Assignment 4: Markov Decision Processess

Shaikh Shamid

April 19, 2015

1 Introduction

In a markov decision process(MDP) the agent senses a set of distinct states of its environment and a set of actions that it can perform, and unlike the supervised and unsupervised learning cases discussed before [1, 2], the trainer provides a sequence of immediate reward values as the agent executes its sequence of actions in order to maximize the total reward. In this assignment we will explore and learn from two different markov decision processes. We will analyze one MDP with a smaller number of states and another one with larger states.

2 Markov Decision Processess (MDP)

In MDP, The task of the agent is to learn a policy for selecting its next action based on the current observed state and action, and not on earlier states or actions, called the Markovian property. If the environment has Markov property then its one step dynamics allow us to predict the next state and expected reward given the knowledge of current state and the action taken in the current state. The predictions made on the basis of the Markovian property are as good as the predictions made with the knowledge of the complete history up to that time.

2.1 Value Iteration Algorithm

In the value iteration algorithm we iteratively calculate the utility function at time step t+1, which is the the maximum of all the actions associated multiplication of transition probability and the utility at step t. Initially the utility of a state is the reward value of that state. Next we calculate the optimal policy from the optimal utility function. The optimal policy is the argument maximum among actions over the multiplication of transition probability and the optimal utility. So the value iteration is an optimal solution to the Markov Decision Problem.

2.2 Policy Iteration algorithm

In the policy iteration algorithm we initialize a policy randomly and calculate all the utilities associated with that policy. We then update policy based on policy formula and repeat the above steps until we get a stable policy.

3 Q-Learning

Watkins Q-learning [3, 4, 5] is a model-free method which is very easy to implement. Q-learning works by estimating the value of state-action pair, $Q(s, a)$. The value $Q(s, a)$, known as Q-value, is defined to be the expected sum of future reinforcement(penalty) obtained by taking action a from state s and following an optimal policy thereafter with some small learning rate. Once these values have been learned, the optimal action from any state is the one with the lowest Q-value. Learning rate determines how much current Q-value is changed on the basis of new observation. Learning rate can take value between 0 and 1. For Q-values to converge to optimal values, it is required that every state and action pair is explored sufficient number of times, ideally infinite number of times. Thus an exploration policy must be used for choosing an action given a state. Some of exploration policies used for this experiment are:

- ϵ-greedy: Select best action with probability $1 - \epsilon$ and choose random action with probability ϵ.
- Changing start state: Once goal is reached, reset to random state for next iteration.

4 Choosing two MDPs

I picked a variation of the grid-world example given in the lecture. The first problem has few states and is easy to solve, and the second one has larger number of states. The experimental setup consists of an agent moving in a discrete state space represented by a maze where each state is represented by a cell in the maze. The maze contains terminal states represented by bright cell in figure 1, and obstacles are represented by dark walls. The maze is bounded on all sides by walls. If the agent tries to transition from one state to another and hits a wall instead then the agent is penalized and stays in the same state. There is a path cost associated with every transition that the agent makes from one state to another. The aim of the agent is to find that path to the goal state which has least cost associated with it.

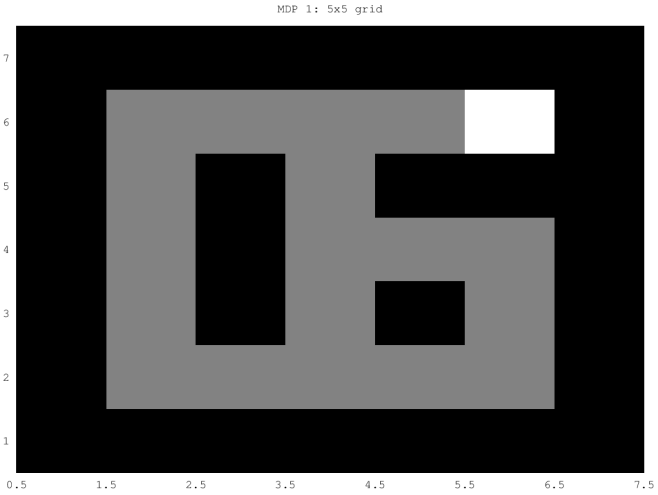
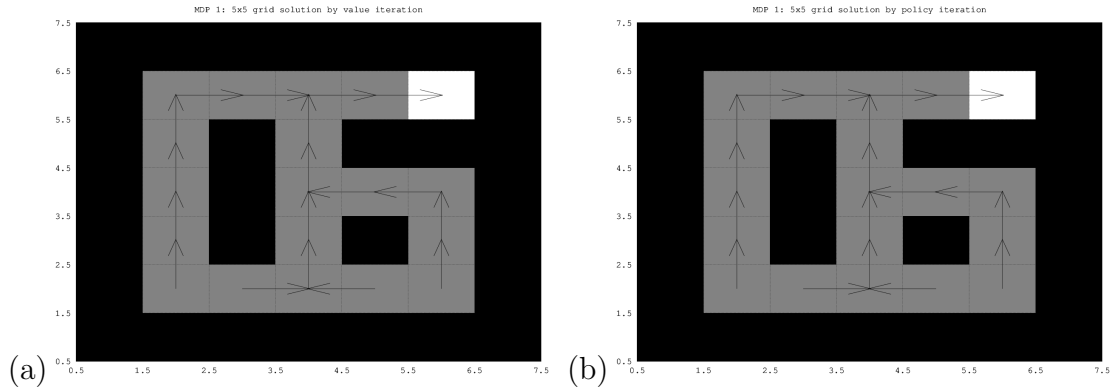


Figure 1: MDP1: 5x5 grid-world problem.

5 Implementations and Discussions

5.1 Value and Policy Iteration

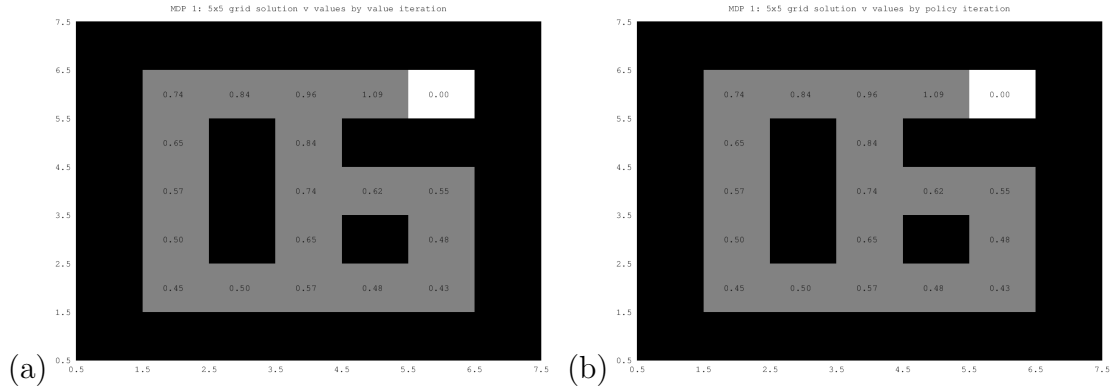
Both algorithms calculate the same optimal policy for the small grid world problem. The optimal policy for both algorithms is shown in figure 2.



(a) (b)

Figure 2: 5x5 grid solution (a) by value iteration algorithm. (b) by policy iteration algorithm.

Certainly the computed optimal policy is a correct optimal policy. Table 1 shows the run-time and the iterations each algorithm need to compute the optimal policy. Value iteration needs around double more iterations and execution times. Because policy iteration solves a linear system, and does not calculate the policy in a greedy manner, it uses more time per iteration step, that is why there is a difference between the factors of iterations and execution time.



(a) (b)

Figure 3: 5x5 grid solution final v values for (a) value iteration algorithm. (b) policy iteration algorithm.

Figure 3 shows the V-values, that is the expected reward of each state following the optimal policy. It is interesting to see that the closer the state to the goal the higher the V-value. In other words, the V-value propagates from the goal, This is because the faster the agent reaches the goal, the higher its reward should be. It is also interesting that the V-value of tiles that are adjacent to the goal is 1.09 instead of 1. This is because it is the expected reward following the optimal policy and there

Algorithm	Problem	Iteration	Time (s)
Value Iteration	5x5 grid	19	0.0064
Policy Iteration	5x5 grid	3	0.0034

Table 1: Comparison of Algorithm Performance for small grid-world problem.

is a 10 percent chance not to execute the action the agent want, leading to other ways to the goal. As expected If we set the error probability to 0, it becomes 1.

I also examined which effect to optimal policy different choices of the probability p have, i.e. the probability to execute the intended action. Figure reffig:g5solp5 shows the result for p = 0.55. the solution at the bottom left tile goes right first instead of going up, and the bottom right tile goes left first instead of going up, but the solution is still optimal because the distance from the tiles to the goal is same whether the agent goes either way. Interestingly, for the case where probability to execute the intended actions is just slightly above average, the problem seems to be so simple, the iterations have been found be almost half compared to p=0.9 case, and so does the execution time. Also the results never change and also converge to a optimal solution.

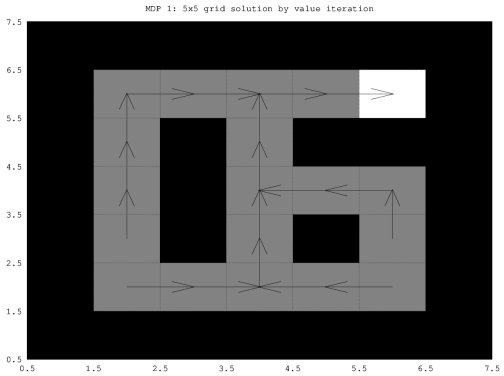


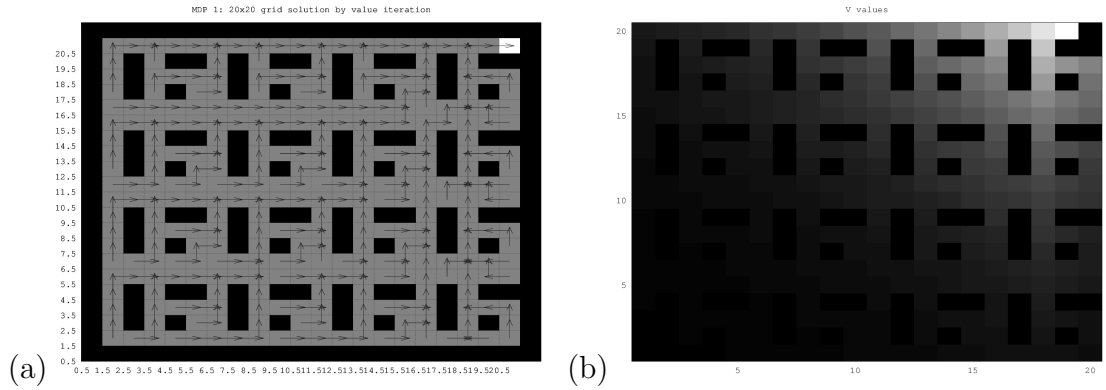
Figure 4: 5x5 grid solution for p=0.55.

For the second 20x20 grid problem the optimal policy as well as the V-values heat map (white color means high V-value, black means low V-value) in value iteration algorithm is given in figure 5. Again figure 5 (b) shows that the closer the state to the goal the higher the V-value. In other words, the V-value propagates from the goal, This is because the faster the agent reaches the goal, the higher its reward should be.

For the second 20x20 grid problem the optimal policy as well as the V-values heat map (white color means high V-value, black means low V-value) in policy iteration algorithm is given in figure 5.

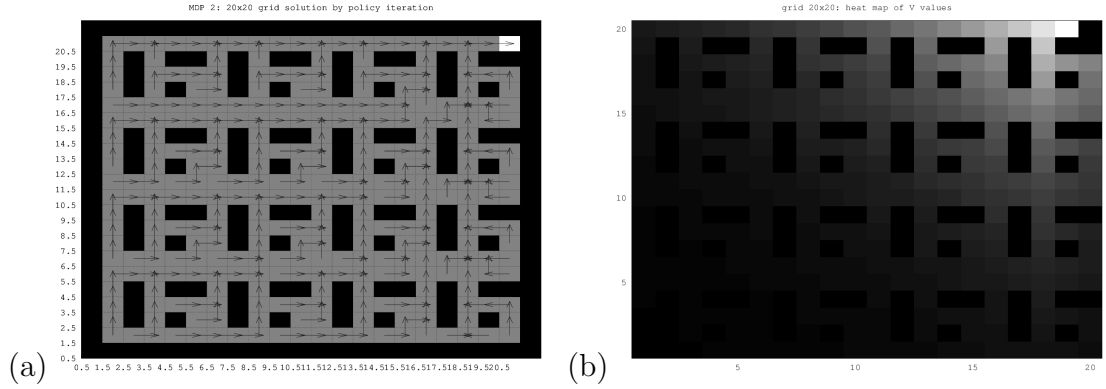
In order to study the scaling behavior of algorithms with respect to the grid size, I implemented the problem for grid size 10x10 and run both algorithms again.

As can be seen in figure 7, if we start from grid 5 both algorithms iterations scale more or less linearly with respect to the square of the input size, although the factor of value-iteration algorithm is higher than the one of the policy-iteration algorithm. I assume that this because the policy algorithm solves for V-values or respectively the Q-values simultaneously keeping the optimal policy fixed, instead of the solving for them in a iteratively greedy manner like the value-iteration algorithm does. This



(a) (b)

Figure 5: (a) 20x20 grid solution and (b) corresponding V-values heat map in value iteration algorithm.



(a) (b)

Figure 6: (a) 20x20 grid solution and (b) corresponding V-values heat map in policy iteration algorithm.

should lead to fewer iterations because of increased stability, but also each iteration takes more time. This can be seen in the figure 7(b). For a grid size of k = 10 value iteration actually performs faster than policy iteration, also it uses more iterations. For higher values the instability of the value-iteration approach seems to outweigh its faster iterations. The figure also shows that the algorithms scale quadratically with respect to the square of the input size, i.e. they scale linear in time with respect to the input size.

Algorithm	Problem	Iteration	Time (s)
Value Iteration	20x20 grid	47	0.090
Policy Iteration	20x20 grid	5	0.041

Table 2: Comparison of Algorithm Performance for second grid-world problem.

5.2 Q-learning

In the Q-learning algorithm we assume the transition and reward models are unknown, the transition and reward models we used to simulate the environment when the agent executes an action. The optimal policy for both problems in the Q-learning algorithm turn out to be same in the end. Truly It is an optimal policy it has to

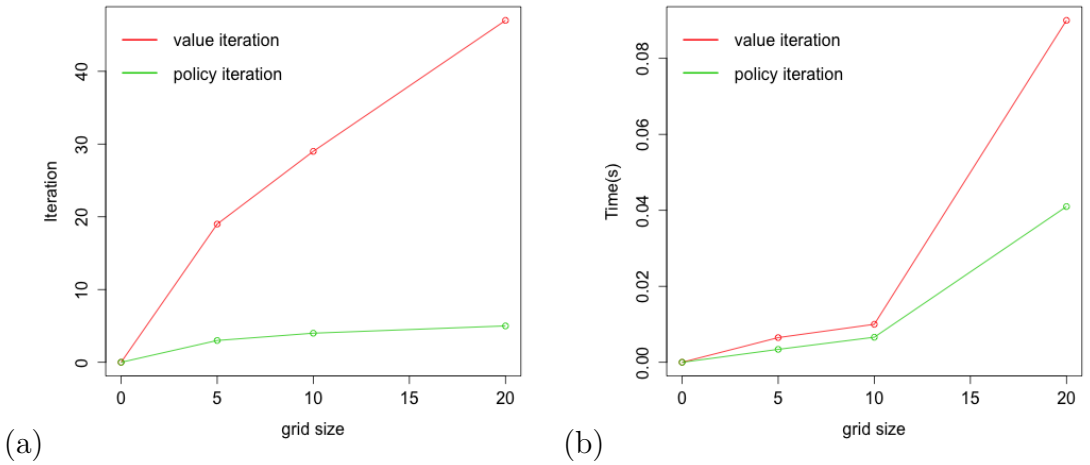


Figure 7: (a) Iteration vs grid size and (b) time vs grid size.

be, but I had to spend a lot of time for that. First of all, with the reward values I have been working with so far, the Q-learning algorithm was giving me negative Q values. I had to change the reward values from 0 to 1, -1 to 0 and 1 to 2. Secondly, a lot of research with learning rate and ϵ values in ϵ-greedy exploration process, that I will be describing below.

Regarding the ϵ-greedy process, there was a tradeoff between exploitation (lower epsilon/less random actions) and exploration (higher epsilon/more random actions). By having a higher epsilon the convergence was more uniform throughout the grid. Each iteration was longer but need less iterations to converge to the correct value. I was able to go such a lower value of epsilon (0.005) that was insured a more local convergence meaning that the Q-values on one path that leads to the goal state converges quickly to certain values but the rest of the grid was stay unexplored longer time for the large MDP problem.

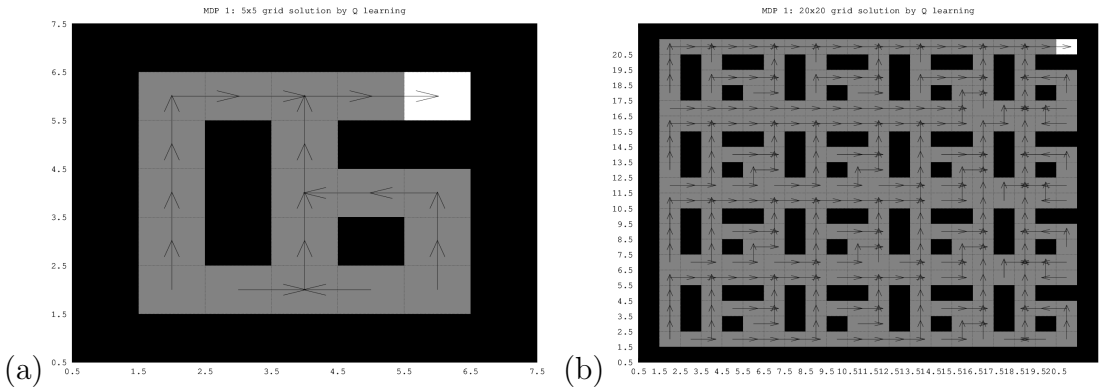


Figure 8: Q learning result for (a) 5x5 grid size and (b) 20x20 grid size.

What I observed is that for a high learning rate the agent improves its policy faster but does not settle down to an optimal policy and keeps oscillating between a near optimal policy and a bad policy. Whereas, the agent with low learning rate it settles to the optimal policy slowly but steadily. This is because of the fact that the agent is more sensitive to the reinforcements it receives from the environment if

its learning rate is higher.

A slight modification I did to Q learning so that it can capitalize on the advantages of both low and high learning rate. This can be achieved by decaying the learning rate as the number of iterations increase by following this equation: $1/log(n+2)$, where n is the number of iteration.

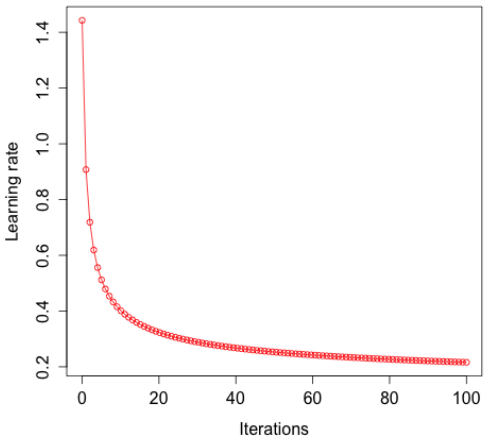


Figure 9: Plot of learning rate vs iterations in Q learning algorithm.

According to this scheme the agent interacts with the environment with a very high learning rate initially so that it can quickly settle down to a near optimal policy. As the agent?s interactions with the environment increase the learning rate decays and the agent now makes only minor modifications to its near optimal policy to obtain the optimal policy. This scheme is guaranteed to asymptotically converge to the optimal policy. The learning rate under the decaying learning rate scheme is shown in Figure 9.

6 Conclusion

We have thus analyzed Markov decision processes and the reinforcement learning algorithms mainly value iteration, policy iteration and Q learning. The learning rate certainly plays an important role in the convergence of the Q learning algorithm and that the decaying learning rate scheme performs better than fixed learning rate scheme for Q learning.

References

[1] `https://github.com/sshamid/ML_projects`

[2] `https://github.com/sshamid/ML_projects2`

[3] T. M. Mitchell, Machine Learning. McGraw Hill, 1997.

[4] L. P. Kaelbling, M. Littman, and A. Moore, 'Reinforcement learning: A survey,' Journal of Artificial Intelligence Research, vol. 4, pp. 237-285, 1996.

[5] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. London, England: The MIT Press, 1998.