Georgia Institute of Technology

Assignment 4: Markov Decision Processes

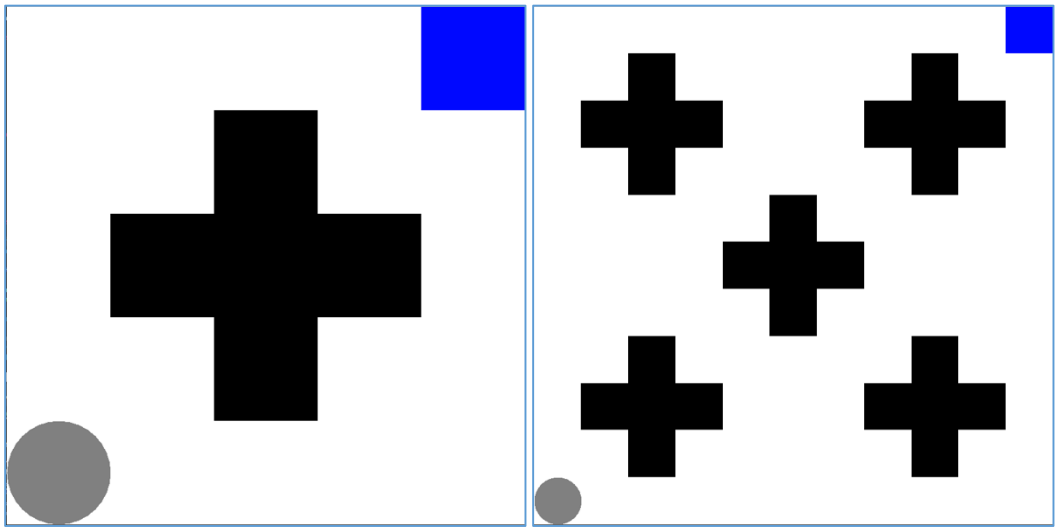
Juan J. San Emeterio
CS 7641: Machine Learning

Abstract

In the final assignment for this course, we have been given the task of developing two separate Markov Decision Processes: One with a small number of states and another with a large number of states. With the processes at hand, we were then to apply Value Iteration, Policy Iteration, and Q Learning in order to develop policies for our agents to reach the goal state. In order to develop the best possible comparisons between the two different processes with varying state sizes, I tried to keep the processes as similar as possible. This is why I chose to develop two different difficulties of the same game, Grid World, where the agent must traverse a grid-like world from the starting location to the goal location while avoiding obstacles and maximizing its reward. By keeping the two processes as similar as possible, I was able to derive very interesting insights about the subtle differences and similarities between the three algorithms. Given the fact that Value Iteration and Policy Iteration both require the transition probabilities and reward functions in order to calculate optimal policies, it was fairly unsurprising to see that these two algorithms typically converged to the optimal policy in much fewer iterations than Q Learning, which had to derive that information from experience. Additionally, Value Iteration and Policy Iteration, given their almost identical paradigms, typically converged to the same optimal policy in the same number of iterations. While Value Iteration and Policy Iteration were clearly extremely similar, Q Learning deviates in both its algorithmic paradigm and results. More specifically, while Value Iteration and Policy Iteration are given a lot of information about the process upfront, Q Learning must derive this information from experimentation by interacting with the process. This ultimately translates to a more difficult problem that requires more iterations and time in order to converge to an optimal policy like the ones calculated by Value Iteration and Policy Iteration. In the Grid World with only 20 states, Value Iteration and Policy Iteration were able to converge to an optimal policy in only 4-7 iterations while Q Learning required around 19 iterations in order to achieve a similar policy. In the high difficulty Grid World, there were now 96 possible states and this magnified the differences between the algorithms' abilities to converge on optimal policies. While Value Iteration and Policy Iteration were able to converge on optimal policies by the 10^{th} to 13^{th} iteration, Q Learning did not come close to converging on an optimal policy until the $35{,}000^{th}$ iteration and even then, the policy still took more steps on average to reach the goal than the policy calculated using the other algorithms. While it is clear that with a higher number of states, there is an exponential increase in number of iterations required for Q Learning to converge on an optimal policy, it is important to note that this algorithm is converging without a transition probability matrix or reward function given up front. If the transition probability matrix and reward function are available up front, you can't go wrong with using either Value Iteration or Policy Iteration to calculate an optimal policy. If the transition probability matrix and reward function are not available then Q Learning is clearly the best algorithm to use.

Markov Decision Process Problem Description

In this assignment, I will be exploring a classic problem in Markov Decision Processes called Grid World. This hypothetical landscape consists of an agent (gray circle), walls (black squares), and a goal (blue square), or terminal state. The objective of this Markov Decision Process is for the agent to traverse the Grid World and reach the terminal state in the least number of steps, or actions, possible. In order to visualize the specific scenarios being tested in these experiments, I have generated visual representations of the two variants. The grid on the left is the "easy" version where there are only 20 possible states and only one obstruction between the agent and the goal. The grid on the right represents the "hard" scenario where there are 96 possible states and there are 5 obstructions blocking the agent from reaching the terminal state.



The agent's incentive to reach the terminal state in the least number of steps is dictated by the reward function, or in most cases the cost, of each additional action. In these instances of Grid World, the agent loses 1 point for every action that does not immediately land it on the terminal state. If the action the agent takes lands it on the terminal state, the agent is awarded 100 points. In order to make this scenario a true Markov Decision Process, there also needs to be a chance that the actions performed have inherent probabilities of success and failure. This is made possible by having a transition matrix that dictates the probability of success that the agent will be able to move successfully in the direction that it intended. Specifically, in this scenario, there are four different actions described by movements north, east, south, and west. A successful movement in the direction intended has a probability of 80% while a movement in any other direction has a probability of 6.67%.

By combining all of the components above, we now have a fully qualified Markov Decision Process that can be optimized algorithmically using Value Iteration, Policy Iteration, and Q Learning.

Why Grid World?

At first glance, Grid World may appear to be a trivial problem with no real world application but this is very far from the truth when you consider the possible applications in automated robotics. In the real world, making an automated robot or vehicle would serve little to no purpose if it could not be placed in a new environment and adapt to it's new surroundings. By adapting, I mean that the robot should be able to avoid obstacles and achieve goals without explicit programming. The best example that comes to mind is a self driving vehicle that needs to make hundreds (if not thousands) of decisions per second in order to achieve its primary goal of delivering its passengers to their destination efficiently and most importantly, safely.

In the context of designing a self driving vehicle, much like in grid world, there will be a starting location and a destination that will be placed on a map. Even though the maps used in Grid World consist of several dozen states, you could instead take a map, much like you would find in Google Maps, and convert every pixel between your starting location and destination into a possible state. This sort of broad stroke approach would mean that you would now have possible states in every road way as well as non-drivable areas such as buildings, parks, and bodies of water. It would then be the task of a separate algorithm to develop a reward function that assigns severe penalties for entering states that are either impossible, such as driving over buildings, or dangerous, such as driving on the opposite side of the road.

In Grid World the agent has to make the decision of taking 1 of 4 possible actions. In terms of a vehicle, the agent would have to decide the rate of acceleration/deceleration, the speed at which to remain, and the degree angle to turn the wheels of the vehicle in order to reach the next desired state. Similar to Grid World, the agent will also be facing measurable probabilities of success in switching states due to the possibility of deterioration of the vehicle over time. This will mean that the agent will have to compensate for these deteriorations much like the agent in Grid World must compensate for its likelihood of success.

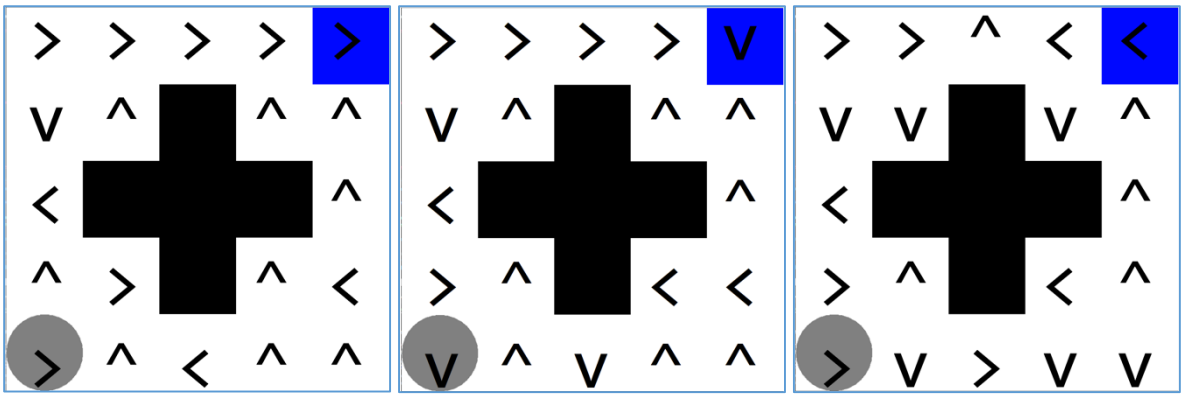
Overall, this is a grossly oversimplified discussion of what it would take to develop the artificial intelligence required for a self-driving vehicle to work in the real world but in principle, the two scenarios are very similar.

Grid World: Low Difficulty

In order to better gauge the differences between Value Iteration, Policy Iteration, and Q Learning I will first apply the algorithms to the lower difficulty Grid World that is seen below and later apply the same algorithms to the harder problem. The first three images below represent the policies generated by the three algorithms after running only a single iteration. As you would expect, Value Iteration and Policy Iteration have both generated very similar policies with only a few exceptions. Surprisingly, the policies generated by Value Iteration and Policy Iteration are reasonably good and clearly show movement towards the goal. On the other hand, the policy calculated with Q Learning suffers from many policies that move the agent directly towards walls, which is clearly not optimal.

1 Iteration:

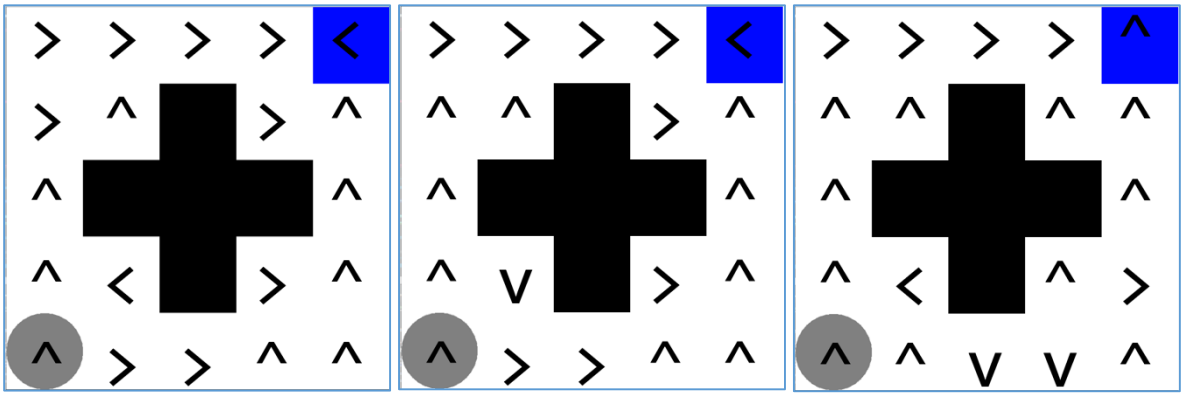
Value Iteration | Policy Iteration | Q Learning



After running each of the algorithms on the same problem for 100 iterations, the policies changed drastically for the better. Value Iteration and Policy Iteration clearly converged on the optimal policy regardless of whether the initial action was to move north or east. Q Learning was also able to find the optimal policy but only if the initial action was to move north around the obstacle. If you look at the Q Learning policy below, you will notice that if the agent moves east as an initial action, the policy is to quickly reroute towards the northern route over the obstacle in the middle of the map. If the agent is unlucky enough to move east twice it will then be subject the policy that indicates the agent should move south towards the wall. Although, this failure scenario is unlikely, it is immediately clear that this policy is less optimal than the other two policies calculated by Value Iteration and Policy Iteration

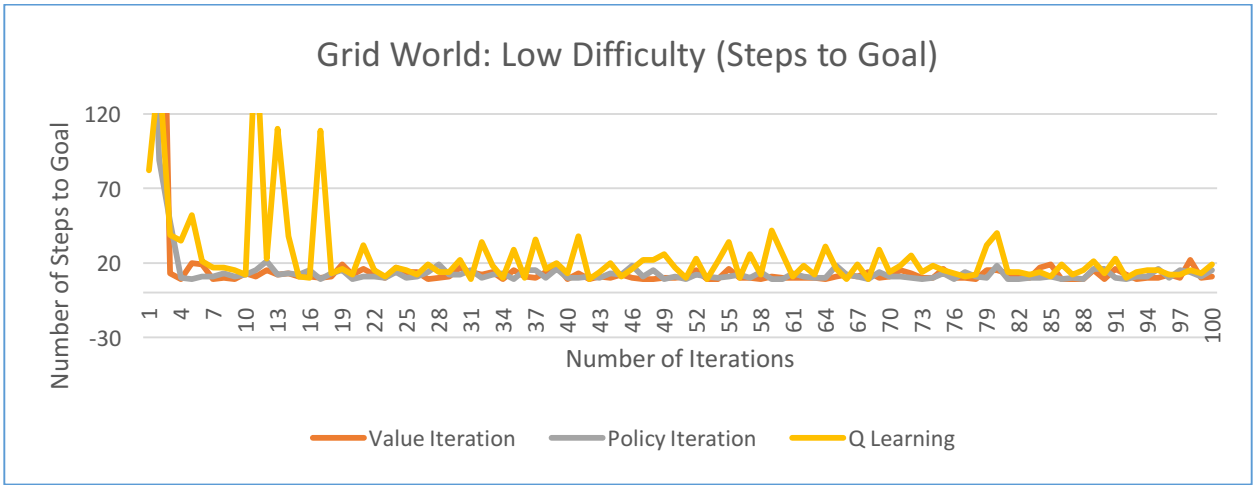
100 Iterations:

Value Iteration | Policy Iteration | Q Learning

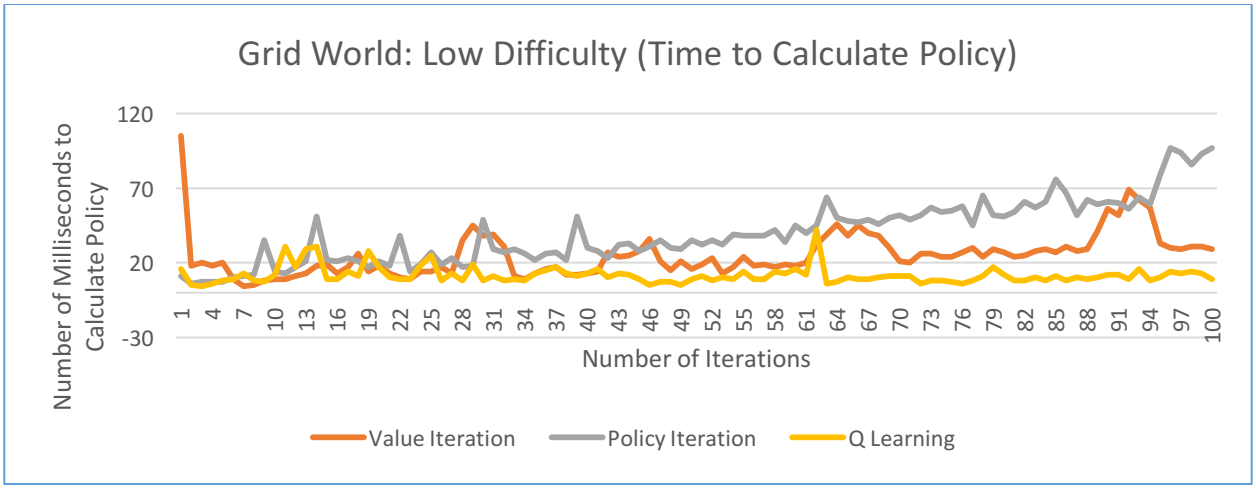


In order to graphically demonstrate the convergence of the policies, I generated the graph below which plots the number of steps required to reach the goal after a certain number of iterations. (Please note that because there are no terminal states with negative rewards, the number of steps taken to reach the goal state correlate directly with the reward function that is typically used to measure convergence. By using the number of steps instead of the reward function, the graphs are cleaner and easier to interpret.) In other words, each algorithm was allowed to run N iterations, a policy was then generated, and a simulation was run using that

policy in order to calculate the number of steps required to reach the goal state. From the graph below it becomes very clear that Value Iteration (red) and Policy Iteration (blue) both converged to the optimal policy in only 4-7 iterations. In comparison, Q Learning did not appear to converge until the 19th iteration and even then, the number of steps to reach the goal were still somewhat volatile with occasional spikes even as the number of iterations increased.



Another interesting comparison that is facilitated by the graph below is the number of milliseconds required for the algorithms to run and generate a policy given a certain number of iterations. Not surprisingly, the three algorithms appear to be running in linear time. Policy Iteration clearly is taking the longest to run, most likely due to the policy generation step that occurs during each iteration. By avoiding the policy generation step, Value Iteration does run a little bit faster than Policy Iteration. Interestingly, Q Learning appears to run at a constant rate of 15-20 milliseconds regardless of the number of iterations. The speed at which the algorithm runs is probably due to the fact that the algorithm, throughout the series of iterations, is not actually performing any computation but rather simply hashing the actions taken and the rewards achieved until a policy is calculated at the very end.

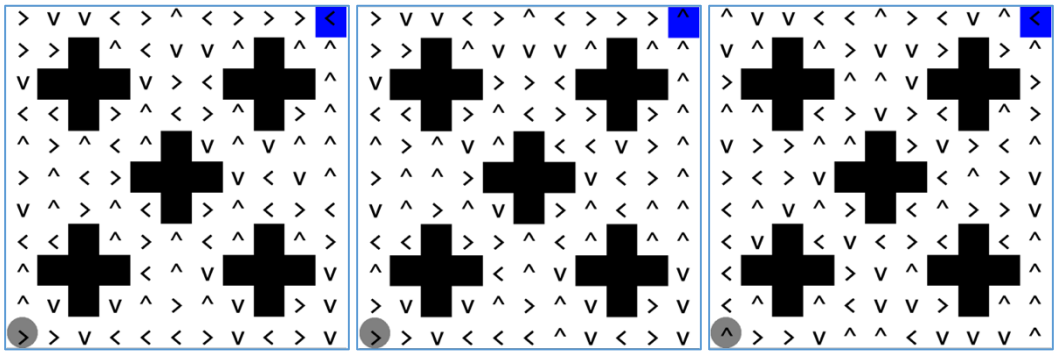


Grid World: High Difficulty

In order to increase the level of difficulty involved with determining an optimal policy, I modified the original map to include 76 more possible states and 4 more obstacles to traverse. The increase in difficulty is immediately apparent after generating the policies calculated after a single iteration. Whereas previously, the policies offered generally good guidance on how to arrive at the goal, now they are only somewhat better than random. In the cases of Policy Iteration and Value Iteration, the policies generated only offered sensible guidance when the agent was within 4-5 spaces from the goal state as seen below. Q Learning performed worse, offering no such guidance unless the agent was within 1-2 spaces from the goal state as seen below.

1 Iteration:

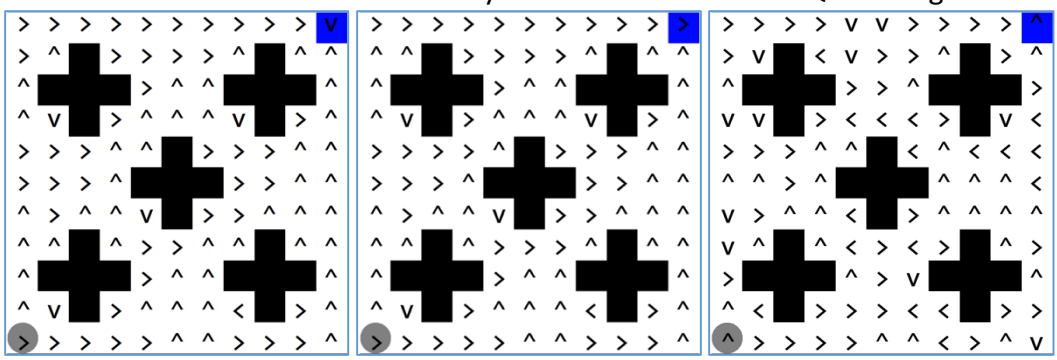
Value Iteration | Policy Iteration | Q Learning



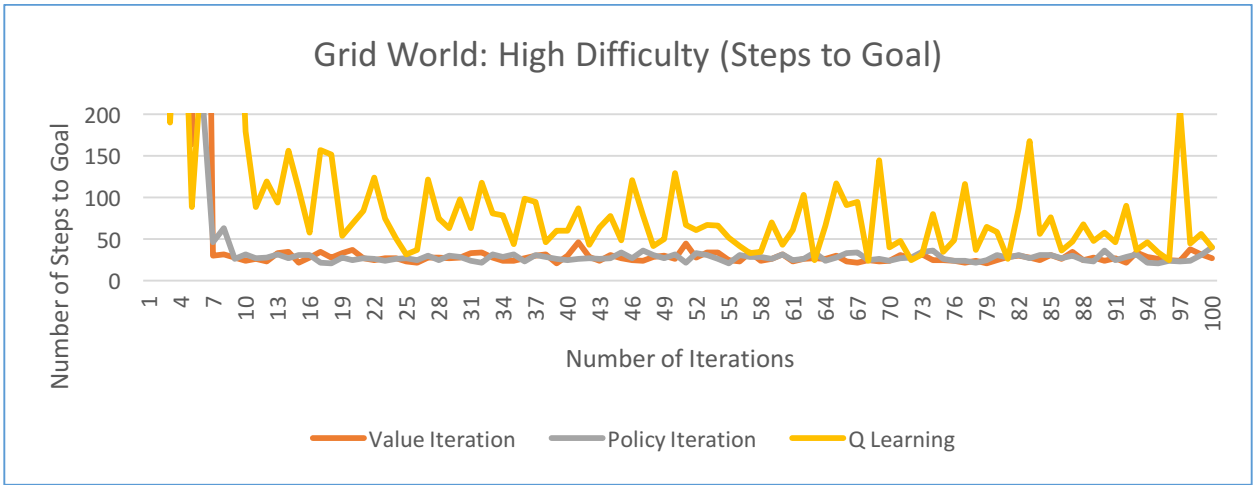
Next, I allowed each of the algorithms to run over 100 iterations and calculate a new policy. The policies generated can be seen below. With very few exceptions Value Iteration and Policy Iteration have almost identical policies and have clearly converged to the optimal policy. Q Learning, however, seems to have only mildly improved with many of the arrows still pointing away from the goal. This is definitely an effect of the increased state space. Whereas, previously the Q Learning algorithm had a reasonably good chance of visiting all of the states in the easy grid world, now visiting all of the states is a challenge that can only be circumvented by performing more iterations, which will be discussed later on in this report.

100 Iterations:

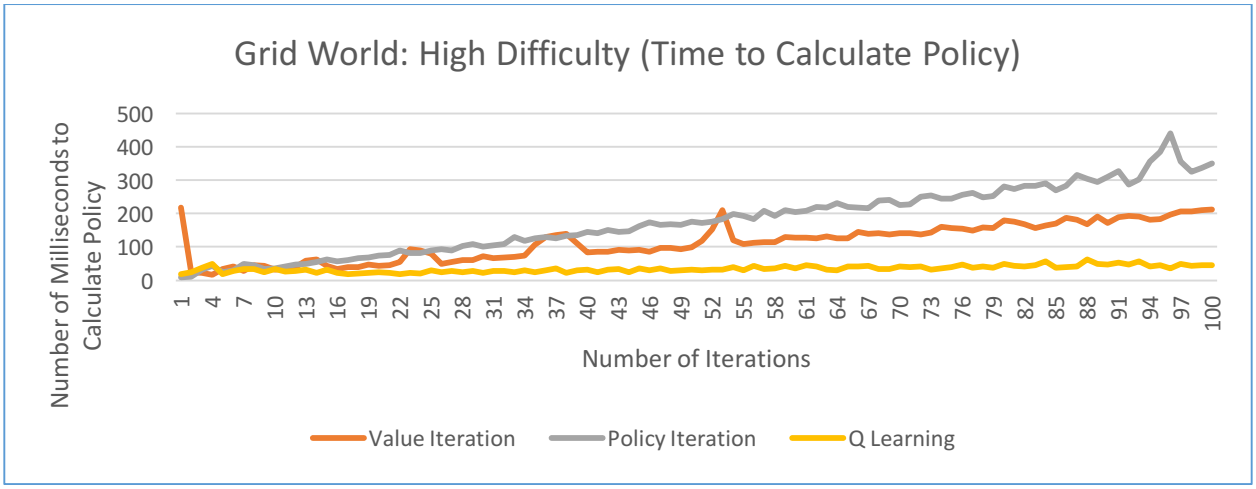
Value Iteration | Policy Iteration | Q Learning



In order to compare the three algorithms' abilities to converge to an optimal policy, I graphed the number of steps required to reach the goal given the policy calculated over N iterations. (As mentioned previously, please note that because there are no terminal states with negative rewards, the number of steps taken to reach the goal state correlate directly with the reward function that is typically used to measure convergence. By using the number of steps instead of the reward function, the graphs are cleaner and easier to interpret.) From this graph, it is clear that Value Iteration and Policy Iteration both converge on optimal policies by the 10th to 13th iteration averaging 20-30 steps to the goal. Q Learning, on the other hand, doesn't appear to converge at any point throughout the experiment seeing volatility throughout the experiment and as many as 205 steps to reach the goal using a policy generated after 97 iterations.

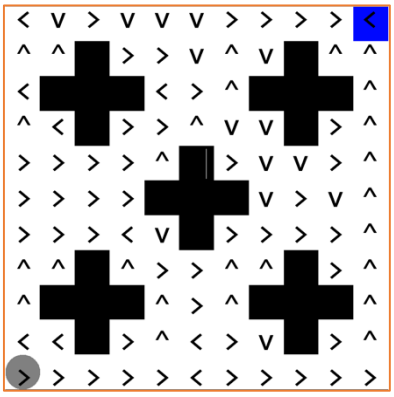


Much like in the low difficulty Grid World, the time required to perform the experiments, regardless of the algorithm, is linear. Policy Iteration is still the slowest algorithm with Value Iteration as the second slowest. Q Learning, also follows a similar pattern as before where throughout the experiment, the time to generate a policy is constant regardless of the number of iterations.



Grid World: High Difficulty (Q Learning Expanded)

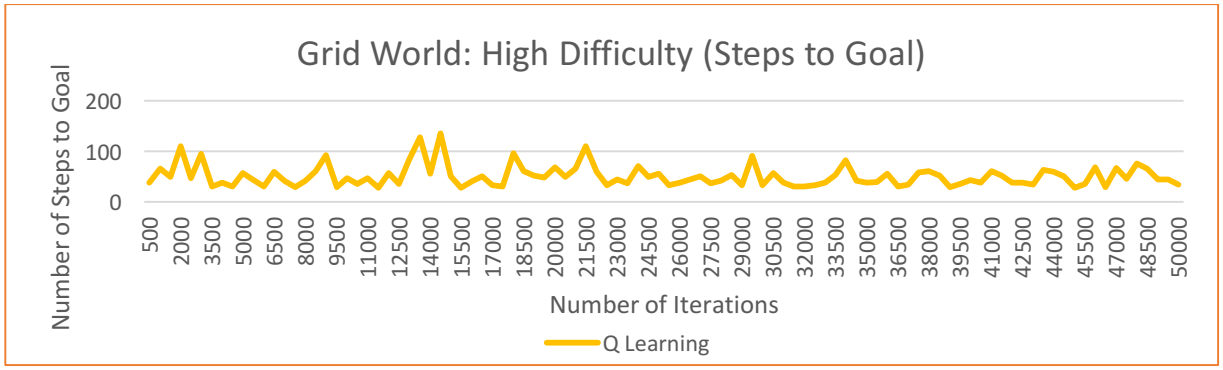
In the previous section, when I compared the three algorithms head-to-head on the high difficulty Grid World, it was clear that Q Learning was not able to extract an optimal policy when given 100 iterations. In this section, I wanted to explore how many iterations it would take for the Q Learning algorithm to converge on an optimal policy, how the time required to compute this policy would change, and see what the optimal policy map looked like.



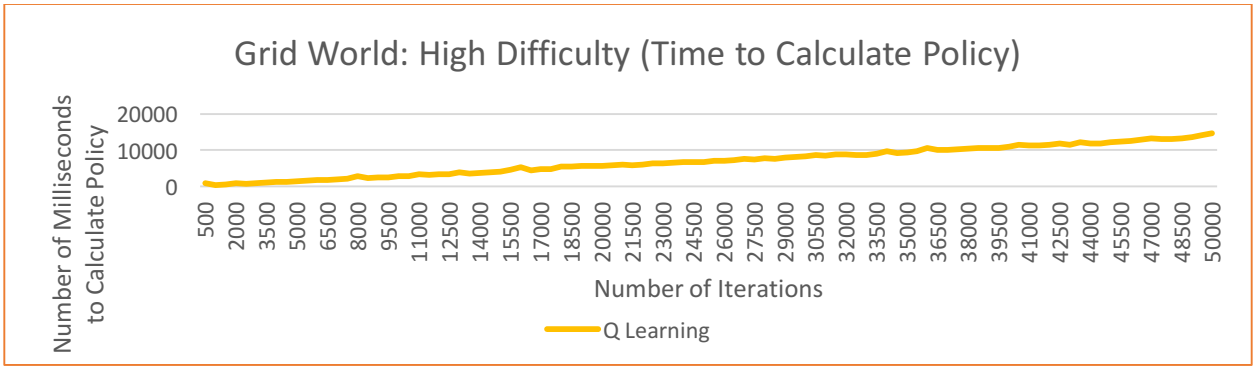
In order to guarantee some level of convergence, I decided to run the algorithm for 50,000 iterations. The policy map that resulted from this experiments can be seen on the right. Unlike the policy map extracted in the previous section, when Q Learning was only run 100 times, this time around the policy looks a lot more decisive about the directions the agent should move from the starting location all the way to the goal. While some of the policies for several states are counter productive in achieving the goal, generally speaking it is a much more attractive policy.

Similar to previous sections, in order to see when the algorithm converges I have measured how many steps it took to reach the goal given a policy extracted after N iterations. In the last section Value Iteration and Policy Iteration were able to converge to the optimal policy in around 10 iterations. Given this optimal policy, the agent was able to consistently arrive at the goal in 20-30 steps. Conversely, Q Learning even at the 97th iteration still offered a policy map that took the agent 205 steps to reach the goal.

As evidenced by the chart below, it is clear that by the 500th iteration, the policies generated by the algorithm are no longer even close to requiring 205 steps to reach the goal, however, there is still a significant amount of variability. This can be seen by the fact that even after 14,000 iterations the agent still, on occasion, can take 130 steps to reach the goal. It isn't until the algorithm has run 35,000 iterations that I can say with some degree of certainty that the algorithm has converged since it is at this point that the agent is consistently reaching the goal in 40 or so steps. Even then, however, this is still sub-optimal when compared to Value Iteration or Policy Iteration, which were consistently reaching the goal in only 20-30 steps.



An interesting effect of running Q Learning over a higher number of iterations were the changes in the amount of time required to compute a policy. In previous sections, when the number of iterations the algorithm was allowed to run was limited to 100 iterations, it seemed like the policy was being calculated in constant time. The graph below still indicates that this is the case for around the first 2,000 iterations. After those first 2,000 iterations, it seems that the time to calculate a policy begins to increase in a linear fashion. This can be explained by the fact that during the first 2,000 or so iterations, it is possible that the agent is still exploring many states and sequences for the first time. As the number of iterations increase, so do the number of conflicting sequences, and for this reason, so does the time to compute the optimal policy.



Conclusion

Given the fact that Value Iteration and Policy Iteration both require the transition probabilities and reward functions in order to calculate optimal policies, it was fairly unsurprising to see that these two algorithms typically converged to the optimal policy in much fewer iterations than Q Learning, which had to derive that information from experience. For this reason, it's hard to really compare these algorithms since the problems that they are trying to solve are inherently different in regards to their inputs. While it is clear that with a higher number of states, there is an exponential increase in number of iterations required for Q Learning to converge on an optimal policy, it is important to note that this algorithm is converging without a transition probability matrix or reward function given up front. If the transition probability matrix and reward function are available up front, you can't go wrong with using either Value Iteration or Policy Iteration to calculate an optimal policy since they mostly follow a similar paradigm. If the transition probability matrix and reward function are not available, then using Q Learning is clearly the best algorithm to use.