Assignment 4 - Reinforcement Learning

Jon Worley

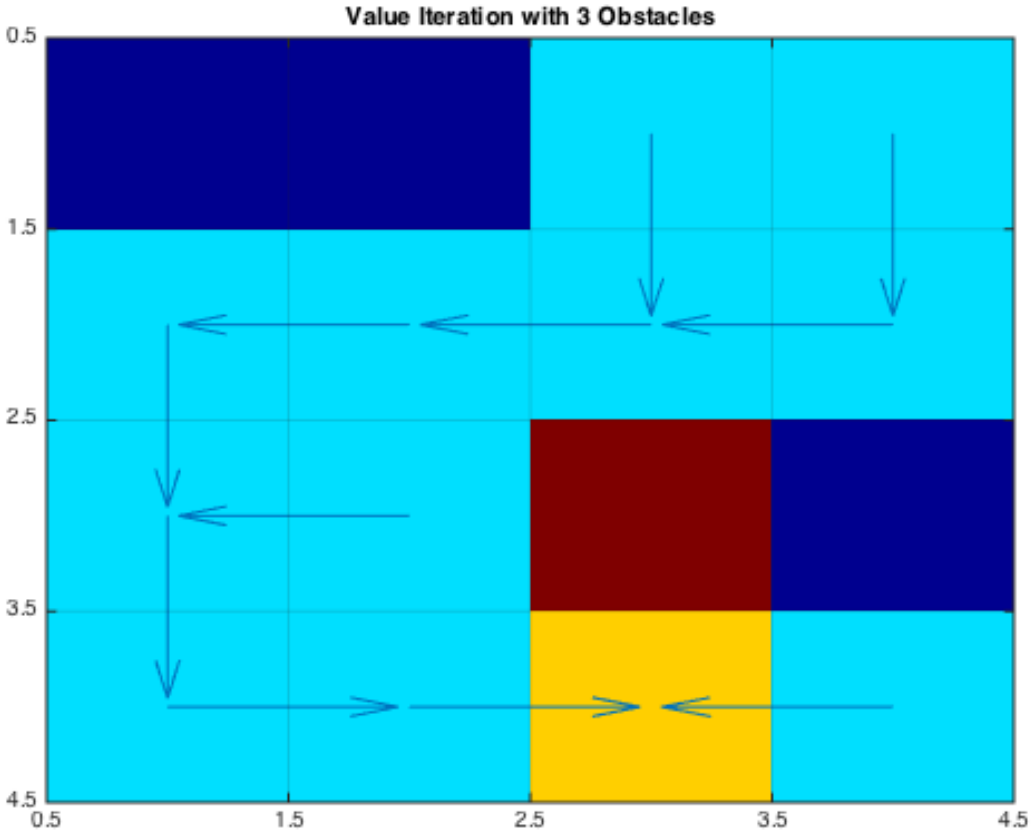
April 17, 2015

Introduction

For the final assignment for Machine Learning, we have been asked to analyze Markov Decision Process and Reinforcement Learning algorithms. Depending on the information that is available to the "agent", various evaluation techniques can be explored. Ideally, after an agent has trained itself suffieciently, it will have an optimal policy that it can follow for any situation it finds itself in. This policy dictates what actions that agent should take to maximize overall utility. To find an optimal policy, a reinforcement learning algorithm can search through policies and values of states. This paper discusses some pros and cons of various ways reinforcement learning can be approached.

Grid-World

For the assignment, I have chosen to analyze reinforcement learning algorithms using basic grid worlds. Grid worlds are simple to understand in that an agent exists in a 2 dimensional plane, and olny has the possibility of performing 4 actions in any given state until the agent finds a termination state.



Grid Size

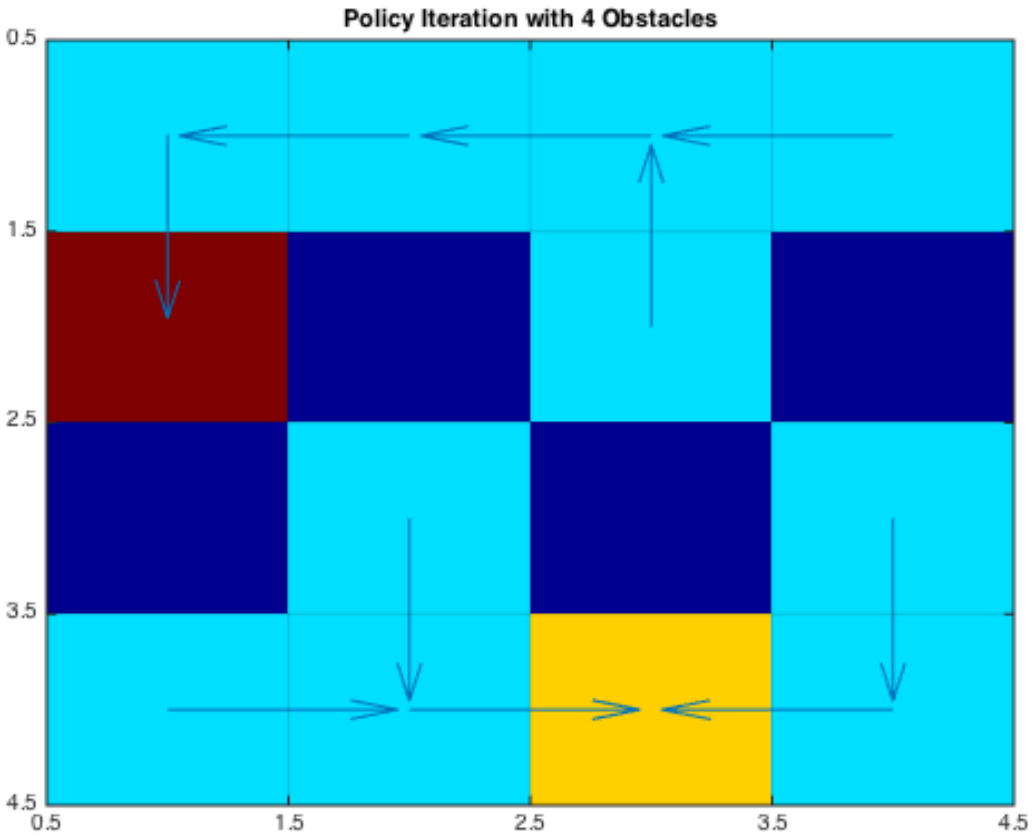
For the assignment, I have code itrate through 5 different sizes of a grid world. The size starts as a 2^2 X 2^2 (4x4) grid world and expands to a 2^6 X 2^6 (64x64) grid world. The grid world expands exponentially so that analysis of the time and iterations required to converge becomes aparent for each of the algorithms used.

Grid Termination

In each grid world, regardless of the size, there are two termination states. +1 and -1 (heaven and hell). Entering those states mean that there are no more rewards that an agent can acrue. Every other state in the grid world is set to -0.4 reward if it is not already deemed to be an obstacle. In all Gridworld plots, +1 is labeled as yellow, -1 is labled as red, and all other valid states are labled light blue.

Grid Obstacles

The algorithm that I ran initializes random obstacles in the path to try and make optimal solutions harder to evaluate. An obstacle state can not be entered, and has to be circumvented in order for the agent to reach the termination state. As the grid world's size increases, so does the number of obstacles. There is no guarantee that obstacles won't block off the good termination state which is always interesting to see how what an algorithm returns given an impossible task. These states are labled as dark blue in all gridworld plots. Below is a small state example of a gridworld where many of the states have no good solution



Value Iteration, Policy Iteration, and Q-Learning

The three algorithms that were used to explore gridworld policy solutions were value iteration, policy iteration, and Q-learning. These three algorithms are very similar, but are different in fundamental ways. In the MDPtoolbox, all algorithms use a transition matrix, a reward matrix, and a discount factor as inputs. Each of the algorithms return policy, value, or Q matricies that show the maximum performer that the algorithm was able to find.

Q-Learning Exploration Strategy

The exploration strategy used for the mdp toolbox Q learning is similar to a simulated annealing randomized optimization search. The algorithm runs a default of 10000 tests to find a good policy, and reinitializes every 100 transitions. Its action choice is greedy with increasing probablilty, and the update condition decays over time.

Small State Results

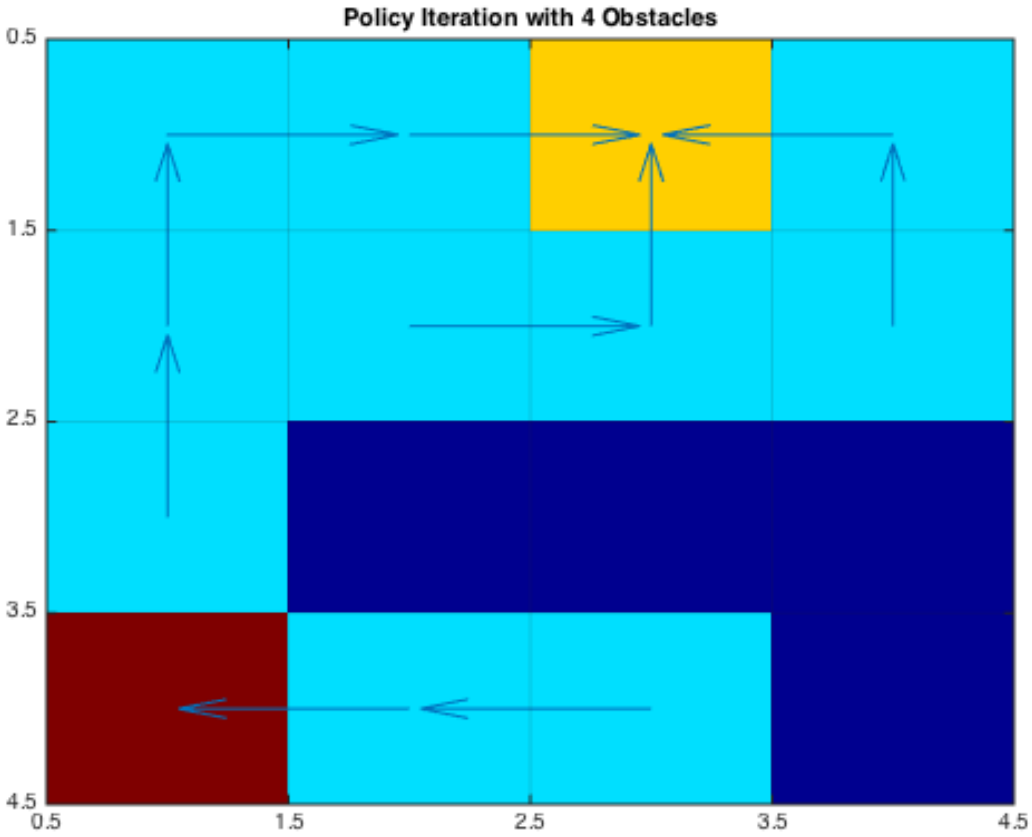
For all 16 gridworlds that were tested, poliy iteration and value iteration came up with the same results, though time and iterations varied widely, so all of the graphs that follow use either the policy iteration result or the value iteration result to illustrate both of those algorithms, while the Q Learning graphs show quite a different behavior.

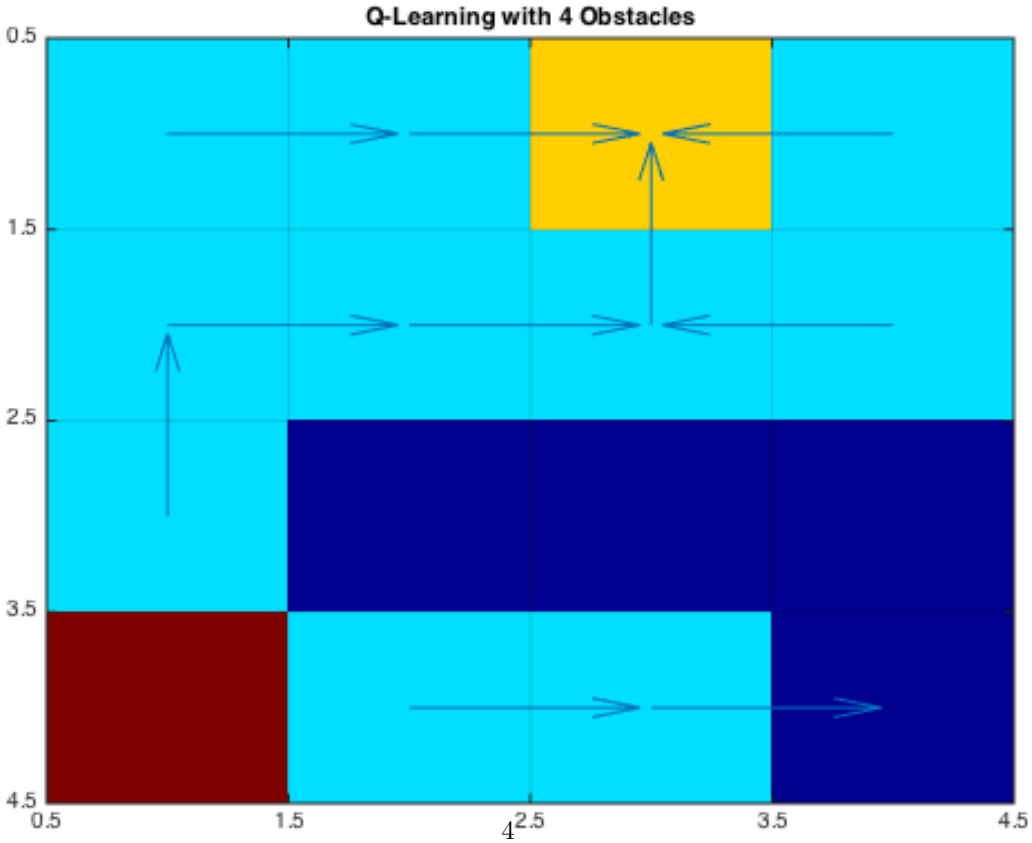
Avoid Forever, or End the Run

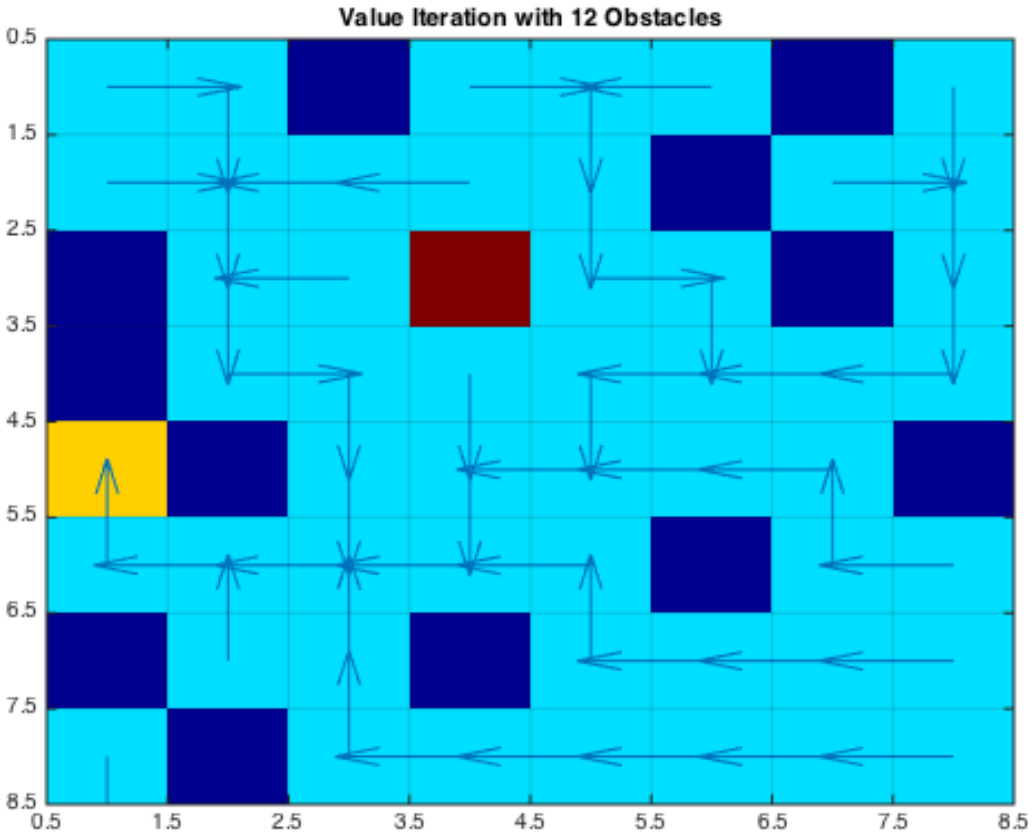
In the first pair of graphs below, there is a grid world where most of the states have an easy time finding the heaven termination state, but two of the states are trapped behind the hell state with no way to get out. Both the policy and value iteration find the optimal policy where the agent must choose to simply end the accumulation of negative reward as quick as possible. The Q learning algorithm decides to always run away from hell. The Q learning algorithm does this because the algorithm does not have a sum of all action state pairs built into its algorithm. States action pairs are more loosely coupled with other state action pairs in q learning algorithms than in policy or value iteration where a full set of state action pairs are considered before choosing a max.

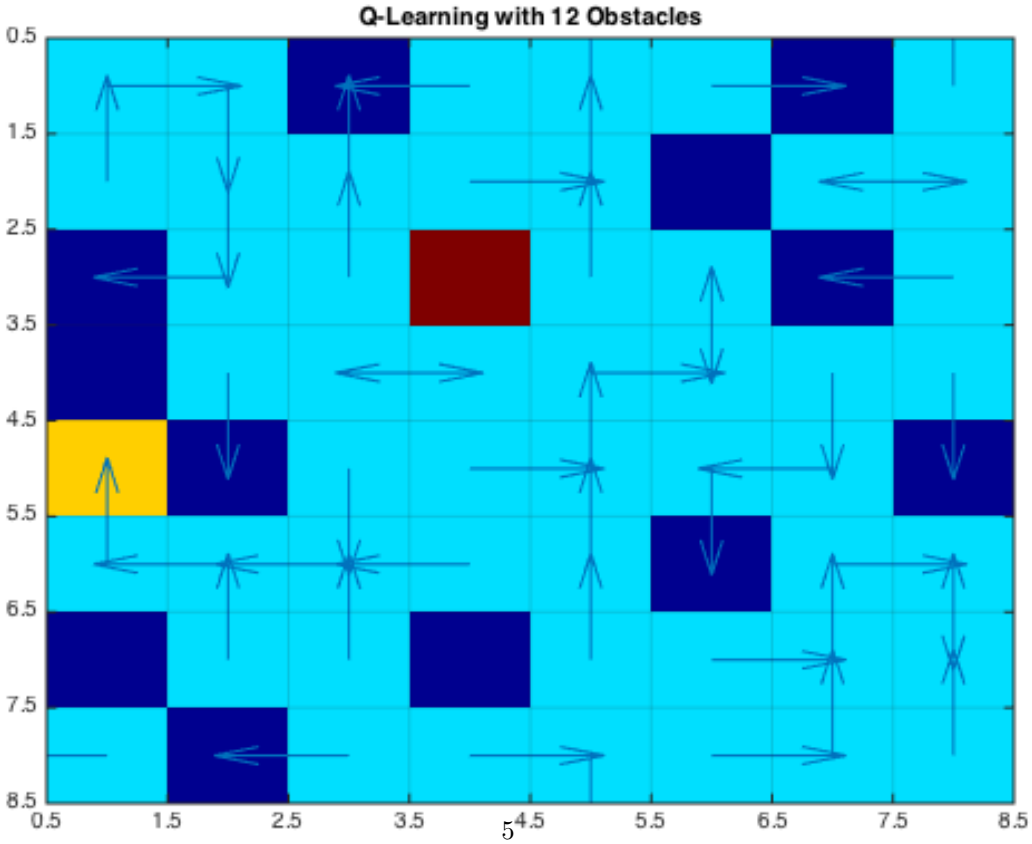
Peak Search with Q-Learning

In the second set of two graphs, there is an 8x8 grid world with 12 obstacles. The value and policy iteration algorithm finds the optimal solution again, where the Q-learning algorithm looks confused. I could have increased the number of iterations past 10000 for Q learning to perform better, but keeping it at 10000 allowed me to illustrate interesting properties of the algorithm. Most notably, the q-learning algorithm almost always finds the correct policy near the heaven state. This makes sense as those states have the highest value, and highest gain in performance if those state action pairs are correct. When evaluating other state action pairs that are further away, there is less gain in overall performance, so it would be harder to find the exact optimal performer. This illustrates Q-learnings ability to find local maxima very easily, and why it is important to consider randomized search algorithms when using Q learning.


Policy Iteration with 4 Obstacles
0.5
1.5
2.5
3.5
4.5
0.5
1.5
2.5
3.5
4.5


Q-Learning with 4 Obstacles
0.5
1.5
2.5
3.5
4.5
0.5
1.5
2.5
3.5
4.5


Value Iteration with 12 Obstacles
0.5
1.5
2.5
3.5
4.5
5.5
6.5
7.5
8.5
0.5
1.5
2.5
3.5
4.5
5.5
6.5
7.5
8.5


Q-Learning with 12 Obstacles
0.5
1.5
2.5
3.5
4.5
5.5
6.5
7.5
8.5
0.5
1.5
2.5
3.5
4.5
5.5
6.5
7.5
8.5

Q-Learning Disadvantage?

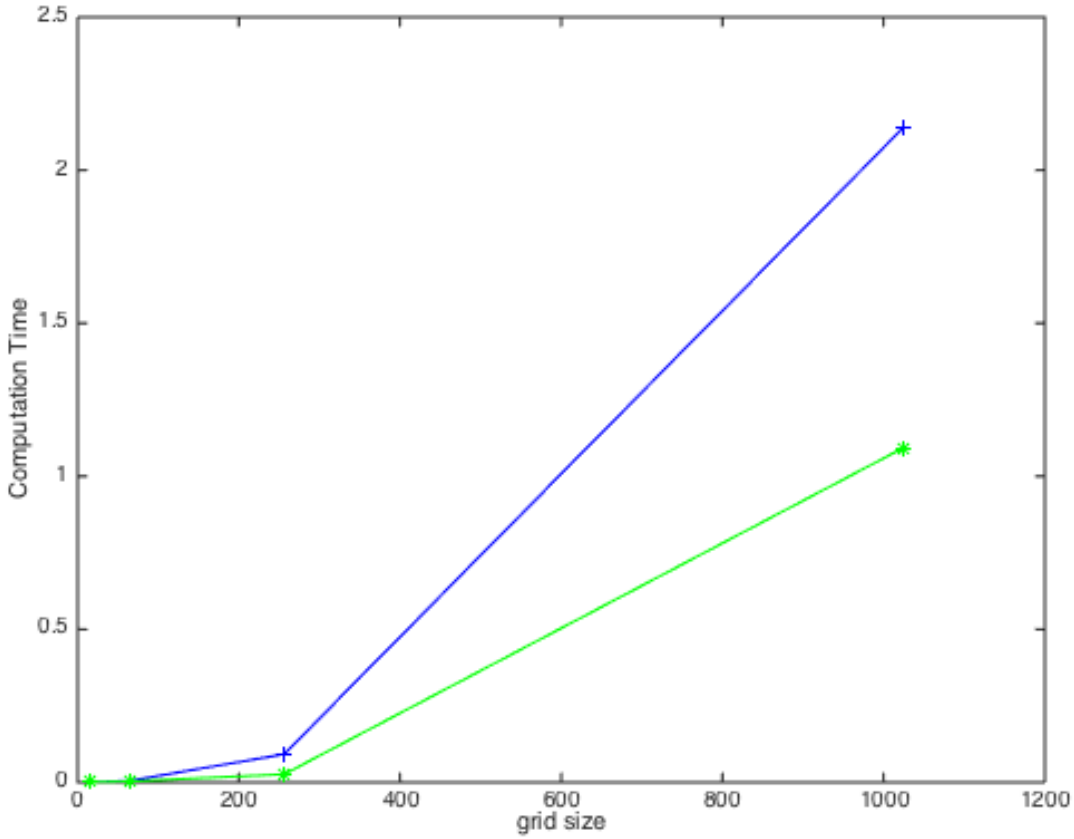
The plots that are being generated for analysis of the code are a little misleading. The grid world model that was created was created with a minimal amount of effort such that a policy could be evaluated. This means that the matrix structues have some 'valid' place holders for states, actions, and rewards that aren't actually valid. The policy and value iteration algorithms seem to be able to come up with correct values for the states that matter, and the analysis package ignores non-valid states. I wonder if the Q-learning algorithm has a disadvantage because it has to visit states and actions that are invalid and analyze a new Q value from that.

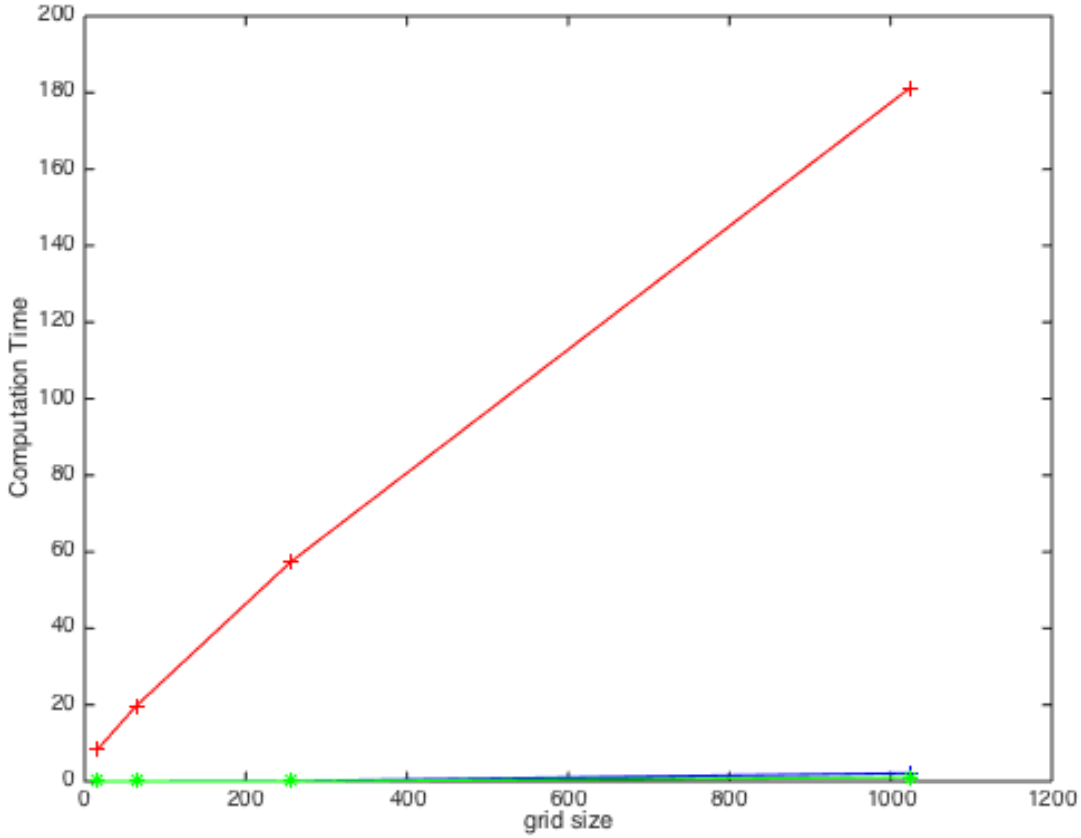
Large State Results

Computation Time

In the two graphs shown below, it is clear that some algorithms peform much quicker than others. In all graphs, Policy Iteration is represented by a green * line, Value iteration is represented by a blue + line, and Q learning is represented by a red +.

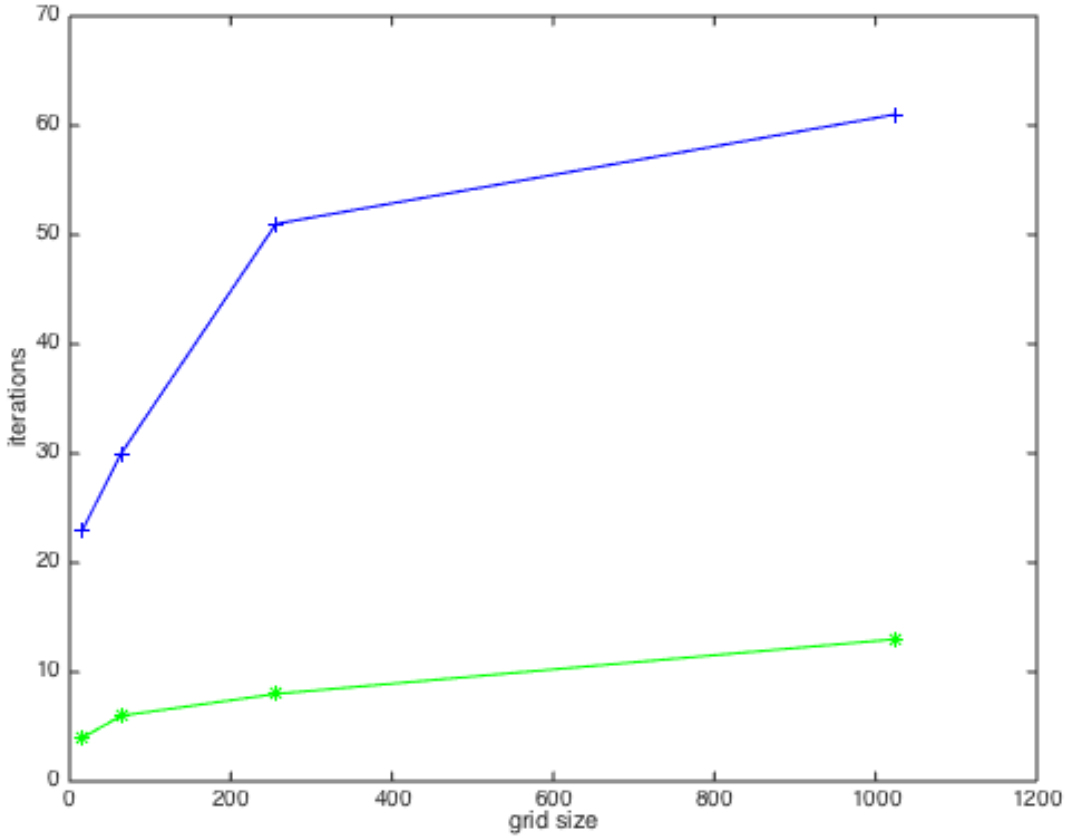
Policy iteration is the clear winner here. Where as Q-Learning is the clear loser.





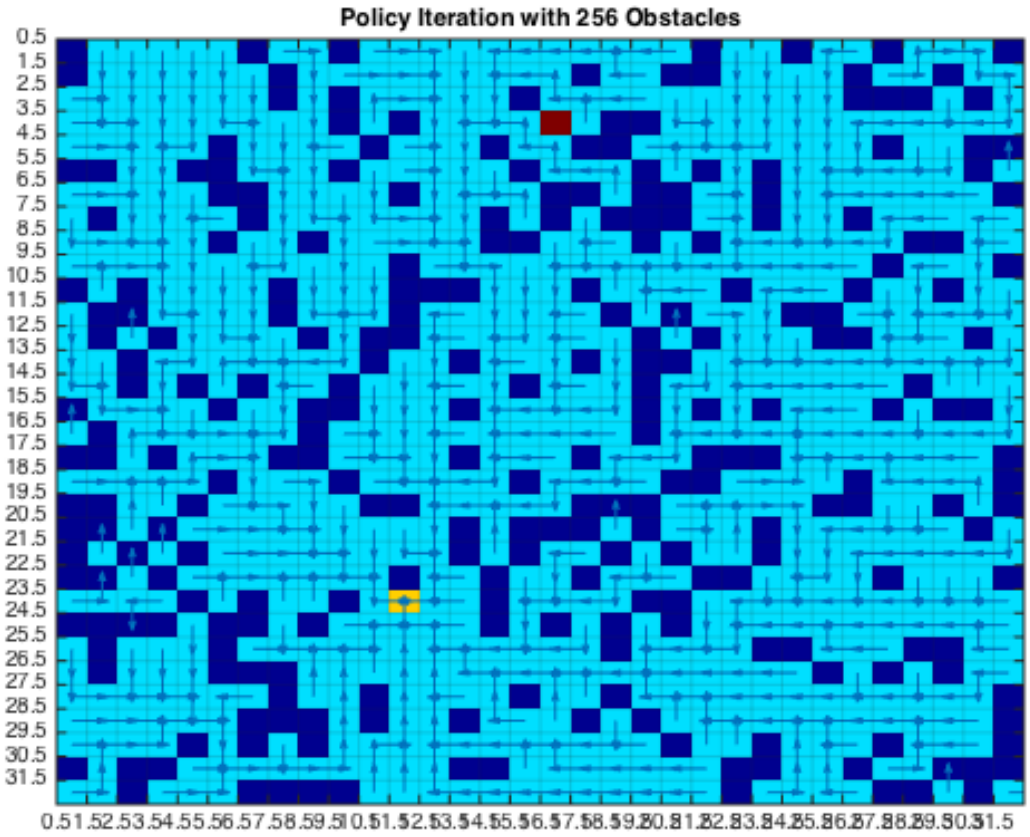
Iterations

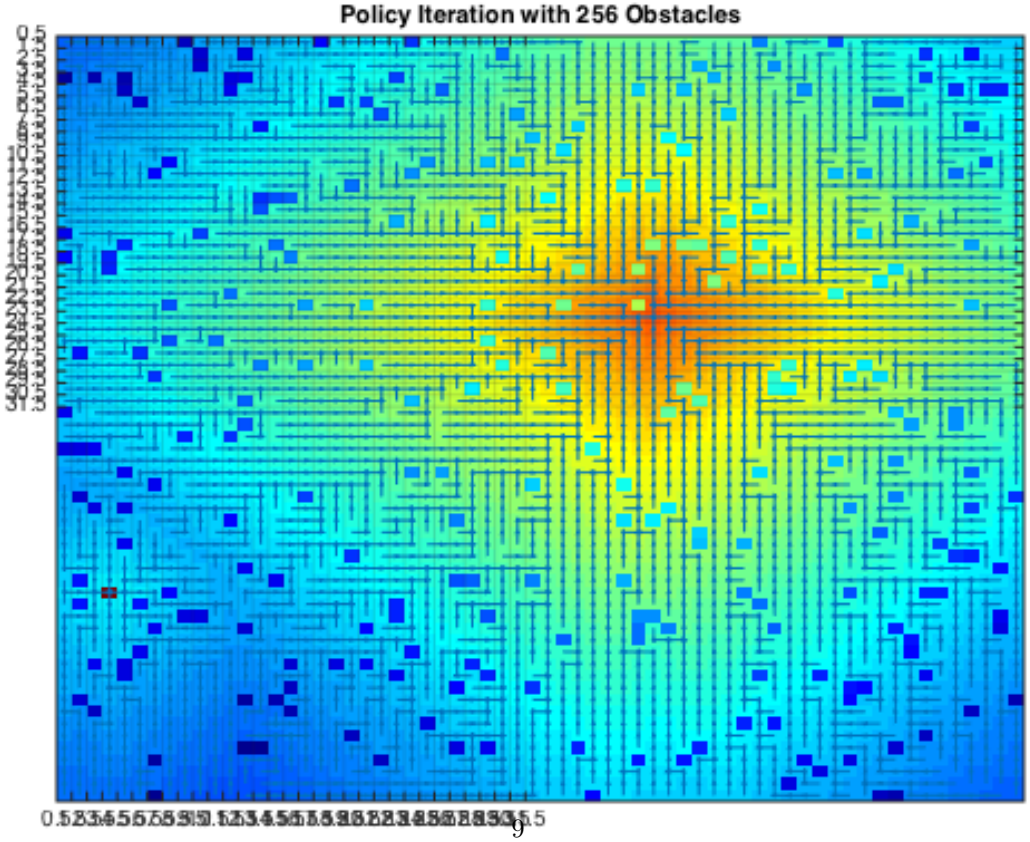
Below is a chart of iterations that it took for Value and Policy iteration to find an optimal performer. Q-Learning isn't on this graph because it was set to run a default of 10000 times for each iteration, and during that time it isn't guaranteed to find an optimal solution.



Optimal Policy and Optimal Value

One subject that I haven't touched on yet is the difference of an optimal policy output and an optimal value output. You can easily generate a policy output with an optimal value output, so they are quite closely linked in the type of information they hold. They are simply interesting to take a look at both to better understand the information that you have. Below are two large state simulations. The first one has just the policy output drawn as vectors leaving each valid state. The second graph shows the value of each of the valid states as well. Blue is a low value, and red is a high value. Invalid states (obstacles) change color, but are easy to see because there are no action vectors going in or out of them. Heaven state is depicted as a deep red, and hell state has a black-ish color near the bottom left corner. Not only does this optimal value graph visually pleasing, but this particular graph has some interesting qualities. If an initial state is near the Hell state, it is more optimal to head towards that instead of take the long journey to get to the heaven state. It's a little hard to see, but there are clearly two peaks in this gridworld. With two diferent optimal output states depending on the initial state.


Policy Iteration with 256 Obstacles


Policy Iteration with 256 Obstacles

Conclusion

These three algorithms are clearly related, but based on the analysis so far, it may seem like there would be no reason to use anything but Policy Iteration for any reinforcement learning task. Since the MDP Toolbox requires an acuarte transition matrix and reward matrix as inputs, this may be the case for any problem that you can use the MDPtoolbox with. The majority of problems that need reinforcement learning don't come in nice packages like grid world. They can be incredibbly difficult if not impossible to model accurately. Policy iteration requires that an entire set of state action pairs be evaluted at one time and compared to another individual complete policy whereas value iteration and Q learning can compute to see whether an individual state action pair was better than a previously picked one. This ability drastically reduces requirements needed to be able to perform reinforcement learning. The fact that Q-Learning also can be proved to converge after some time is definitely reassuring to know that given enough time, a better policy can be reached.