CS 4641 – Machine Learning
Assignment 4: Markov Decision Processes and Reinforcement Learning

Abstract

This paper explores the Markov Decision Processes (MDP) and explains how different solution techniques and algorithms result in different policies to reach to an end goal. In order to explore these topics, two main examples will be used throughout the paper: a simple example to explain different algorithms and a more complicated one in order to compare and contrast these algorithms. First of all, the two well-known policy making algorithms will be explored (Value Iteration and Policy Iteration) to show how policies for MDP's can be created in different ways. Then, a reinforcement learning algorithm (Q-Learning) will be explored in the context of those two examples.

Background

An MDP is a way of modeling a world or a problem in a sense that it can be solved through an algorithmic way to produce a policy. The resulted policy would describe the agent's (the target decision maker of the problem/world, could be a person, a robot or software) decisions in order to reach a target goal in the most efficient way. A very important aspect of an MDP is the fact that it would only care for the current state that the target agent is in and it make its policy to reach for the end goal by only considering the current state and the future steps without looking at the past.

In order to exemplify the entire process of MDP's and Reinforcement Learning, the famous 'gridworld' problem will be used throughout this paper. A gridworld problem is a 2D mapping of a grid world that runs based on certain rules. Each block (square in grid) is called a state and each state would be represented with the corresponding coordinates of the grid block in the gridworld. The rules of the world explain the possible movements of the agent and a reward function describes how each step the agent takes benefit or harm the agent itself. The reward function that will be used in the examples is to receive 100 points to reach an end goal and -1 points for every step taken that results in a non-goal state. Occasionally some examples will have a penalty for trying to move into walls but this will be explained when the particular example is subject to analysis.

MDP's can be used to simulate any world problems so an optimal solution to that problem can be produced. Some examples would be modeling the environment of a car, the road, other cars, traffic lights, signs and each action the car takes can be transformed into a state. This modeling would allow an artificial intelligence algorithm to be developed for a self-driving car. Another example would be modeling an entire soccer game and all the players in it. The ball itself would be the main agent and each position change for the ball would mean a change in the current state. In a stochastic world (where each action has a small chance to produce an unexpected state), the optimum game play could have been probabilistically calculated using the MDP. The goal would be the ending point

with maximum reward and going out and opponent players would be negative rewards or penalties. *Figure 1* down below shows the gridworld modeling of a soccer field with a 4-4-2 gameplay. The MDP in target is modeled using value iteration (will be talked about later), and the light colored areas are desired areas while darker areas are not desired areas. In the end, the model succeeds to capture the idea of getting close to the goal as a positive event and being away from the sides, the posts and the opponent players as negative events.

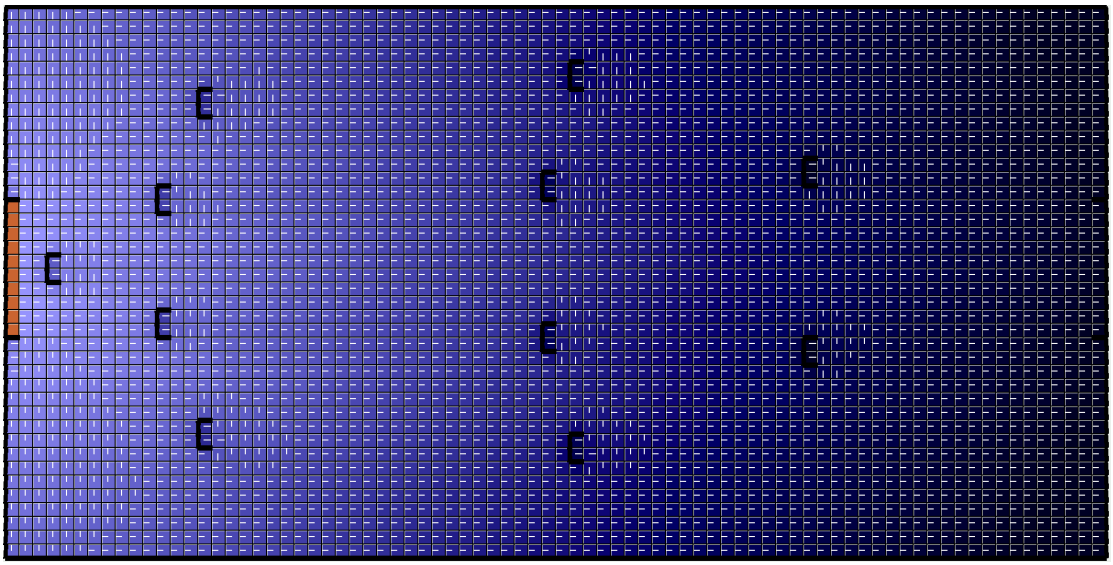

Figure 1: A gridworld example modeling a soccer field. Orange tiles model the goal and each rectangular object is an opponent player.

To perform the experiments on gridworld and obtain images, the two Java libraries (RL_sim and BURLAP) will be used. RL_sim is a library that allows the users to define their own mazes (gridworlds) using a GUI and offers control over the stochastic consequences of the actions. It also offers different algorithms to test these mazes and display them visually. BURLAP is very similar to RL_sim, however it allows users to specify a starting point as well, which will be important when the Q-Learning algorithm is analyzed.

Stochastic Worlds

The first MDP problem that will be discussed is a simple 5x5 gridworld example that aims to show how algorithms change policies when the how stochastic the world changes. To briefly explain what stochastic is, it's the random aspect of the world. If a world is described as stochastic in the case of MDP's, it means each action may produce an undesired action that was not planned. This randomness would be included in the rules of that world. For example, in a stochastic world, an action may produce the desired state 80% of the time, meaning that 20% of the time the next state would be an unexpected state. So if an agent tries to go north in that exact gridworld, it may end up in the north state with a probability of 0.80 and with a probability of 0.20 it may end up in any other adjacent state (block in the gridworld). These kinds of worlds are also described as non-deterministic worlds as well.

In real world, many unexpected events that might affect the consequences of an action can occur. In real life, a block might slip out of a robot's gripper, or a web page might fail to load, or a doorway might be blocked and all these are events that make the world non-deterministic.

Problem 1: Simple Gridworld

The simple gridworld defined for this problem is shown in figure 2. This particular gridworld has its ending goal at the orange square. Even though RL_sim doesn't allow users to define the starting point, the bottom-left corner will be accepted as the starting point. This gridworld is in a non-deterministic world where all the bottom corridor walls have a penalty of -100 meaning that if the agent tries to run into a wall it receives -100 points and the ending goal has a reward of 100. Each non-goal state has a reward of -1 to encourage the agent to reach to the goal state as fast as it can. Also, the rest of the walls do not have any penalty cost.

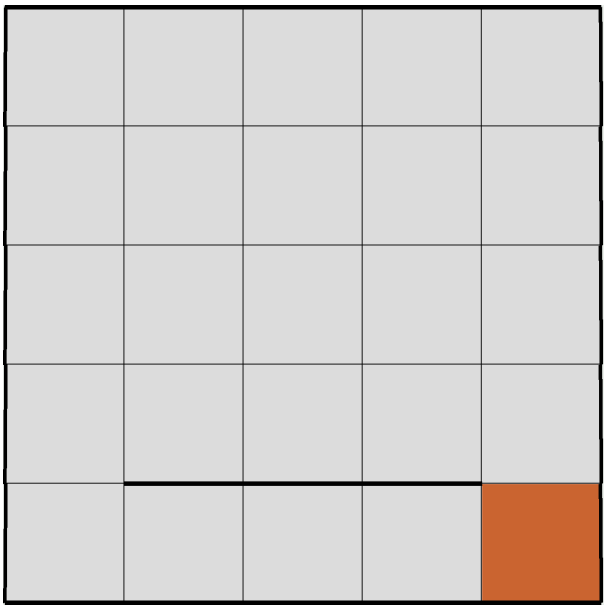

Figure 2: Simple gridworld for the first problem. The bottom corridor has a penalty worth as much as the end goal with a negative sign.

Value Iteration

Value iteration is the simplest way to find a policy for a non-deterministic MDP. It basically starts with using the immediate rewards for each state and decides which way leads to the best next state from a given state. In the end, it greedily searches all the states and choses the groups of lowest cost or highest reward states. Simply, it does this for all states and in the end the map tells which next state should be preferred given a state.

In this problem, the randomness of the world has been changed 3 times. The RL_sim library allows the user to tune the PJOG value which is the probability showing how much the action could result in an undesired and unexpected way. So in a world where PJOG = 0.2, the action would result in the desired way with a

probability of 0.80 (1 – 0.2). This randomness value is also described as the noise of the world. The point of this experiment was to determine when there's a penalty on the shorter path, if the agent will take the short path or the longer path depending on the discount factor.

In the figure 3 below, three different PJOG values (0, 0.3, 0.99) are used. The light colored tiles are the preferred states and the dark colored ones are the less preferred states. The right image has no randomness, therefore the agent would prefer to take the corridor even though it has penalty since in this world, it's know that each action will result in the desired state. The center image has the randomness value of 0.3 and it can be seen that the agent prefers to avoid the corridor and prefers the longer path. The image in the right shows almost true randomness of 0.99 and as it can be seen, any state is preferred over the corridor since almost any action in the corridor will result with a penalty value of -100.

Another important point is the number of iterations that each sub-problem takes to converge to the final result. The first image takes 9 iterations, the second image takes 48 iterations and the last image takes 186 iterations. This suggests that as the randomness or the noise of the world increases, value iteration needs more and more computations in order to determine the final policy.

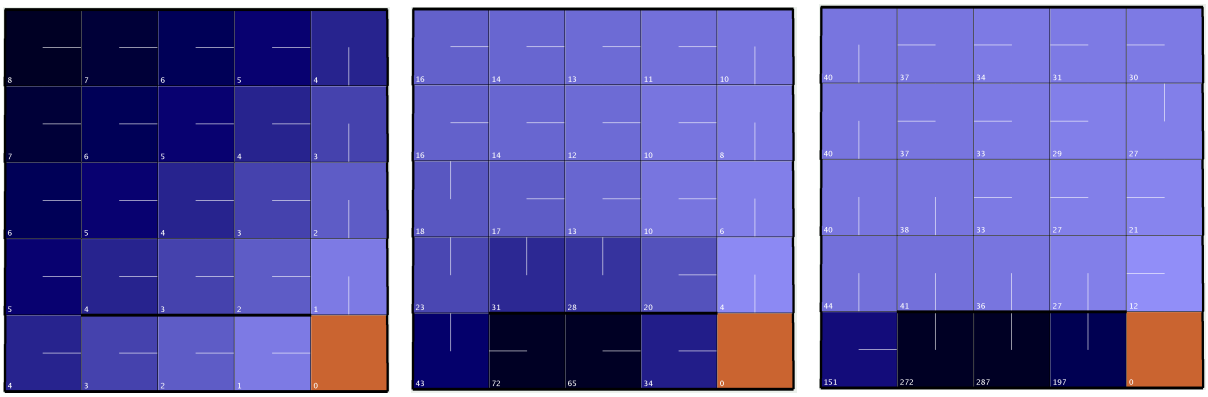

Figure 3: Value iteration for the simple gridworld. The PJOG value for each of the world is 0, 0.3 and 0.99 respectively.

Policy Iteration

Another policy-making algorithm is the policy iteration. What differs policy iteration from the value iteration is the way it looks the policies as a whole. While value iteration looks at each state and greedily choses the least cost one, policy iteration calculates policies this way and then compares the entire policies with each other to find the optimal policy.

Policy iteration algorithm was tested on two versions of the simple gridworld of problem 1. The first sub-problem had the PJOG value of 0.3 and the second had 0.99. Figure 4 shows the corresponding gridworlds with the optimal policies. The very first image looks very similar to the value iteration policy of the same gridworld with the same noise value and that is how it is supposed to be.

However, the policy iteration method took only 3 iterations to reach this result, proving that this algorithm is superior version of value iteration and this can be explained with its top-down approach compared to value iteration's bottom-up approach.

The second image on the right in figure 4 is different from the value iteration's policy. This image shows no reasonable policy since all the paths that go to the end point has a possibility to hit a wall because 0.99 noise means almost true randomness and in such a world, any policy would make any sense.

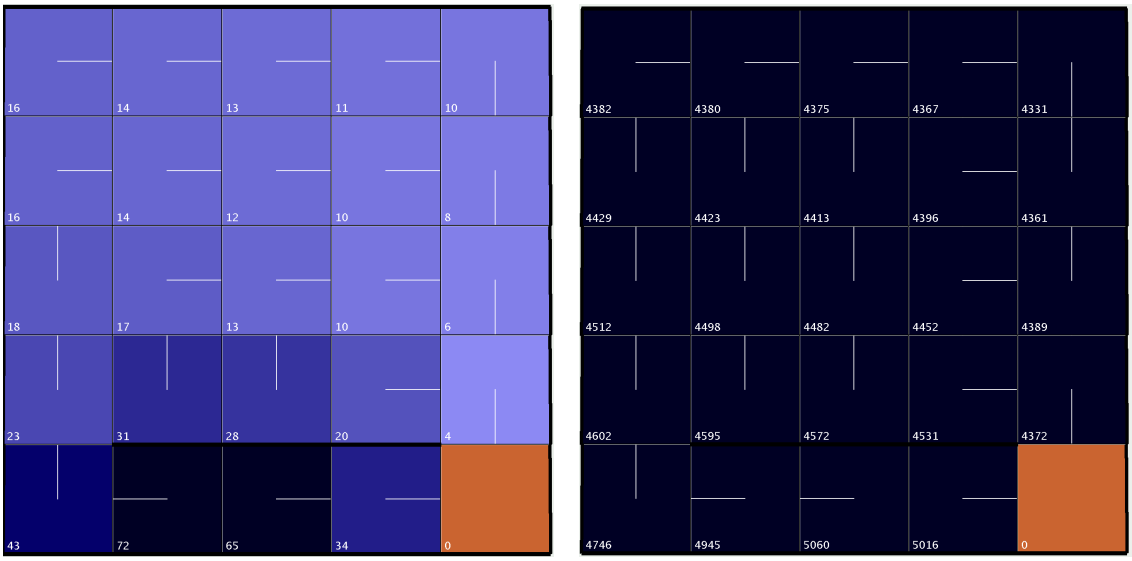

Figure 4: Policy iterations for the simple gridworld problem. The PJOG values are 0.3 and 0.99 respectively.

Q-Learning

Q-learning is reinforcement algorithm and out of all these 3 algorithms that this papers describes, it is the only real learning algorithm. What differs q-learning from the other policy-making algorithms is that it works with very limited world and domain knowledge. Until now, all the discussed algorithms had known everything about the worlds that they were running in. They knew where the walls were, where the end goal is, the rewards functions and the cost functions. However, q-learning only needs the available states and the performable actions in those states. It finds the rest of the knowledge by experimenting with different policies. In the end, it takes more time and iterations, but it manages to learn the world and model it using almost no information.

Figure 5 shows the gridworld before the q-learning algorithm starts and after 100 iterations have been made. The pink face represents the agent and the second image shows the policy it learnt after iterating and experimenting with the actions and rewards. As it can be seen, it prefers the long path to the short path with the penalties. It also takes more iterations compared to the first two algorithms since it took 100 iterations and it can still take more to come up with a better policy.

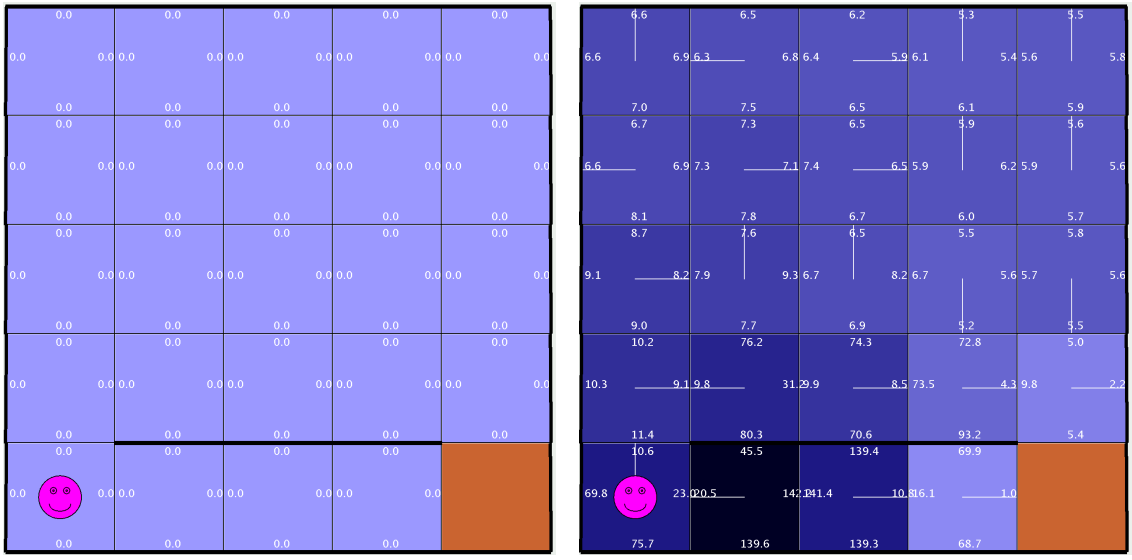

Figure 5: Q-Learning algorithm on the simple gridworld problem. The first image show the policy prior to the algorithm and the second image shows the policy after 100 iterations.

Problem 2: Complicated Gridworld

In order to understand the differences in algorithm better, a bigger and more complicated gridworld might be analyzed. Since all the subject algorithms are defined and explained, the main differences between the algorithms using a different example will be discussed in this section. The complicated gridworld example for this problem is created using the BURLAP library since it shows the exact number of iterations needs to find the optimal policy for any of the algorithms and the whole point of a complicated example is to compare and contrast how algorithms perform compared to each other and a simpler gridworld.

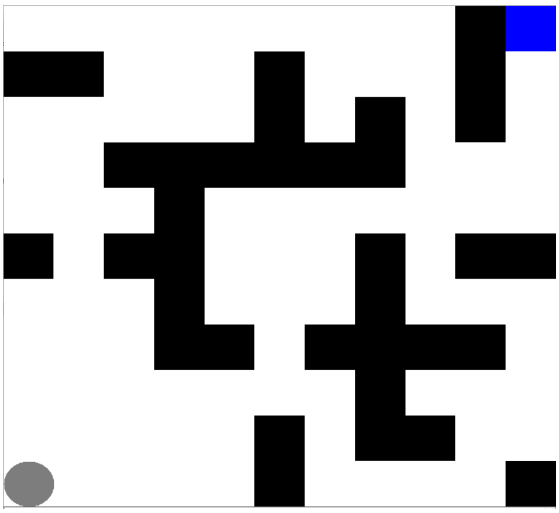

Figure 6: Complicated gridworld problem

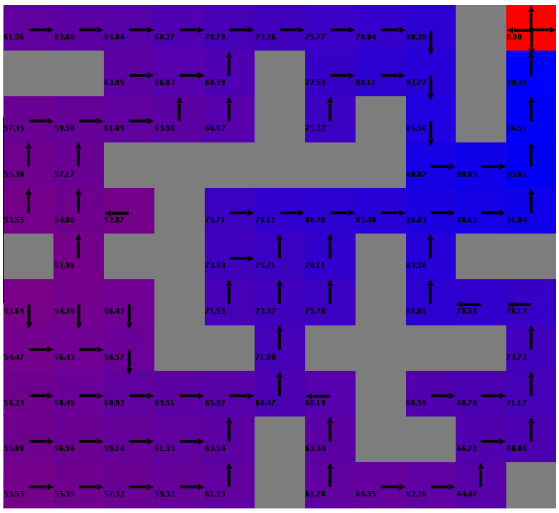

Figure 7: Value Iteration policy ma

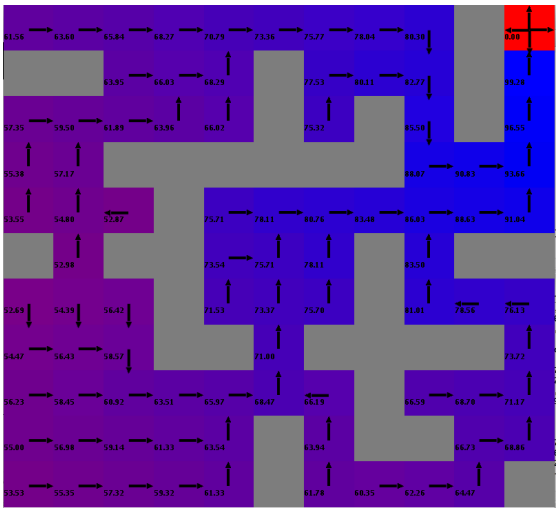

Figure 8: Policy Iteration policy map

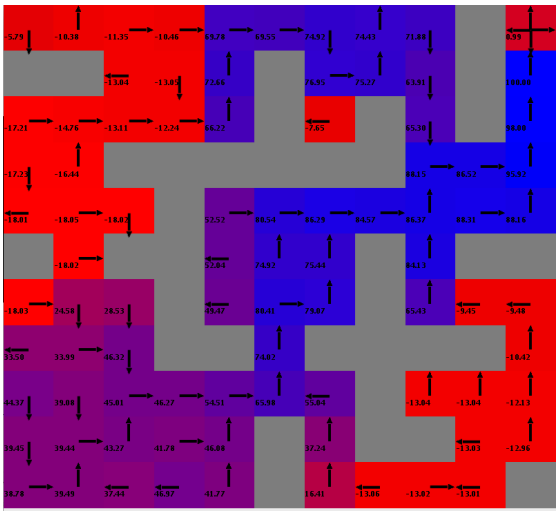

Figure 9: Q-Learning policy map

The figures 7, 8 and 9 given above show how different algorithms run on the gridworld problem given in figure 6. First of all, the policies that the value iteration and the policy iteration are almost the same due to the fact that they have similar working principles. On the other hand, Q-Learning comes up with a different policy compared to other algorithms. Also, red states can be seen on the policy map of Q-Learning suggesting that those tiles are really undesired. It comes to this conclusion because the algorithm visits those states without knowing they are the longer paths so it receives really low rewards when it reaches to the end goal using those paths.

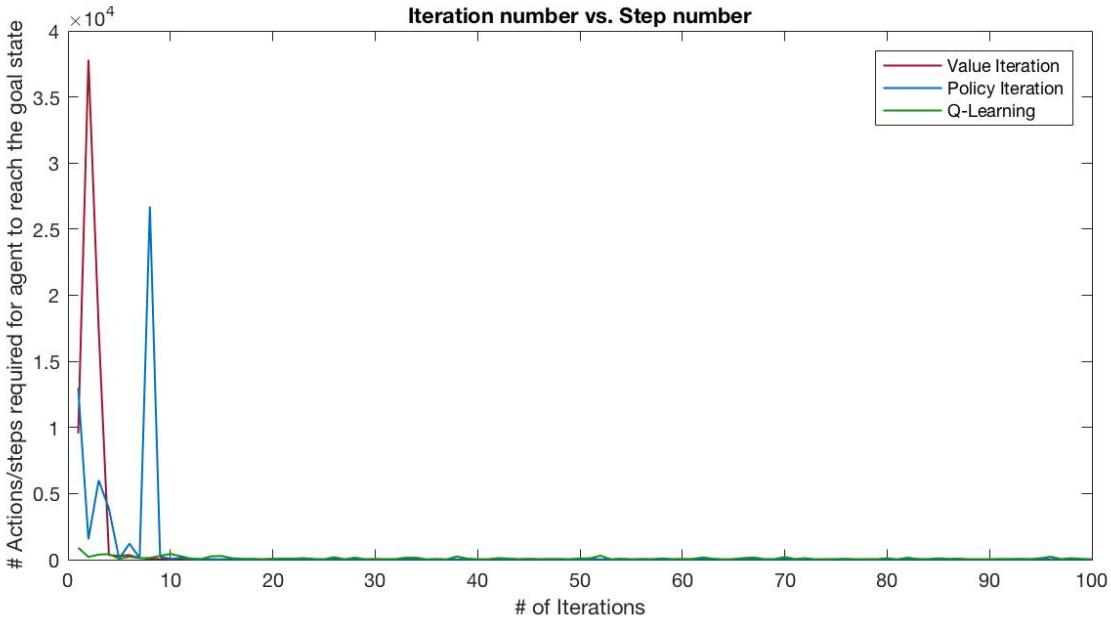

Figure 10: Number of iterations vs. min. number of steps needed by the agent to succeed for different algorithms shown on graph.

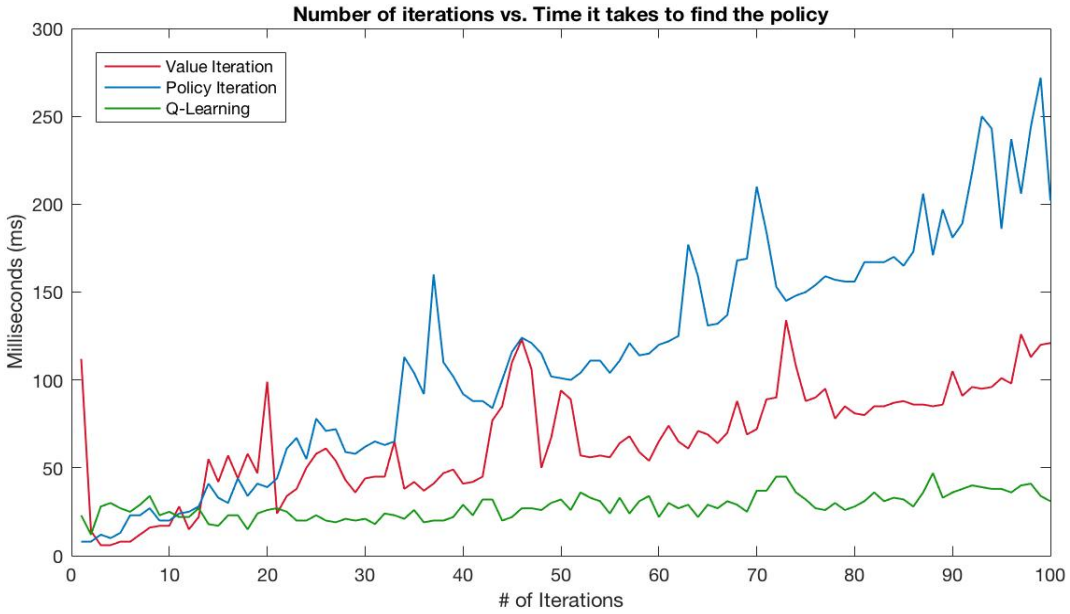

Figure 11: Number of iterations vs. the time it takes for each algorithm to find the optimal policy shown on graph.

Looking at figures 10 and 11, we see how each algorithm performs on the large (second) gridworld given in figure 6. Figure 10 shows the minimum number of steps of a given policy to reach to the end goal starting from the beginning from the starting point. The graph shows that value iteration and policy iteration starts with huge values since they need to search all the states first. However, even though it can't be seen in graph, they converge to around 20-30 range around 15-20th iterations. The green line showing the progress of Q-Learning seems lower at the beginning however, just the fact that it can be seen on the graph on such a high scale means that it fails to converge to a certain value. Looking at the real values, it still finds the minimum number of steps between 200 and 50 around 90th and 100th iterations. This means Q-Learning needs more and more iterations as the gridworlds get bigger since it learns the entire domain from scratch.

Figure 11 shows the time it takes for each algorithm to find the optimal policy for each number of iterations. With each coming iteration, algorithms add up on the previous iterations by calculating the Bayesian possibilities. As it can be seen, even though value and policy iterations find the optimal policy quicker than Q-Learning, it takes more and more time for them to compute the same policies since they add up to the previous ones. Q-Learning on the other hand, with each iteration, continues to learn more and more about the domain but this happens in linear time in iteration-wise since each iteration reveals a tiny bit of new information instead of searching the entire gridworld. In the end, Q-Learning gives up the domain knowledge that the other algorithms have to have a faster runtime even though it takes more iterations compared to the other algorithms to find the optimal policy.

Conclusion

Seeing that both value and policy iterations use noise calculations and the reward functions in order to find the optimal policies for the given gridworlds, it was expected of them to converge to optimal policies faster and in less iterations than Q-Learning. However, given the very minimal amount of domain knowledge, Q-Learning experiences the entire gridworld and discovers the reward function as well as the randomness of the world. For this reason, the fact that they have so much differences in their inputs, it doesn't make a clear sense to compare these algorithms one by one, however they are different approaches to solving the same problem that each of them have their own strengths.

On the other hand, it is clear that with a high number of states, there is an exponential increase in number of iterations required for Q Learning to converge on an optimal policy. If the domain knowledge is known in a situation, value and policy iteration methods should be preferred whereas in the case of an unknown world, Q-Learning stands superior to those algorithms in order to discover more about the given universe.