Andrey Lukin
CS 4641 Machine Learning
Project 4: Markov Decision Processes
Professor Charles Isbell

Project 4: Markov Decision Processes

For Project 4 of CS 4641, we will be discussing Markov Decision Processes (MDP for short). MDPs allow us to figure out what to do on every step, given a stochastic environment. A stochastic environment is one allows for probabilistic error. This mean that given an action we want an agent to take, there is a chance that the action will not be correctly fulfilled. Unlike examples in Artificial Intelligence in regards to search, there is a chance that things will no go as planned, and this is a lot more similar to how the real world operates.

Another important thing in MDPs is that it only cares about the state that the agent is currently in. This is a good way to model real world environments, because usually there isn't necessarily an overall view of the model to make general decisions about the problem. You won't always have something like the heuristic function in A* search to help you out to solve problems, and MDPs make it easier to solve these problems. Additionally, with MDPs we don't care about the past.

To push the machine to solve the problems using MDPs in the quickest way possible, we have the idea of rewards: once the agent reaches the goal state, they receive a high reward, and the longer they wait, the less reward they will receive. This pushes the machine to figure out what to do in the most efficient way possible.

For this paper, we will be using Grid World to analyze how Policy Iteration and Value Iteration preforms. Grid World is the most fundamental MDP problem: you have 2D world on a coordinate system where there is/are terminal states. Whenever the agent wants to perform an action, there is a 6.6666% chance the agent will accidently move to the left of the action, 6.6666% chance it will move to the right of its action, and 6.6666% it will move in the opposite direction. This problem is interesting because it's the most basic MDP problem. What that means is that any MDP problem can eventually boil down to grid world. Problems like self-driving cars or chess playing machines can be mapped out in this way, (with a considerable amount of generalization). For this paper, we will be looking at a small grid world problem, and also a very large one, and seeing how they differ.

The different ways we will be solving these problems using MDPs is by Value Iteration and Policy Iteration.

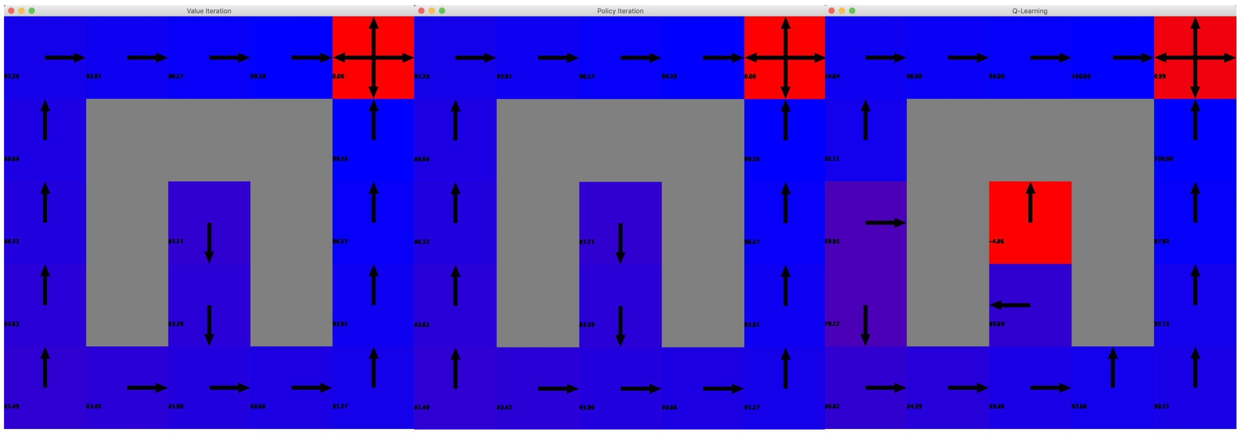
Value Iteration: In value iteration what we are aiming for is that given the current state and the states that the agent can move to from the current state, what would be the the best thing to do given the utility of the states that the agent can move to. This works with the neighbors to update the values.

Policy Iteration: Works very similarly to value iteration, but does not do values but rather updates the policy itself of where to go next. Works with neighbors.

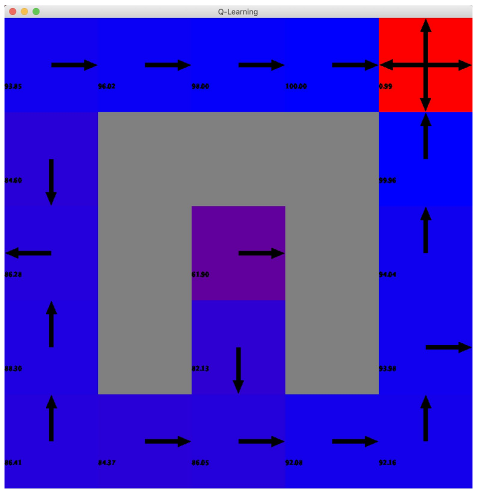
Q-Learning: No transition or reward models and doesn't know if the route is optimal. It needs to try out the different combination slowly. I used the Epsilon Greedy exploration strategy after comparing Epsilon Greedy, simply random exploration, and Boltzmann exploration, and Epsilon Greedy was easiest to work with.

Small Grid World

On the left, you see the current map that I created for testing MDPs in the small grid world. The gray circle demonstrates the start state, and the blue square is the goal state. The black squares are the obstacles that I programmed to stand in the way of the agent. I decided to do this formation for the map because it provides an interesting problem: the little alcove could attract the machine in the beginning, and could get it stuck. As more iterations go along though, the agent will understand that this alcove is useless, and thus will try not to go into it.

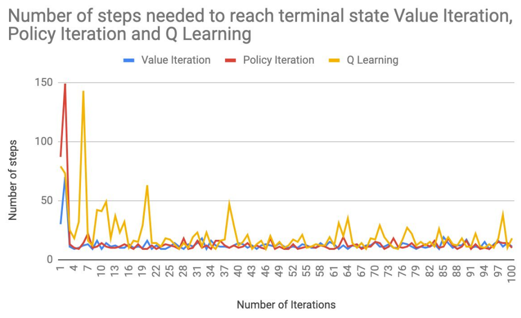


Here we have the policies per state after 100 iterations with value iteration, policy iteration, and Q-learning (left to right in that order). Interestingly enough, Q-learning is a little "confused", since in that indentation it continues going into the alcove even though there's no way for it to get out of it. The reason behind that is that Q-learning has no general view of the model, and must decide on the policy it wants to do only based on the current state.



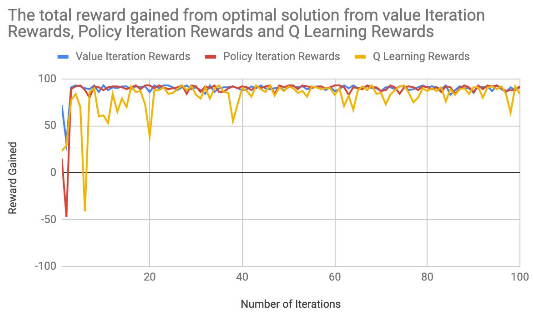
Even after 1000 iterations, Q-Learning still is not able to have clear way of getting out of the alcove. Here is the policy map that was generated by it. This makes sense. Q learning doesn't work on rewards and hence is the true realistic exploration algorithm.

On the other hand, value and policy iterations were able to correctly do all the correct policies on the map. They point to get out of the alcove as soon as possible, and go directly to the goal state.

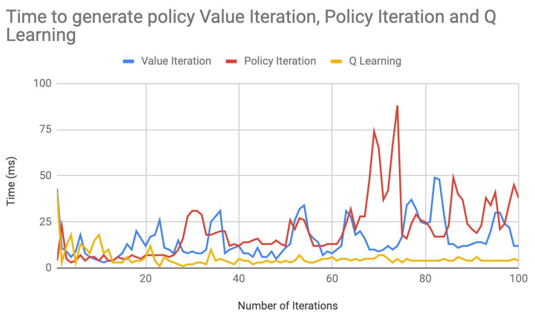


Now, how do these three compare with objective metrics?

Here we see that policy and value iteration outperform (on average) q-learning in the number of steps need to reach the terminal state after x number of iterations. That makes sense, since Q-learning has less information than value and policy iterations. Both value and policy iteration converge at about 4 iterations. This is not surprising, because the map is small. Q-learning relies more on randomness so it takes longer to converge.



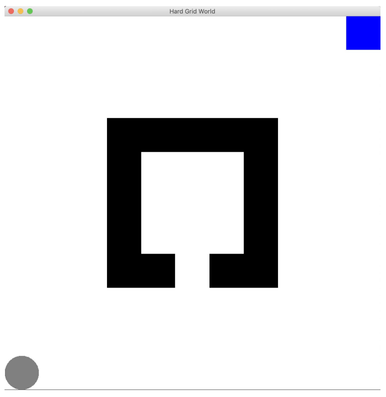
It's not surprising to see then that the same can be seen with rewards that the different algorithms achieve. All algorithms are aiming for the max score, but q learning doesn't improve to maximize the score. This is why this graphs looks like the inverse of the number of steps one.

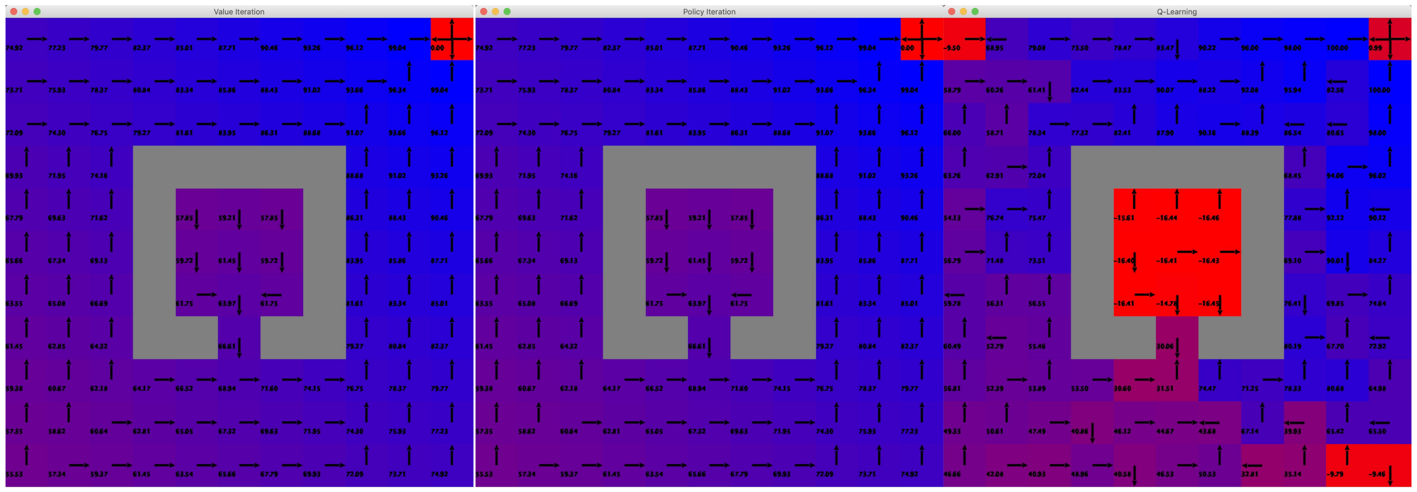


Because this is a small map, policy and value iteration preform about the same. This makes sense for smaller maps (Kaelbling "Enhancement to Value Iteration and Policy Iteration" 1996).

Large Grid World

For the second problem, we will be making the problem a little more difficult, and making the map larger. We will still have the alcove to try and purposefully catch the agent and complicate things. This is an interesting problem in general because the real world is not made for computer agents to work in, so a lot of things might not be created for the efficient use by agents. Again, the gray circle is the starting point, blue square is where we want to end up, and black squares are walls.

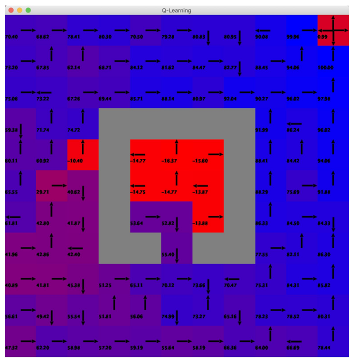


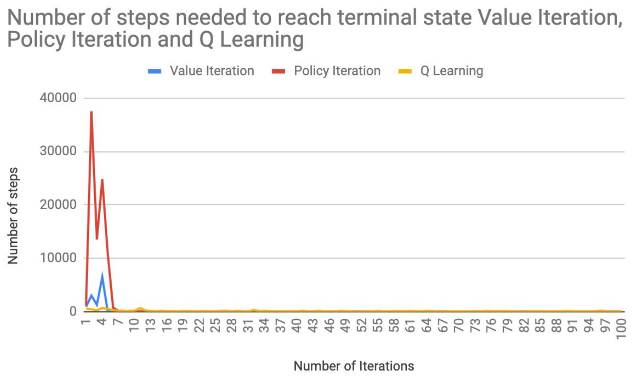


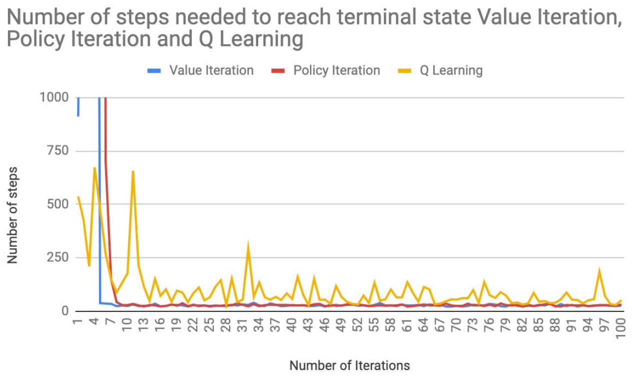
Here we see we are a similar situation as last time. Value and Policy Iterations are able to correctly get out of the alcove whereas the q-learning one doesn’t do a good job of that. I decided to run q-learning for 1000 iterations again just like last time, and here is what I got.

It’s a bit better, but because q-learning is based primarily on randomness, it makes sense that it takes longer for q-learning to converge to the optimal solution

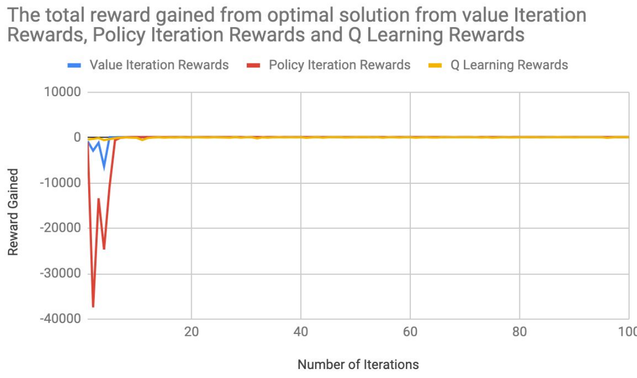
Now, let’s compare other metrics.

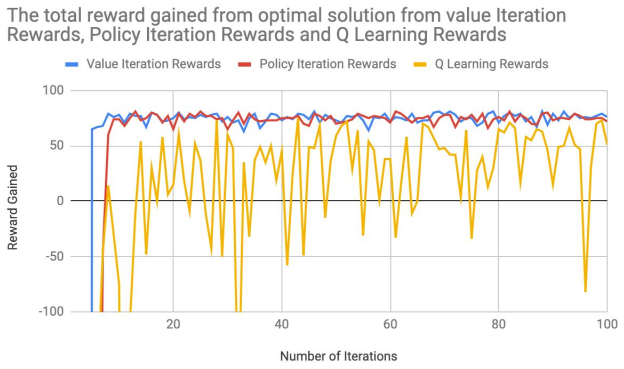


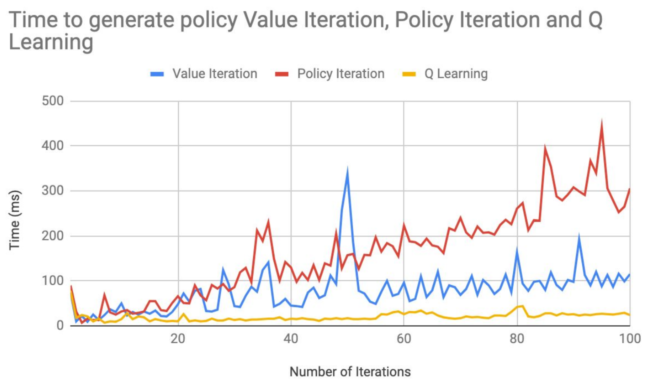




Here we have the graph to demonstrate the number of iterations vs the number of steps it takes to find an optimal solution. These two are the same graphs, but the left one is magnified to look at convergence. Here we see that Value iteration converges at about 5 iterations, while the Policy Iteration converges at 8. They do converge to the same answer, but surprisingly value iteration does it in fewer iterations. This is unusual since value iteration hypothetically requires more computation (Kaelbling "Enhancement to Value Iteration and Policy Iteration" 1996). Since Q learning is based a lot more on randomness, it doesn’t converge as easily. Again, the total reward seem like the same graph reversed.







Since the map grew to be a lot bigger now though, policy iteration takes longer per cycle to do. This makes a lot of sense based on comparative research between value and policy iterations (Kaelbling "Enhancement to Value Iteration and Policy Iteration" 1996).

Conclusion

After comparing and contrasting Value Iteration, Policy Iteration, and Q-learning, we have been able to understand the general purpose of the different algorithms, and when we would want to use one vs the other. In general, it seems that even though Value iteration and Policy iteration performed better than Q-learning, it makes sense why this was the case if you think about how much more information is available to those two algorithms vs Q-learning. Hence, Q-learning would be best for maps that do not have the amount of information that we got here, like the rewards, etc.

For this project, I used the code of juanjose49 (https://github.com/juanjose49/omscs-cs7641-machine-learning-assignment-4). I modified some of the code to test some of the algorithms in a more in-depth manner.

Bibliography

Kaelbling, Leslie Pack. “Enhancement to Value Iteration and Policy Iteration.” Enhancement to Value Iteration and Policy Iteration, 1 May 1996, www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node21.html.