

Charlie McKnight  
CS4641  
Charles Isbell  
04/23/2017

# Reinforcement Learning

## Introduction

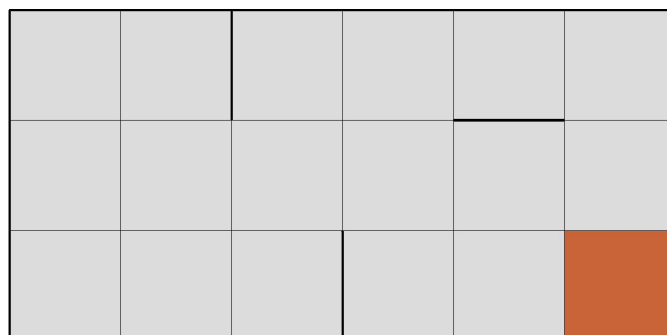
This assignment deals with Markov Decision Processes and Reinforcement Learning Algorithms. The three algorithms which I will be experimenting with are Value Iteration, Policy Iteration and Q-Learning (with an epsilon-greedy approach to exploration / exploitation). To Simulate the following algorithms I used the “Reinforcement Learning - Simulator” developed as a learning tool at Carnegie Mellon (<https://www.cs.cmu.edu/~awm/rlsim/>). The problems are set up as a sort of grid world maze problem. Every transition has a cost of 1 associated with it, while bouncing into a wall has a cost of 50. The results of trying to go from one state to another are non-deterministic with the probability of going to an unchosen state being specified in each experiment (PJOG).

## Problem Declarations

I chose to do a sort of maze problem because I believe that maze problems demonstrate many of the advantages of Reinforcement Learning. With a standard maze problem before any learning has occurred each state seems to be just as good as every other state (ignoring the final states of course) it takes time for the data from the goal states to propagate outward before a path appears. Now if we were simply solving a maze, depth first search would be a much faster alternative (running in linear time). However, in our mazes we take into account the fact that there are non-deterministic actions as well as the fact that running into a wall incurs a penalty. This makes the maze problem well suited to RL.

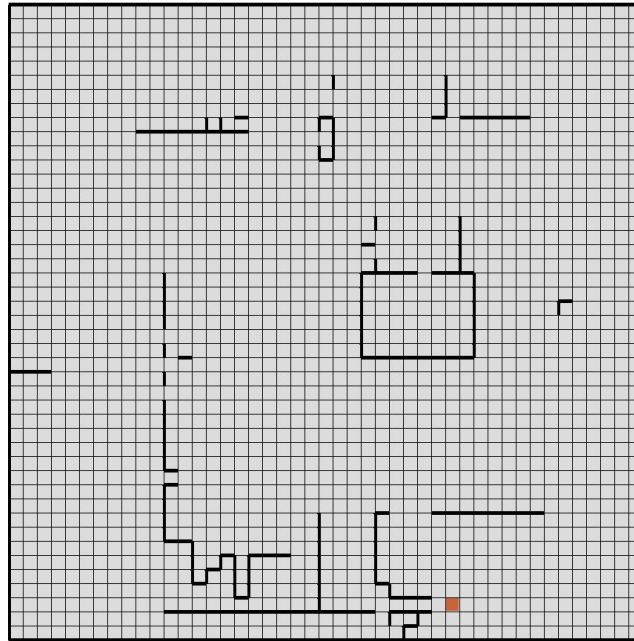
## Easy Problem

Below is the first map that I chose. As one can see it is a very small map measuring 3X6, having 18 possible states. The final state is marked in orange. Even though MDP's do not rely on previous states and only on the current state, we do have a starting state. The starting state is in the bottom left. This will come into play for Q-Learning as the location we start learning from. With Policy Iteration and Value Iteration the starting location largely irrelevant.



## Hard Problem

For the hard problem I chose a similar map, however one that had many more states. This map is 45X45 having 2025 possible states. This map is also more interesting because the larger size allows for more interesting features. The most interesting of which is the wall that separates (0,0) from the goal with many holes in it. Whether or not the algorithms decide to go through the hole in the wall and risk a penalty or go around will be interesting to see. The rooms long corridor and the bottom middle, will also be interesting because it is a very direct path to the goal but also dangerous because of all the potential penalties.



## Comparison

These two problems end up yielding interesting results when compared to each other because of the differences in number of potential states. Many comparisons in algorithms can only be seen when looking at problems that are different orders of magnitude.

## Value Iteration

The idea behind value iteration is that each state has a utility based on its reward and potential rewards or deficits from future states. If the true utilities were known then one could simply take an action from the current state that will optimize our utility, thus maximizing rewards. Value iteration attempts to find the true utility of a state by assigning arbitrary utilities to states and iteratively updating until convergence (within a threshold). The updated utility is calculated as follows.

$$U_{t+1}(s) = R(s) + \gamma \max_a (\sum T(s, a, s') U_t(s'))$$

That is we use Bellman's equation to calculate the new utility of our state based on the utilities that our actions could take us to. We do this for every state until the utilities converge (within a threshold).

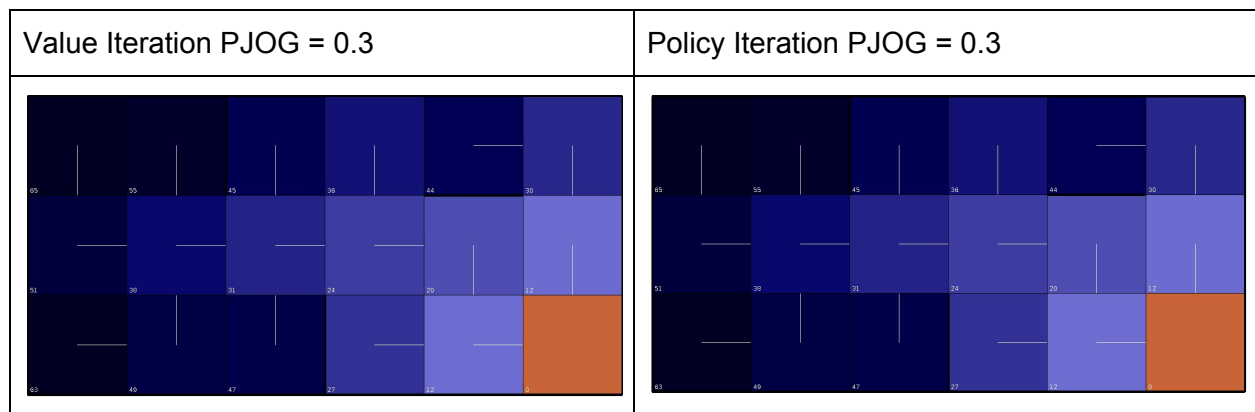
## Policy Iteration

Policy Iteration is meant to be a kind of simplification of value iteration. Whereas with value iteration you were trying to find the utility of each state. Policy iteration says, "I don't care about the utility of a state, all I care about is finding what action I should take in each state". Thus policy iteration usually converges in fewer iterations.

## Comparison Of Value Iteration and Policy Iteration

### Easy Problem

As we can see both Value Iteration and Policy Iteration converge to the same answer. Which in this case ends up being get to the center to avoid walls and go to the goal location.



In each of these cases however the number of iterations and the time it took varied immensely between algorithms. As was predicted policy iteration took many fewer iterations to solving the problem than value iteration. I attribute this to the fact that policy iteration takes a more direct route to what we really want to know which is the policy. In many ways the policy derived from value iteration can be seen as a byproduct of the calculation.

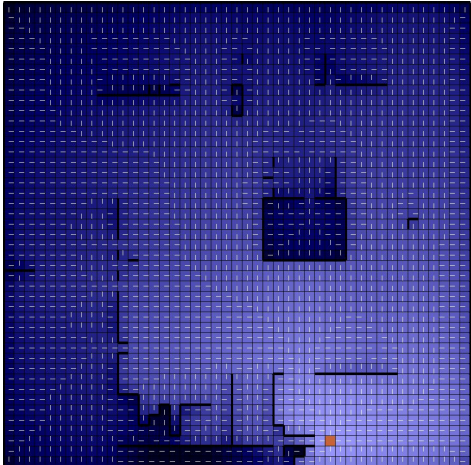
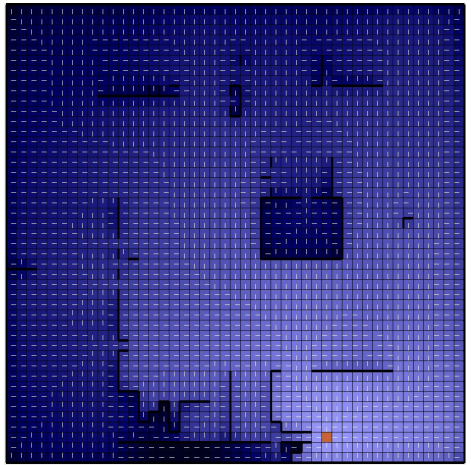
However, when it comes to time, it becomes clear that the benefits that policy iteration had in terms of number of iterations does not hold true when it come to clock time. Both algorithms take similar amounts of time to arrive at the same policy. This holds true regardless of how deterministic the actions are. One side note that I found interesting was that as the actions became less deterministic, the number of iterations required to converge also shrunk for policy iteration. The reverse was true for value iteration. I believe this is true for value iteration because as the determinism goes down each state's utility is more determined by the utility of

the states around it as opposed to being almost solely reliant on one state's utility. This makes convergence take longer.

Algorithm	PJOG	Precision	Iterations	Time (ms)
Value Iteration	0.4	0.001	68	11
Value Iteration	0.3	0.001	44	3
Value Iteration	0.2	0.001	31	4
Value Iteration	0.1	0.001	21	2
Value Iteration	0	0.001	8	1
Policy Iteration	0.4	0.001	3	10
Policy Iteration	0.3	0.001	4	5
Policy Iteration	0.2	0.001	4	4
Policy Iteration	0.1	0.001	4	7
Policy Iteration	0	0.001	7	3

## Hard Problem

For the harder of the two problems, as expected, both converged to the same policy. Albeit with different utilities for the individual states. However, since all we really care about is the policy that doesn't matter.

Value Iteration PJOG = 0.3	Policy Iteration PJOG = 0.3
	

In terms of the number of iterations over this larger map with more states we see similar trends. The first being that Policy iteration takes way fewer iteration that value iteration. However, the time takes way longer. This all goes back to the complexity of each iteration. It

can be shown that each iteration of value iteration can run in  $O(|A||S|^2)$ . Each iteration of policy iteration can run in  $O(|A||S|^2 + |S|^3)$ . In our problem the number of actions is four, which means that the  $|S|^3$  is the dominating part. This holds especially true in the larger of the two maps. This ends up causing policy iteration to take much longer in our map problems especially as the map size increases. However, in a problem where the number of actions approaches the number of states, I speculate we would see increased performance of Policy Iteration as opposed to Value Iteration.

Algorithm	PJOG	Precision	Iterations	Time (ms)
Value Iteration	0.4	0.001	260	5485
Value Iteration	0.3	0.001	181	3848
Value Iteration	0.2	0.001	136	2833
Value Iteration	0.1	0.001	106	2312
Value Iteration	0	0.001	74	1497
Policy Iteration	0.4	0.001	13	9924
Policy Iteration	0.3	0.001	15	9334
Policy Iteration	0.2	0.001	17	11454
Policy Iteration	0.1	0.001	49	10039
Policy Iteration	0	0.001	66	10327

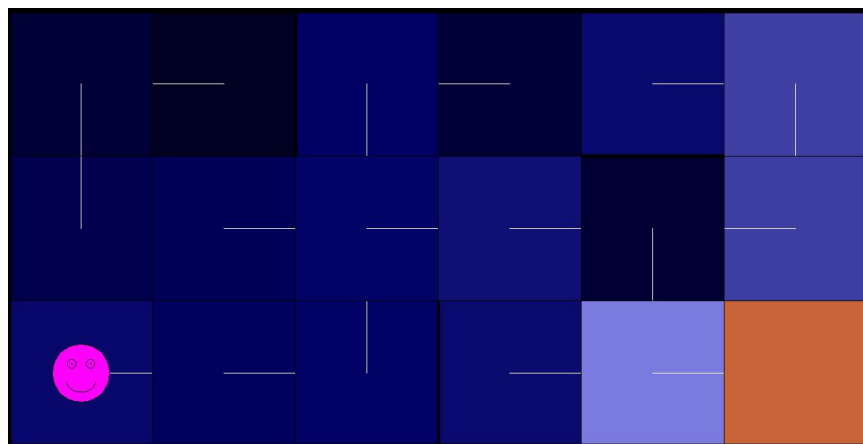
## Q-Learning

Q-Learning sets out to solve the same problem as Policy Iteration and Value Iteration but under different circumstances. In many ways Q-Learning can almost be seen as trying to solve the problem blind. Whereas in the previous two you knew all potential states, all rewards at those states, and the actions that took you from one state to another. With Q-Learning you don't have to know all of those things. You can start with just an initial state as well as transitions that can be made from current states. This is unideal, but it is a reality in many RL problems that you will not have all the information that you want.

The main decision that must be made with Q-Learning is how to choose your actions. As is common in ML it all comes down to exploration vs exploitation. If you always choose the action that seems best at the moment (greedy approach) then you will get stuck in what could be a local min/max. However, if you proceed randomly every time (random-approach) then you are in essence not using what you have learned so what is the point in learning at all. A common approach called epsilon-greedy is to use what you think is ideal a certain percentage of the time and proceeding randomly the rest of the time. For the results in this paper I will be focusing on epsilon greedy, with the other two acting as comparisons. When unspecified, I will be using a learning rate of 0.7.

Epsilon Greedy $\epsilon=0.1$	Greedy	Random
Episodes = 699000 T = ~10s	Episodes = 699000 T = ~6s	Episodes = 699000 T = ~60s

However, one benefit to Q-Learning is that it doesn't take long to arrive at a suboptimal solution. After only 20 episodes, Q-Learning( $\epsilon = 0.1$ ) is able to find a fairly good path, with none of the information that PI and VI had.

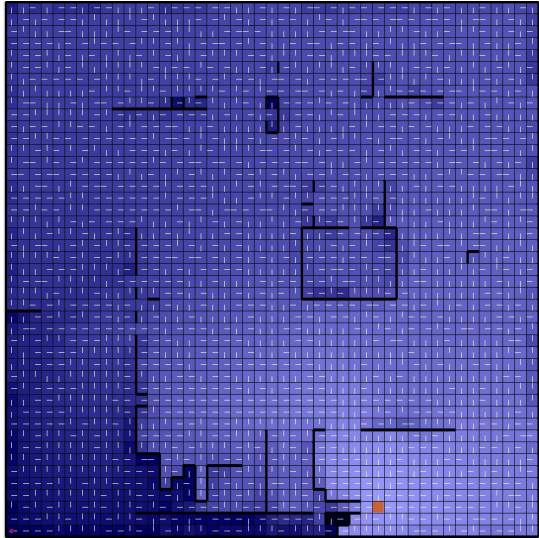
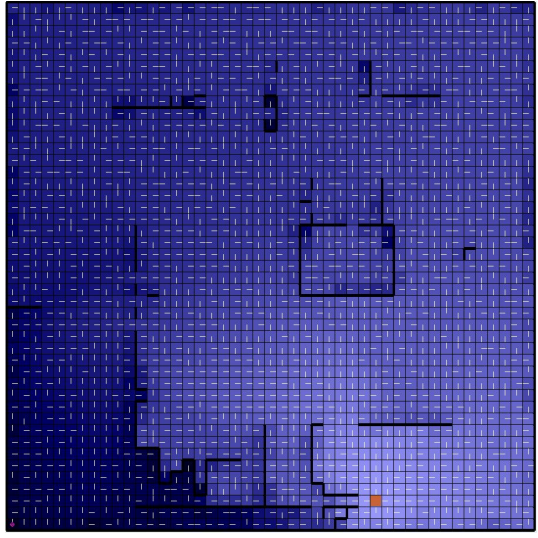


## Hard Problem

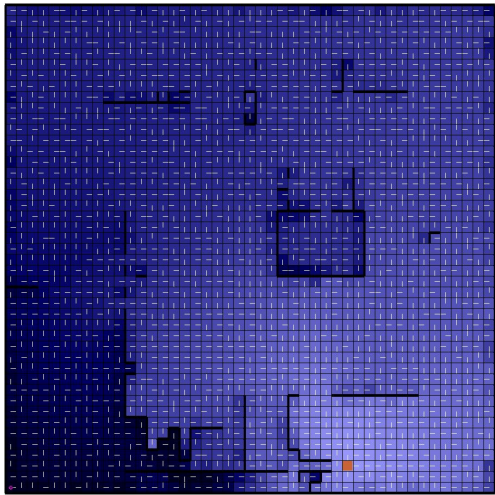
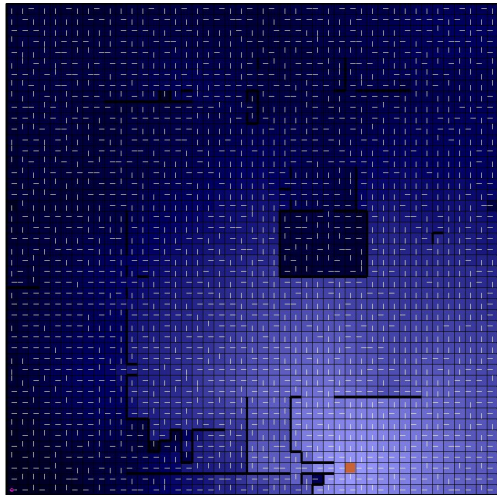
The hard problem, with its more interesting map, yielded more interesting results than the small map. For the hard problem instead of comparing number of iterations and episodes (Which is like comparing apples to oranges) I will be focusing on execution time. With PJOG set to 0.3, Value Iteration took ~4 seconds to converge and Policy Iteration took ~9 seconds.

As one can see these maps are not nearly as correct as the PI and VI version. In fact in some places states lead to each other which is clearly not optimal. However, from start to finish the path is fairly well defined. It is not optimal by any means but is a fairly good path when considering the amount of data Q-Learning (with an epsilon-greedy approach) has access to. It is also clear that as the time increases the map is becoming close to the correct map, especially with regard to the Q-Value of each state.



Epsilon Greed (e=0.1) 4 seconds	Epsilon Greed (e=0.1) 9 seconds
	

The other two approaches demonstrate the strength and weaknesses of exploration vs exploitation. The random approach was only able to achieve 100 episodes in 4 seconds as compared to over 100,000 for the other 2 approaches. This make sense, since the random approach only stumbles onto the goal by accident. However, because of this it is able to learn a lot about the whole grid in the 4 seconds, achieve much higher similarity overall the ideal maps, especially in areas far removed from the initial state and goal state.

Greedy 4 seconds	Random 4 seconds
	

The Greedy approach on the other hand is very well defined around its suboptimal path but suffers when looking at the far reaches of the maze, since it only reaches those locations when non-deterministic actions lead in that direction.

This is why the epsilon-greedy approach is so powerful. Since Q-Learning only truly converges after having visited all the states an infinite number of times we must make tradeoffs. In the majority of cases when you have a start state and a goal state (as we do in this problem), you care very little about what to do in states that are far removed from what you believe is the optimal path. In this case the top half of the grid is largely irrelevant when trying to get from the start to the goal. Thus spending time improving those states is wasted time. However, if we only follow what we believe is the optimal path, we end up refining an area that in the end could be sub-optimal. Thus epsilon-greedy is a logical middle ground. It spends the majority of the time following what it believes to be the ideal path and then ventures out to look for a new path on occasion. This ends up yielding a good path in very little time.

## Conclusion

Policy Iteration and Value Iteration do an excellent job of maximizing our reward in a given problem. In our case Value Iteration outperformed Policy Iteration in what really mattered time. However, depending on the setup of the problem and the number of actions vs. states as well as other factors could yield different results. However, in many real world problems we are not going to have the luxury of knowing all the states and rewards. This is where Q-Learning really shows its strengths. In similar amount of time to VI and PI, Q-Learning can find fairly good solution to the problem, even if the arrived at solution is suboptimal. With enough time, Q-Learning will even eventually find the optimal policy (with an error threshold). It accomplishes all this with very little knowledge about the problem. By using an epsilon-greedy approach we can also fine tune this so that we use most of our time refining what we believe to be a good policy while also spending some of our time searching for new paths. This provides a good middle ground in the exploration vs exploitation debate.

In conclusion, Policy Iteration and Value Iteration are probably best when you have access to the required data. However, in a large number of problems, this data will not be available. In these cases Q-Learning perform very well at solving the problem at hand.

## Sources

<https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node22.html>

<http://www.cs.cmu.edu/~awm/rlsim/>