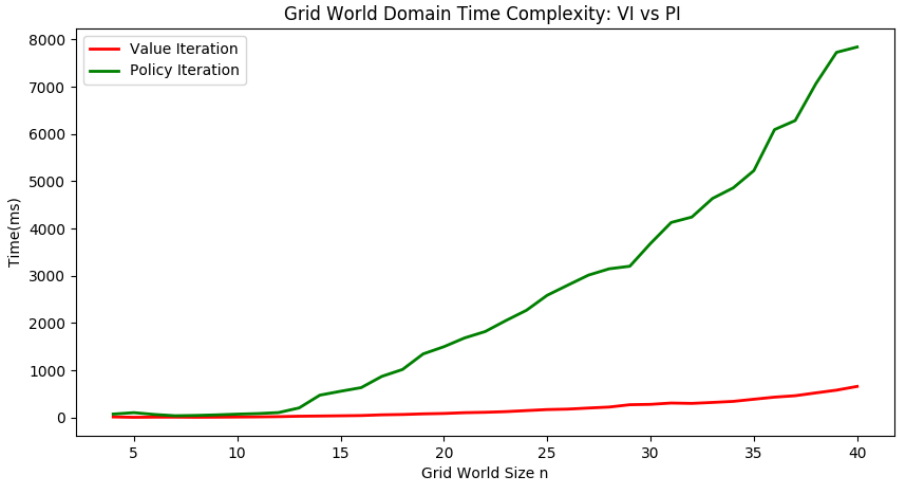

Fig 2.7. VI vs PI: Grid World Domain Time Complexity Related to Size

Here are more experiments about Value Iteration vs Policy Iteration. This experiment was based on the primitive Grid World Domain with different size n, which means all the states are accessible, all the reward of states are -1 and the agent needs to walk from bottom right corner to upper right corner.

In figure 2.7, we could see that as matrix size increased, the time of PI would increase very fast, nearly close to exponential curve, but on the opposite, the Value Iteration curve remained very small and didn't move at all! And in figure 2.8, the iteration times of VI was linear related to matrix size, but iterations of PI only kept a small amount of increase. These was also the result of their different DP equation. However, it was this equation difference that makes them show different performance to the same problem.

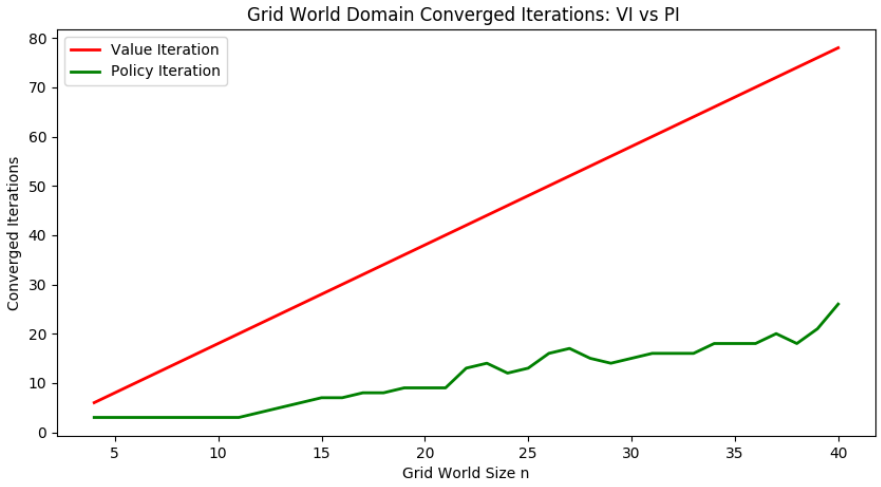

Fig 2.8. VI vs PI: Grid World Domain Converged Iterations Related to Size

3. Reinforcement Learning Algorithms Analysis

3.1. Q-Learing Description

By learning from Professor classes and WIKI, we can conclude that Q-Learning is used for a single agent to find a best action-selection policy for MDPs problem. It works by learning a value function estimated by taking different actions that gives an expected utility by taking these actions in the current state. In other words, the agent tries to learn this optimal policy from its history of its actions with its states. Such a history of the agent is a sequence of state-action-rewards with finite tuples of $<(s_0\ a_0\ r_0), (s_1\ a_1\ r_1)\ldots>$, where s_i means the state, a_i is the action and r_i is the reward by this action a_i in state s_i. Therefore, we can treat each tuple of history as an experience $(s_i\ a_i\ r_i\ s_i')$, where s_i' is the next state after taking this action.

By taking such a transformation, we can have the agent maintains a table of Q[s, a] to represent its current estimate value $Q^*(s, a)$ instead of remembering all the states and actions. By propagation to update the current reward by adding the discounted estimated future value, and we can have:

$$Q[s,a] = Q[s,a] + \alpha(r + \gamma MAX_{a'} Q[s', a'] - Q[s,a])$$

Where α is learning rate and ϒ is the discount factor for future reward.

As the result, we can know that Q-Learning is a model-free algorithm which means that we needn't know the actual model including all its states and actions. All we need to know is the agent's interaction history with the model. And because it learns the optimal policy without mattering what policy it is carrying out, it is called an off-policy learning method.

However, all the MDPs problems we chose had clear connections with states and actions that we had already know, so it may not show the strength of Q-Learning or Reinforcement-Learning.

3.2. Q-Learning Result and Analysis

In figure 3.1, we could see that in Easy Grid World problem, the Q-Learing learner started to converge around 40 iterations but its best reward value was always less than VI and PI.

In figure 3.2, it was difficult to tell when it converged but comparing to the VI and PI algorithms, but we could know that it was around 400 iterations. However it was with the worst reward all the time, which means it learned the worst policy although it was close to the value of VI and PI at last but there was still a difference between them.

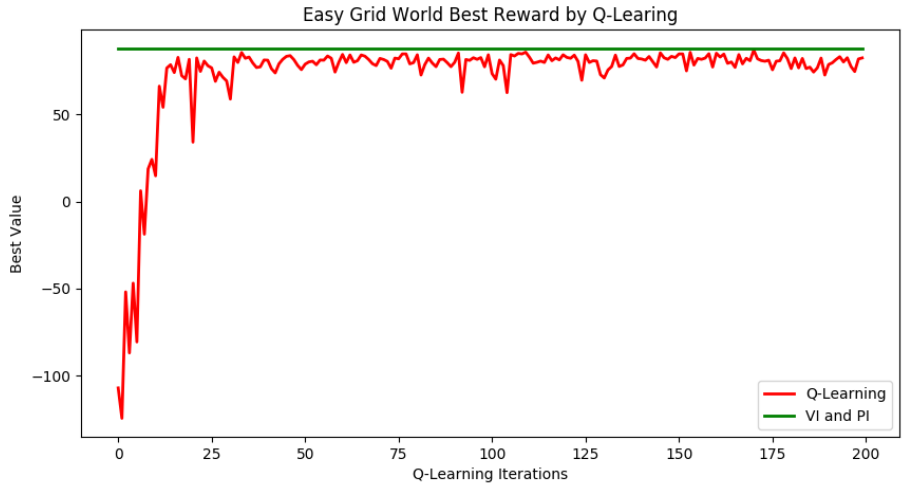

Fig 3.1. Q-Learning Learning Curves of Easy Grid World

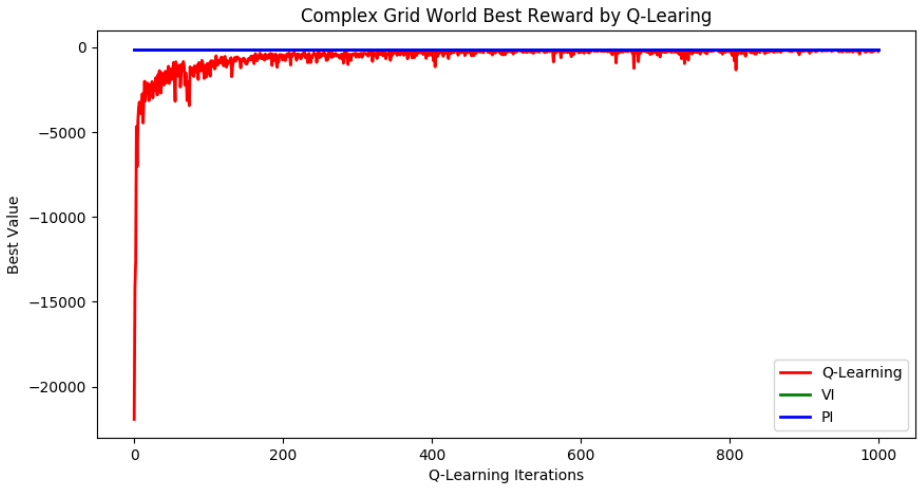

Fig 3.2. Q-Learning Learning Curves of Complex Grid World

But unlike Value Iteration and Policy Iteration, which could always get a stable, or fixed value, the total value decided by the optimal policy of Q-Learning was very “unstable” just like the curves in this figure which would jump up and down. That was because Q-Learning could help the agent to learn the important features of the map, but could not tell what the real map looked like to the agent. As the result, the agent could only choose the best policy according to what it had learned by the Q-Learner.

When we check the figure 3.3 for the Easy Grid World example, we could see there was some contradiction in the route of the optimal policy so it got a result of 80, not as good as the result of VI and PI.

In the figure 3.4, it was also difficult to tell the difference between figure 2.2 and 2.4 of VI and PI. But actually, its upper left part was with more purple, which means this part had a worst reward comparing to the other part. That was because the agent was exploring from the bottom right to the upper part.

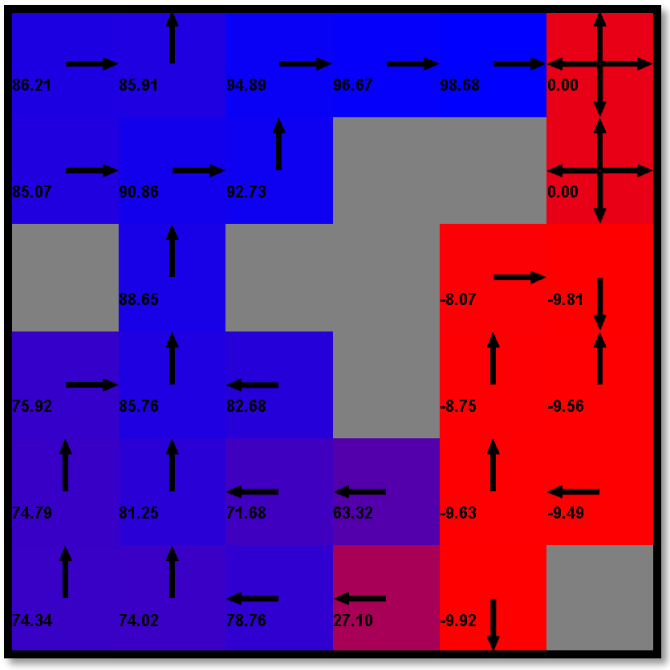

Fig 3.3. State Value (Iteration = 200) of Q-Learning for Easy Grid World

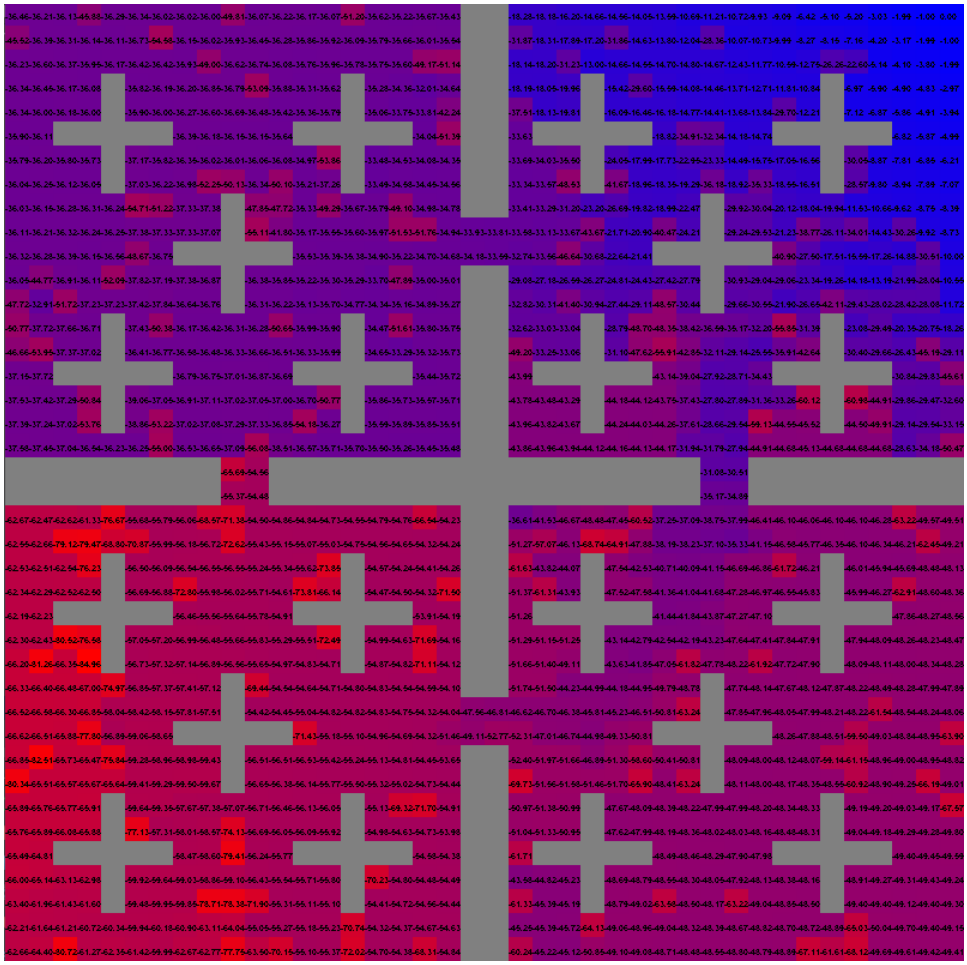

Fig 3.4. State Value (Iteration = 200) of Q-Learning for Complex Grid World

3.3. Comparison with Value Iteration and Policy Iteration

Q-Learning was very interesting because it was really learner instead of the iteration to get the best reward according to the Dynamic Programming equation. As the result, unlike Value Iteration and Policy Iteration, which needs to know all the states and their entries to run the equation iteration, Q-Learning's strength is that it needn't know what the exact model looks like for the agent but it could teach the agent to know the important features and at last, the agent can get use of this feature to select the optimal policy.

Moreover, it was got a worse result than Value Iteration and Policy Iteration because the agent is always in an exploring state, which means it doesn't know all the information of the model. However, since it just updates

4. Summary and Arguments

4.1. Summary

Value Iteration: Based on DP equation to update the value from its neighbors. Fast converged time with a very good policy and value. However, it needs to know the whole model of the problem, including all the states and actions and their value.

Policy Iteration: Based on DP equation to update its policy by controlling an inner iteration to get the "local" optimal policy. It usually gets the best result but like VI, it needs to know the whole model.

Q-Learning: A "real" learner for Reinforcement Learning. It needn't know the whole clear model because the learner can help the agent to learn the features of it. However, it will get the worst and unstable result compared to the other two algorithms because the agent is always learning the information from the map and updating its optimal policy according to what he has.

4.2. Arguments and problems

This homework was with a lot of fun! I tried to write a new domain based on the existed grid world domain but it failed. However, I learned a lot even though I only used two similar problem and there were still a lot of interesting problem to explain!

However, here are some questions about this assignment and BURLAP. In BURLAP, it always sets the terminal state with a reward of 0 but why it should always be 0. Even though I modified the codes to make multiple terminals with different non-zero reward, why it designs to be 0? Second, in grid world domain, we could see that it used current reward R(s) instead of R(s, a) and R(s, a, s'), I know they are the same but why they are the same result? Whether there are some cases that they are not equivalent?