Markov Decision Processes

CS 7641: Machine Learning Assignment 4

By Divya Paduvalli (GT ID: dpaduvalli3/902913716)

Abstract

This report is divided into two sections. In section 1, two interesting Markov Decision Processes (MDPs) namely: Grid world with small and large number of states were analyzed. In section 2, each of the two MDPs were solved and analyzed using value iteration, policy iteration, and Q - learning (my favorite algorithm). The analysis results presented in this report was accomplished using Eclipse and Burlap.

Section 1: Introduction to Reinforcement Learning and Markov Decision Processes Problems (MDPs)

Introduction to Reinforcement Learning

Reinforcement learning is a type of machine learning algorithm where the model learns from delayed reward. Here delayed reward means the feedback on the overall performance of the model is received several steps later after the decisions made. A classic example to explain this concept would be playing the game of tic tack toe, where you would loose by putting an X or an O in the wrong place. After you lose, you get a feedback on where you did well and where you did poorly. So essentially you are playing a game without knowing any rules or at least knowing how you win or lose but being told occasionally, you have won or lost. Reinforcement learning allows software agents and computers to automatically identify an ideal behavior with a specific context to increase or maximize its score or performance (2). Within reinforcement learning, there are several types of algorithms or approaches and one such approach that is analyzed in this report is called the Markov Decision Process (MDP). In MDP, decisions are taken in a grid world like environment which consists of the following:

1. **State:** a set of possible state grids an agent can be in. For example, a person named John (our agent) can travel between three different places, namely work, home, and school. Here each of these three locations are called a state denoted by S.
2. **Actions:** a set of possible actions that an agent can take being in a state. For example, different directions that John might take to reduce his commute time to get to his destination place. This is denoted by A or A(s).
3. **Models/Transition Models:** yields an action's effect in a state. For example, which place is John likely to travel. John might have a 0.5 chance of going to work, 0.2 chance of going to school, and 0.3 chance of going to home (total probability equals 1). This is denoted by P (s'| s, a). These probabilities are typically selected by observing a process happening repeatedly before making conclusions about the probabilities.
4. **Reward:** a reward function with a real value of being in a state. For example, John gets paid for being at work. This is denoted by R(s, a).

5. **Policy:** a solution to the MDP task. The goal is to find the "optimal" policy. For example, John's policy could be to optimize his overall commute distance to save time and gas (1).

MDPs Problem Statement

The two MDPs that I have selected are the classic grid world problems with small and large number of states. In the grid world problem, the agent lives in a grid and the walls block the agent's path. The agent is supposed to explore around the grid to reach its destination grid. Along the agent's path there could be grids that contain rewards, obstacles that the agent should avoid, and blocked grids that acts as wall which prevents the agent from entering in it. The agent starts its exploration at an initial state. The agent can move up (north), down (south), left (west), and right (east). A simplistic application of this problem would be playing the famous arcade game of Pac-Man where the game is bounded within a defined grid which contains various rewards in the form of fruits and obstacles in the form of walls or other characters.

X	0	0	1	1
S	1	0	0	S
1	M	0	0	1
1	M	1	0	0
L	0	1	L	G

Small MDP Grid Representation

X	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
L	0	0	0	M	M	M	1	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1
L	0	0	0	M	M	M	0	0	1	S	S	1	S	S	0	1	S	1	S	S	1	S	S	1
L	0	0	0	M	M	M	1	0	1	S	S	1	S	S	0	1	S	1	S	S	1	S	S	1
L	L	0	1	M	M	M	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
L	0	0	0	M	M	M	0	0	M	M	M	0	0	0	0	S	S	S	S	S	S	S	S	1
L	0	0	0	M	M	M	1	0	M	M	M	0	S	0	0	S	S	S	S	S	S	S	S	1
L	M	0	0	M	M	M	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	M	M	M	0	1	0	1	0	0	1	0	1	0	0	S	1
1	M	M	S	0	1	1	1	0	M	M	M	0	1	0	1	0	0	1	0	1	0	0	S	1
1	0	0	S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	S	0	1	1	1	0	M	M	M	0	1	0	1	1	0	1	S	S	S	0	0	1
1	1	0	S	0	1	1	1	0	1	1	1	0	1	0	1	1	S	1	1	1	1	0	0	1
1	1	0	S	0	M	M	M	0	1	1	1	0	1	0	1	1	0	1	1	1	1	0	1	1
1	1	0	0	0	L	L	L	0	1	1	1	0	1	0	1	1	0	1	1	1	1	0	0	1
1	0	L	1	0	L	L	L	0	0	0	0	0	0	0	0	0	S	S	S	S	S	0	1	1
1	1	0	1	0	L	L	L	L	L	L	L	L	L	1	0	S	S	S	S	S	S	0	0	1
1	1	0	1	0	L	L	L	L	L	L	L	L	L	1	0	0	0	0	0	0	S	0	S	1
1	1	0	1	0	L	L	L	L	L	L	L	L	L	1	0	1	1	1	1	0	S	0	S	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1	0	1	1	1	0	1	1	0	0	L	L	0	0	S	1
1	1	1	0	1	0	1	0	1	1	0	1	1	1	0	1	1	0	0	L	L	0	0	S	1
1	0	0	0	M	0	M	0	M	M	M	M	M	M	0	M	M	M	M	M	M	M	0	0	1
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	0	0	0	0	0	0	0	0	0	0	L	0	0	0	0	0	0	0	0	0	0	0	0	G

Large MDP Grid Representation

For my small MDP, I have selected a grid size of 5 * 5 to start with very low complexity and for my large MDP, I have selected a grid size of 25 * 25. In the above visual representation, X is my agent and it starts at the upper left corner, 1 represents wall, 0 represents the path without any obstacles, S, M, and L represents small, medium, and large hazard with negative reward values, and G represents the final goal. The exact parameter values will be discussed more in section 2.

Why are the MDP problems interesting?

The grid world problem might look and sound simplistic, but this can be used in many real-world applications. As an example, Toyota Prius has a feature called Intelligent Parking Assist. This feature automatically parks your vehicle in the desired parking lot. It can do several different styles of parking such as parallel, angular, perpendicular, and reverse parking. It uses a type of grid world where the size of each grid can be a little bigger than the size of the vehicle but equivalent to the size of an average parking lot. The vehicle is the agent. The vehicle starts from an initial state or location or grid and automatically slowly moves to the destination parking lot. The vehicle has sensors to detect obstacle around it like cars, walls, light post etc. The vehicle is only bound to move in certain directions given the size of a parking lot. Apart from Prius, the grid world problem is also used in automatic self-driving vehicles where the vehicle, acting like an agent, will have to make a series of decisions based on its path and obstacles to navigate to its destination. For example, if there is rush hour traffic, the self-driving car will use the sensors around its car to detect obstacles in the form of other vehicles and will automatically reduce its pace.

In addition to automobiles, the grid world problem is also used in GPS applications such as Uber, Lyft, Google maps etc. that aid in route planning. In Google maps, we have the option of setting a start and a destination location. Based on the time and the day, Google maps uses either historical data or real time data to predict the total destination time. For example, let's say you are in a traffic jam, if most people around you are using Google maps, Google uses location service information from these devices (which is detected by the GPS satellites orbiting the earth) which is then transferred to the cell towers and then to the Google servers. The servers uses machine learning algorithms to make a conclusion that since there are a lot of vehicle that are in the same spot (around you) moving very slowly, thus there should be heavy traffic and the path with the heavy traffic will be highlighted in either red or yellow. This lets other commuters know that this path, that you are taking, might not be the best path to reach their destination due to heavy traffic congestion. Since Google maps constantly keeps recording our commute data daily from all around the world, their algorithms almost always make accurate predictions on route planning. This just shows how powerful and insightful big data is, and this proves again that the data is the "king" in machine learning and not the algorithm.

Section 2: Solving the MDPs using Reinforcement Learning Algorithms

MDPs Parameters Tuned for the three Reinforcement Learning Models					
MDP Problem Type	**Grid size**	**Model**	**Actions**	**Obstacles**	**Rewards**
Small MDP Problem	5 * 5	Gamma = 0.95, MaxDelta = 0.001 (value and policy iterations), Q - Value = 0.3, Q-Learning Rate = 0.1	North, South, East, West	Walls	small hazard = -0.5, medium hazard = -1.0, and large hazard = -1.6, path cell reward = -0.1, end goal reward = +50
Large MDP Problem	25 * 25	Gamma = 0.95, MaxDelta = 0.001 (value and policy iterations), Q - Value = 0.3, Q-Learning Rate = 0.1	North, South, East, West	Walls	small hazard = -0.5, medium hazard = -1.0, and large hazard = -1.6, path cell reward = -0.1, end goal reward = +50

The above parameters were set for the two MDPs to experiment with the three reinforcement learning algorithms. Here gamma refers to a discount factor (should be less than 1) that the MDP uses. I set the gamma to 0.95 to ensure that the total expected value of the MDP converges to a finite solution. Max delta is the maximum change in the function for value and policy iteration. I just used the default value of 0.001. For the Q value and Q learning rate, I went with the default values of 0.3 and 0.1 because after conducting several experiments, the end results did not change significantly. There is a combination of both positive rewards of +50 for reaching the end goal and three negative rewards of -0.5, -1.0, -1.6. I have also included a reward of -0.1 for the empty path cells to incentivize the agent to not aimlessly wander around and be efficient in reaching its goal quickly.

**Due to page limitation, the size of the graphs in the below sections have been reduced. If you would like to view the graphs in good quality, please click on this excel spreadsheet.

Value Iteration

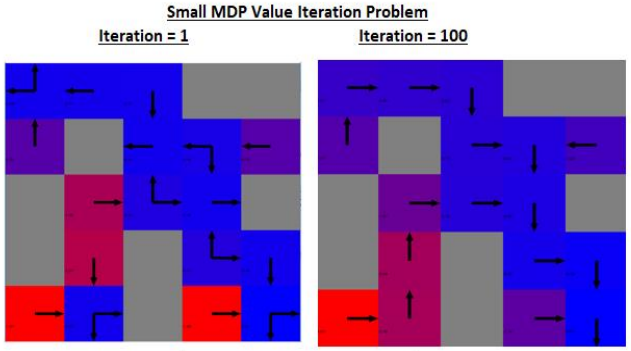


Figure 1.1

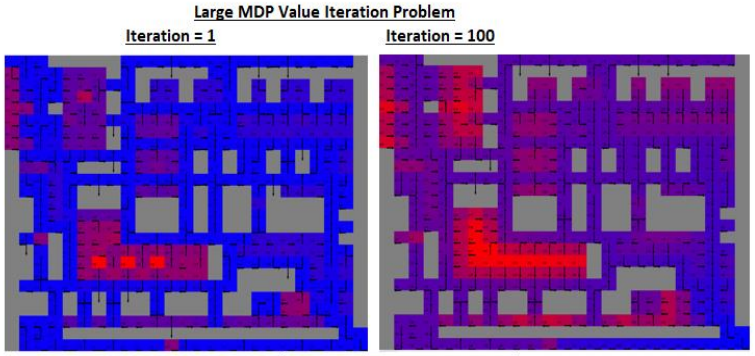


Figure 1.2

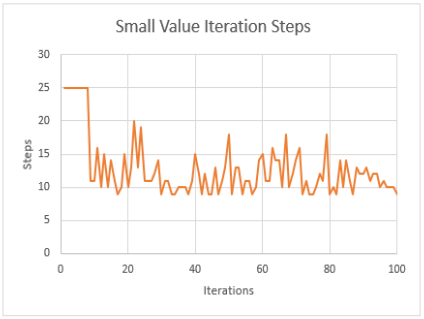


Figure 1.3

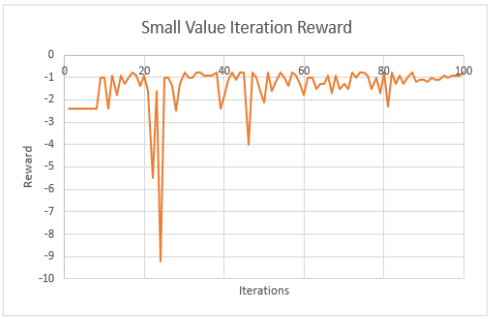


Figure 1.4

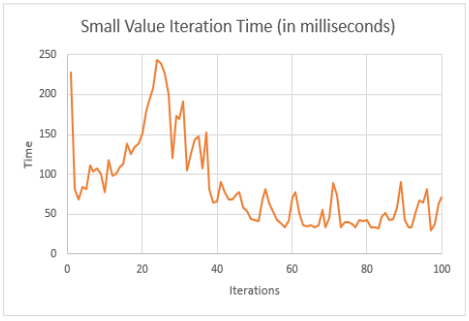


Figure 1.5

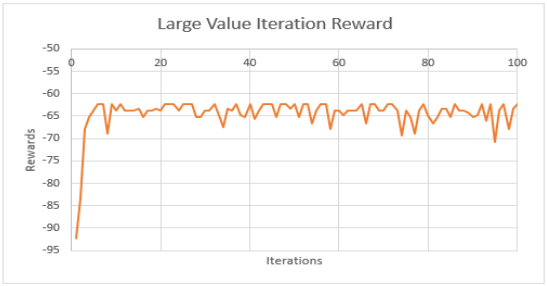


Figure 1.6

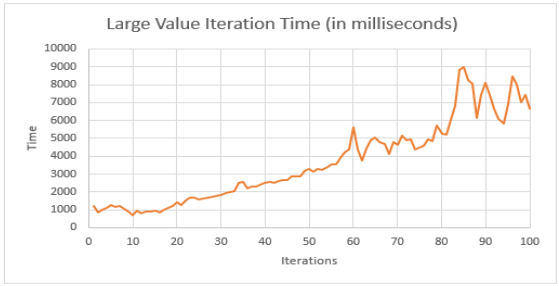


Figure 1.7

Value iteration, also called backward function, starts from the goal state and works back to its initial state and uses the Bellman's Equation to compute the optimal policy and value (maximize the expected utility). This algorithm requires domain knowledge. Since the back-propagation process of this algorithm does not end, an arbitrary end point is selected to stop the entire process when there is a good approximation. Maximum number of iterations and maximum delta threshold are used as stopping thresholds. For the small MDP, the average reward is -1.49, the average number of steps is 13, the minimum number of steps is 9, and average time in milliseconds is 107. For the large MDP, the average reward is -64.54, the average number of steps is 625, the minimum number of steps is 625, and average time in milliseconds is 3460. For both the small and the large MDP, I used 100 iterations and did not go beyond that because the algorithm converges within these iterations. It was interesting to notice that the number of steps increased and decreased as the number of iterations increased for the small MDP. However, for the large MDP, the number of steps remained constant at 625 as the number of iterations increased, which is also the total maximum steps within the grid. A possible explanation for this behavior is due to the nature of the value iteration algorithm and the way I have designed my complex grid, which allows the agent to quickly explores all the 625 steps which may also contain obstacles or walls. In essence, for the large MDP, convergence is achieved very quickly and does not require a lot of iterations and this phenomenon is noticeable in figure 1.6 where the agent starts with very low rewards but quickly converges right at the 1st iteration. The computational time for every iteration is calculated in milliseconds. For the small MDP the time increases and decreases between 20 and 40 iterations and but for the large MDP, the time increases linearly.

Policy Iteration

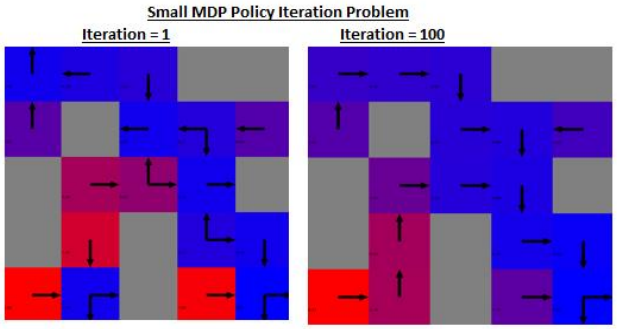


Figure 2.1

Figure 2.2

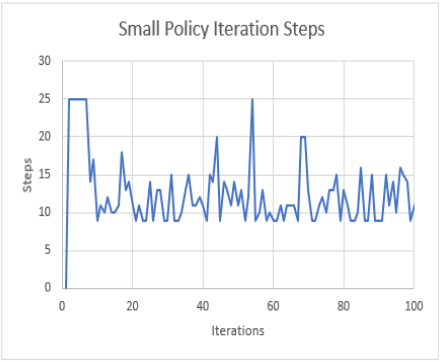


Figure 2.3

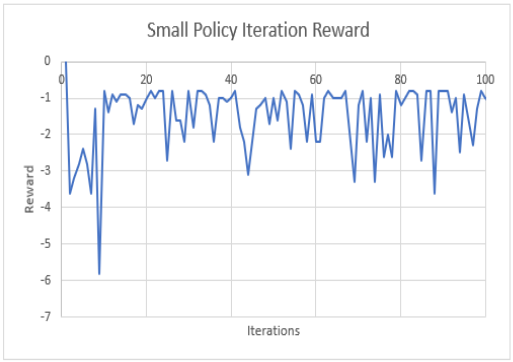


Figure 2.4

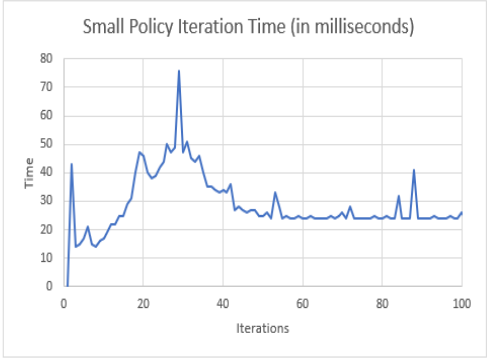


Figure 2.5

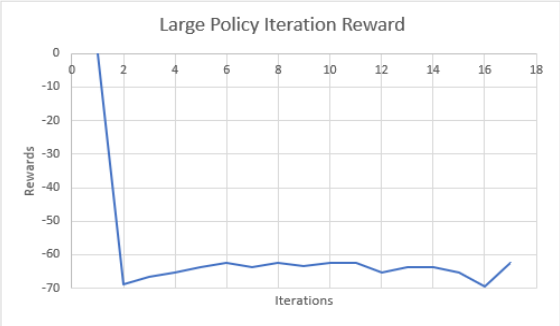


Figure 2.6

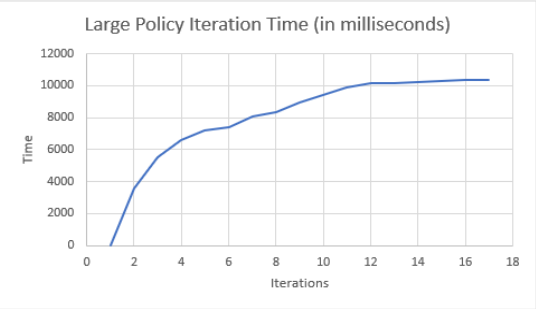


Figure 2.7

Policy iteration, like value iteration, finds the optimal utility using the Bellman's equation. This algorithm requires domain knowledge. However, this algorithm does not determine the values of all the states but instead it creates a random policy and computes the expected value at all the states under that policy. In this process, the algorithm iterates by modifying the policy to see whether the value at a state increases and if there is no improvement, the algorithm keeps spitting out the same values at the best possible policy that it has found. Although policy iteration is more complicated than value iteration, it is computationally very cheap. As seen in figure 2.7, the large MDP converges very quickly in terms of time at the 10th iteration. For the small MDP, the average reward is -1.52, the average number of steps is 12, the minimum number of steps is 9, and average time in milliseconds is 29. For the large MDP, the average reward is -64.42, the average number of steps is 625, the minimum number of steps is 625, and average time in milliseconds is 8529. I used 100 iterations for the small MDP because of the increased fluctuations in the plots. But for the large MDP I had to set the number of iterations to 16 because I had initially set it to100 and the algorithm started yielding the same results at the16th policy repeatedly and I had to intervene and hit the stop button. Just like in the pervious section, the number of steps for the large MDP remained constant at 625 indicating that the agent had to explore all the 625 grids in every iteration. It is interesting to notice the increase in complexity in terms of the number of directions that the agent took in both the small and the large MDPs as displayed in figures 2.1 and 2.2. The agent has done a very good job in avoiding areas with high negative rewards or bumping in to walls and has made an efficient use of its time in reaching its destination quickly.

Q – Learning

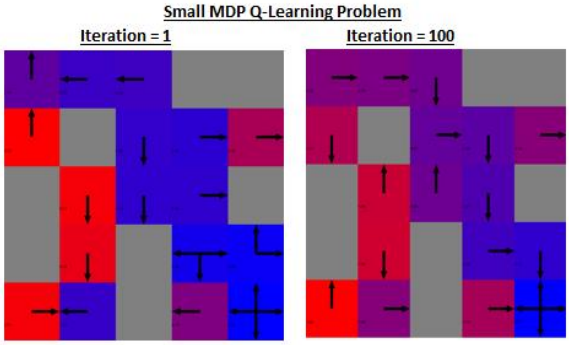


Figure 3.1

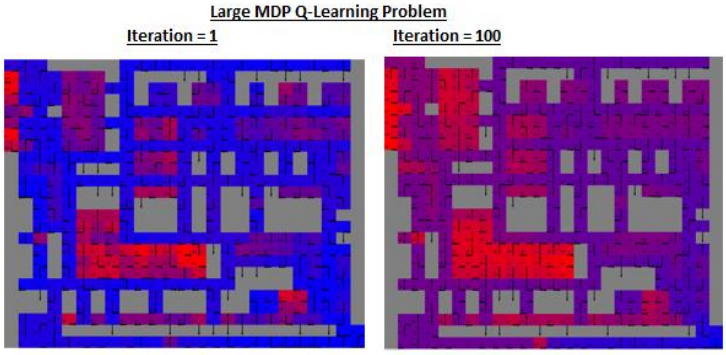


Figure 3.2

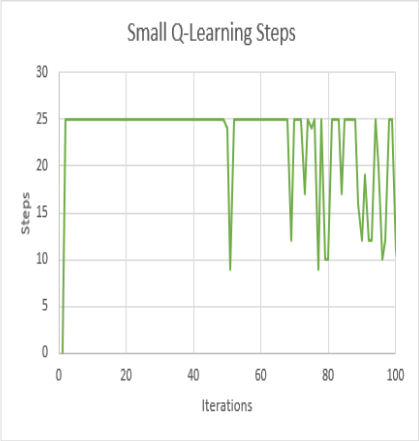


Figure 3.3

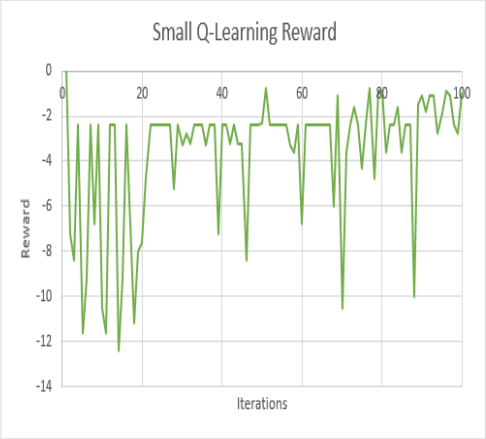


Figure 3.4

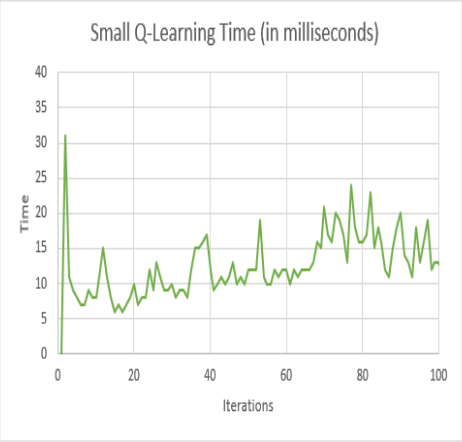


Figure 3.5

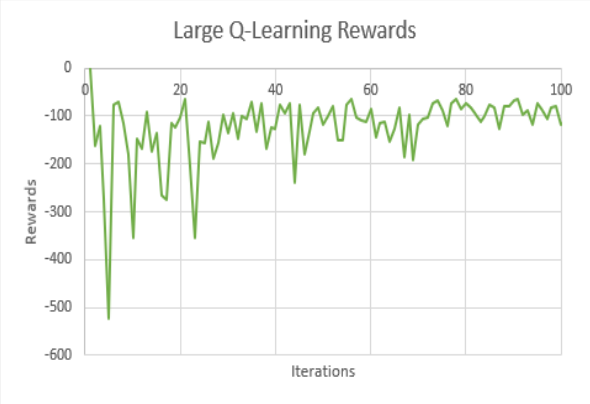


Figure 3.6

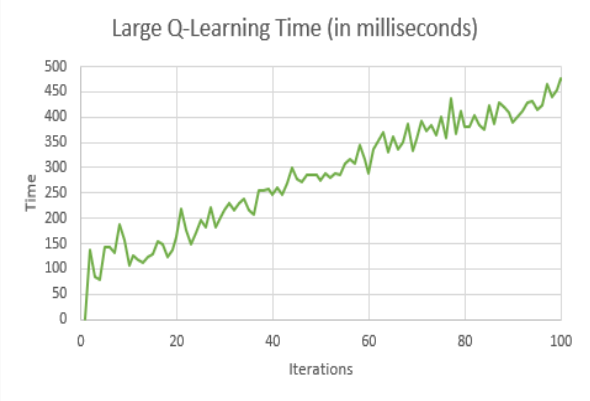
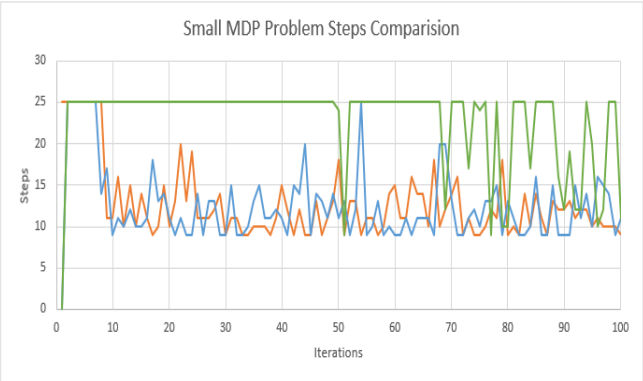


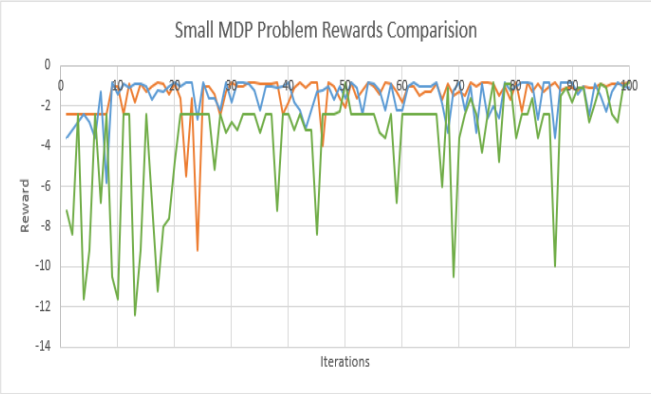
Figure 3.7

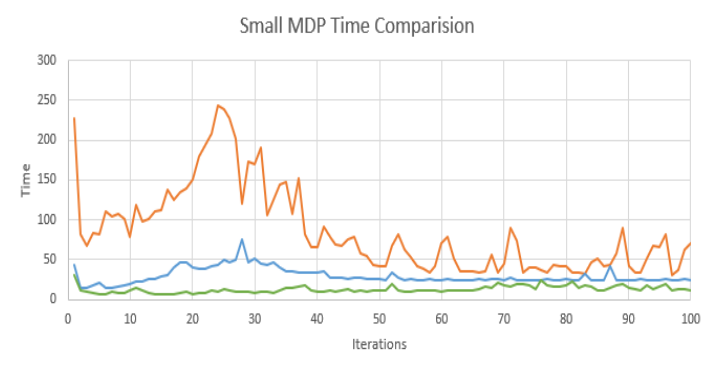
Unlike both the value and policy iteration, the Q-learning algorithm (model-free) does not require domain knowledge to calculate transition function. The model assigns all states a Q value then the agent revisits each state to re-evaluate or reassign new Q values based on delayed or immediate rewards. This process is very similar to the concept of exploration versus exploitation from randomized optimization because it balances between exploring delayed rewards versus executing immediate rewards. The Q-learning model's end goal is to learn or find a policy which will maximize the expected value of the total reward and which will also direct the agent in taking the appropriate actions under specific circumstances. For the small MDP, the average reward is -3.64, the average number of steps is 22, the minimum number of steps is 9, and average time in milliseconds is 12. For the large MDP, the average reward is -125.45, the average number of steps is 625, the minimum number of steps is 625, and average time in milliseconds is 287. I used 100 iterations for both the small and the large MDPs to be consistent during comparison and the models seems to converge within these iterations. Just like in the previous sections, for the large MDP, the number of steps for each iteration remained constant at 625 indicating that all the states are explored in every iteration. If you notice figure 3.3, up until 48th iteration, the step value remained constant at 25 and later fluctuated a lot. A possible explanation for this behavior is that the model is exploring the surface for delayed rewards

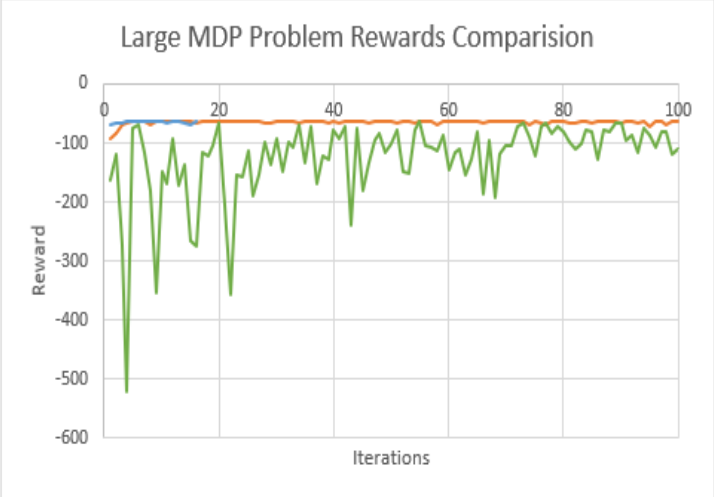
before exploitation. For both the small and the large MDPs, the computation time increased linearly as the number of iterations increased.

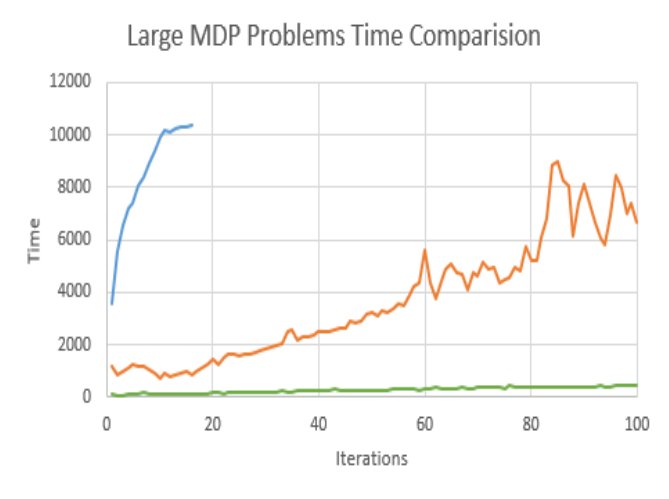
Conclusion and Model Comparison

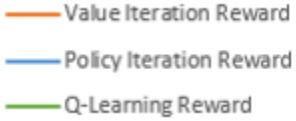




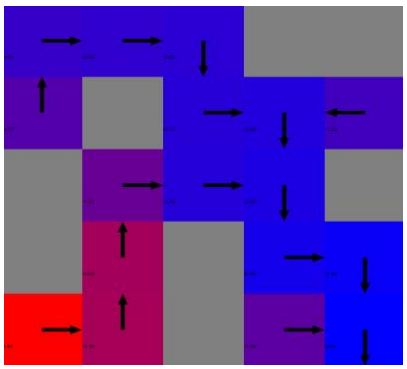




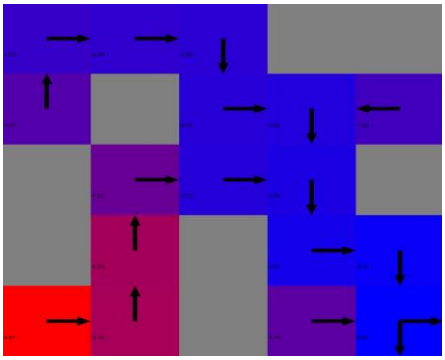




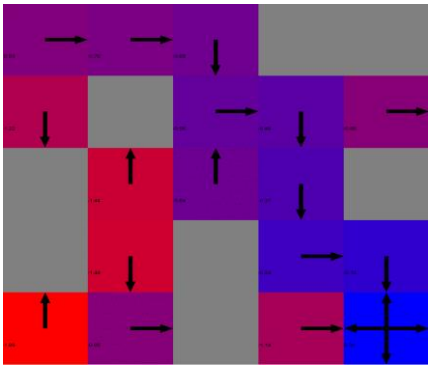
Small MDP: Iterations = 100



Value Iteration

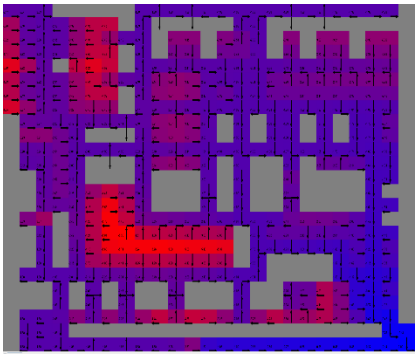


Policy Iteration

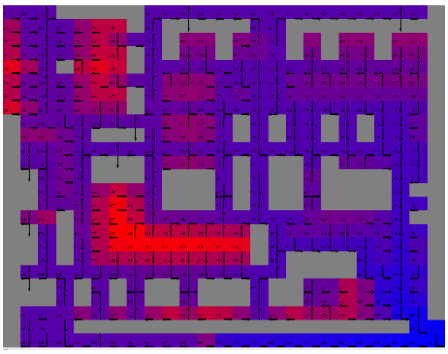


Q-Learning

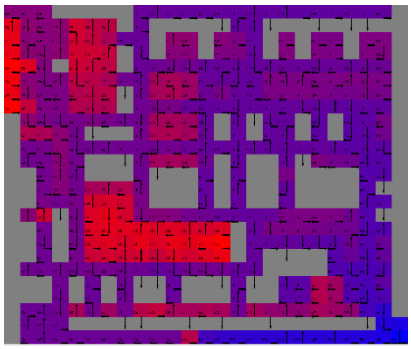
Large MDP: Iterations = 100



Value Iteration



Policy Iteration



Q-Learning

To summarize, two MDPs were analyzed in this report. The small MDP had a grid size of 5*5 and the large MDP had a grid size of 25*25. In both the MDPs, the agent's initial state was on the upper left corner of the grid and the end goal was on the lower right corner of the grid. Three reinforcement learning algorithms were applied on the two MDPs. For the small MDP, value iteration had the highest average reward of -1.498, Q-Learning had the highest average number of steps of 22, all the three reinforcement learning models had the equal minimum steps of 9, and Q-Learning had the lowest average time of 12 milliseconds. For the large MDP, policy iteration had the highest average reward of -64.425, all the three reinforcement models had the same number of average steps and minimum steps of 625, and Q-learning had the lowest time of 287 milliseconds. For the small MDP, 100 iterations were used for the three models. For the large MDP, value iteration and Q-learning required 100 iterations, but policy iterations required only 16 iterations for convergence. Overall, based on these results it is difficult to pick a winning reinforcement model because each of the them performed well in different categories. Although, I would choose to go with Q-learning model because of its incredibly fast computation time and its ability to not prematurely pick an optimal value since it uses exploration and exploitation to choose its optimal value.

Sources

1. https://medium.com/coinmonks/implement-reinforcement-learning-using-markov-decision-process-tutorial-272012fdae51
2. Class Udacity Machine Learning Lectures
3. http://burlap.cs.brown.edu/
4. https://www.eclipse.org/
5. Coding resource: https://github.com/svpino/cs7641-assignment4
6. Paper on Q-learning: http://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf
7. Used MS Excel for the graphs