Napa (Tan) Vananupong

Dr. Brian Hrolenok

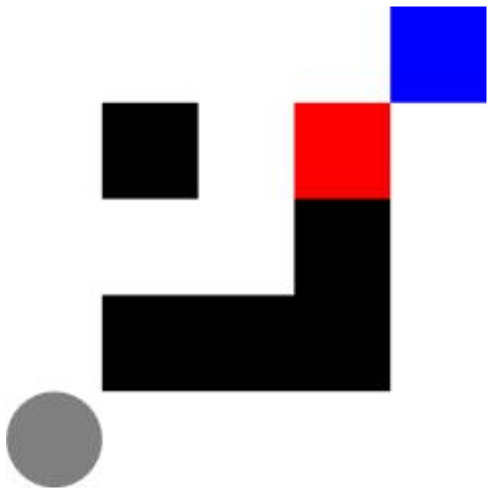
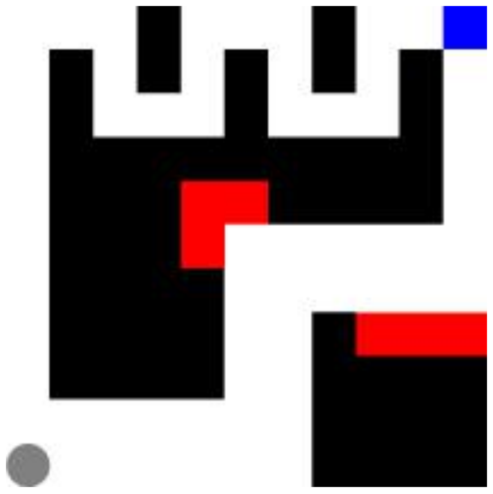
CS 4641

April 19 2019

Assignment 4: Markov Decision Processes

Introduction to the 2 MDPS:

For the last assignment of the semester, we are wrapping up our learning of machine learning techniques by using the various forms of function approximation and applying them to decision making processes – more specifically, the Markov Decision Processes. The proceeding document will present two Markov Decision Processes, which I will be referring to as MDPs 1 and 2, MDP 1 being the process with a small number of states and MDP 2 being the process with a comparatively large number of states. Furthermore, I will go into detailed analysis of the performances of the three reinforcement learning algorithms on each of the MDPs. The three reinforcement learning algorithms I will be comparing are value iteration, policy iteration, and Q-learning. The performances will be compared with a strong variation of metrics, namely iterations and runtime. The MDPs that I used are Grid World problems, in particular the ones from Brown-UMBC Reinforcement Learning and Planning (BURLAP) Java library. All algorithms and software were from this BURLAP. While a world grid problem is not exactly ‘interesting’ in terms of personal hobbies and interests, I do find it interesting for the class and for machine learning in general because it has a lot of practical real world applications I can use it for, and its also a good problem to test the algorithms on. Furthermore, it is very interesting in terms of real world applications if I ever had to work in AI game design, I think grid world problems are extremely relevant because they involve finding one or multiple optimal paths to navigate through an environment and minimize reward, which would be very applicable in games where you try to make an AI agent search for an object or target in an environment in the minimum amount of time.

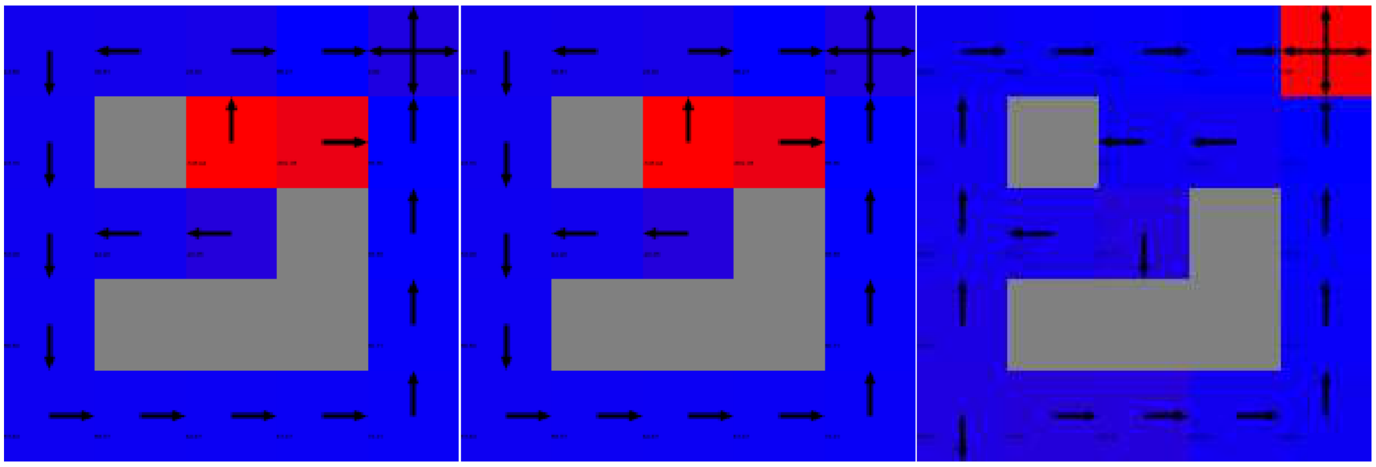
Figures 1 and 2: EasyGridWorld (left) with 15 states and HardGridWorld (right) with 63 states.

Comparison of Policy Iteration vs Value Iteration vs Q Learning

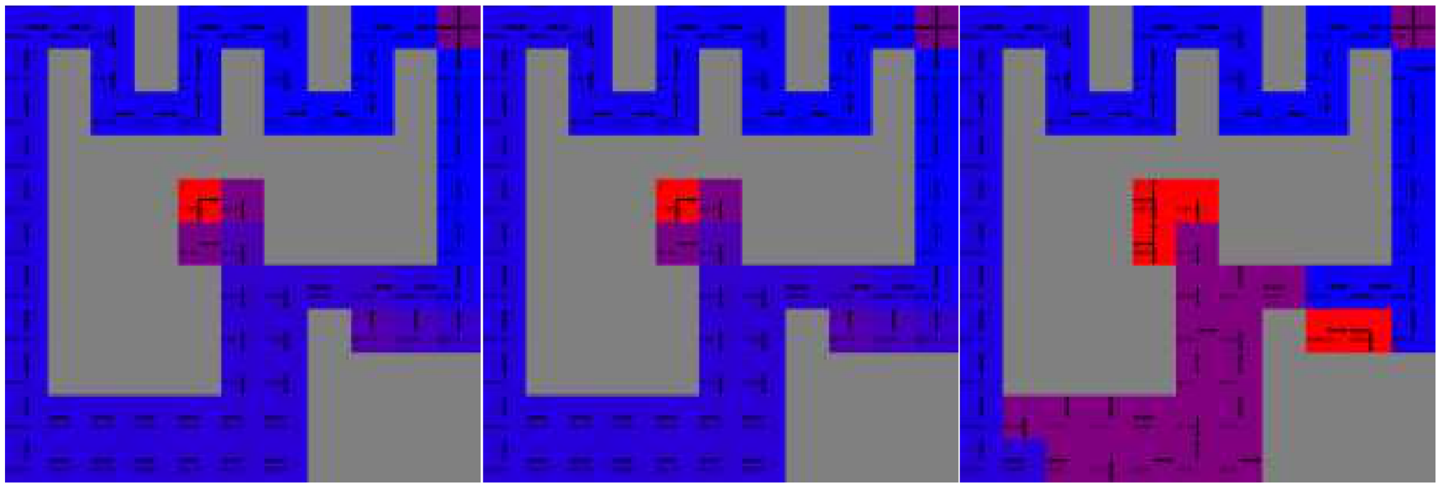
I will be applying the policy iteration, value iteration, and Q-learning algorithm to MDP 1 (easyGridWorld) and MDP 2 (hardGridWorld). Policy iteration works by iteratively updating the policy function while value iteration works by iteratively updating the value function. Policy iteration usually converges faster than value iteration due to the fact that in value iteration, there is usually a threshold amount of iterations that the policy will just not change after it hits that threshold. Meanwhile in policy iteration, the algorithm simply stops when the policy stops changing from the previous iteration. On the other hand, value iteration allows the agent to pre-plan its policy before actually interacting with the environment. It does this by iteratively updating the value at each state. Because we have to wait for the value function to converge, value iteration can sometimes take a while to run, but for the small state spaces we are using it won't be an issue.

The last algorithm I applied to these problems is Q-Learning with a lookup table. The lookup table, also known as the Q table, is a way to store information about the optimal action to take for a given state. The agent will initially start with a Q table of all zeros, and as it explores the world grid, it will learn which actions and directions to take to get the optimal rewards in different states. This is extremely different than the other two algorithms because the agent "learns" instead of just computing utilities (i.e. it learns as it plays rather than planning its strategy beforehand). Additionally, the Q learner has to balance something known as exploration versus exploitation. Essentially, at first its best for the agent to randomly explore its options for actions so it can get a feel for how they reward the agent. Then later on once the agent has found a decent policy via the Q table, it will choose less and less of its actions randomly. I chose the agents choice of random action by using an epsilon greedy strategy. The agent will choose a reward randomly if a generated random number is less than epsilon, and then we decay epsilon over time to that the agent acts less randomly over time. After running this strategy, the result was that allowing the agents to randomly explore its action space significantly improved their results – without exploration in the easy problem the Q-learner completely failed to learn anything and find any answer.

Below are graphs for policy iteration, value iteration, and Q-learning on each of the Grid World problems. As you can see, policy iteration clearly converges faster than value iteration for both MDPs. This is due to the aforementioned reason. It only took 16 iterations to converge for the MDP 2 (hard) even though there is literally 4 times more states than MDP 1 and 6 iterations for MDP 1 (easy). We can see that the number of iterations did not affect total reward of the policy and value algorithms, which is expected because both policy and value iteration will converge to the optimal policy, albeit at different times. Value iteration and Policy iteration converged to the exact same optimal answer for both MDPs, as you can see from the visualization of their optimal answers below. On the other hand, the Q-learning algorithm did not end up with the same optimal answer, also for both problems, as it seems to have moved away from the optimal policy in certain states, which makes sense because Q-learning does not have as much detailed information and domain knowledge as the first two algorithms, therefore it usually can miss out on the finer details. It seems that the Q- learning algorithm's optimal answer was very different for the easy grid world problem, but more similar for the hard grid world problem, although still different. Q-learning also took a noticeably longer runtime than the other algorithms, which was also expected because of the exploration-exploitation problem, which is that it does not simply compute the utilities like value and policy iteration does, but instead actually 'learns' them. With that being said, the total reward for the Q-learning for MDP 2 converged quicker and it actually got closer to the optimal answer in the hard problem with 63 states than in the easier problem with 15 states. This may be due to the fact that the hard grid world problem has a longer path and Q-learning is able to take advantage of this and avoid risks.



Figures 1, 2, 3 : Application of value iteration (left), policy iteration (middle) and Q- learning (right) on EasyGridWorld problem, MDP 1.



Figures 1, 2, 3 : Application of value iteration (left), policy iteration (middle) and Q- learning (right) on EasyGridWorld problem, MDP 2.

Conclusion:

After running the algorithms on the problems and visualizing their results, I have found out that Q – learning is not very good at finding solutions when a.) it does not get to explore the action space, and b.) when the number of states is small, as we saw that it did much better on the hard grid world problem, producing nearly the same result as the value iteration and policy iteration algorithms. Furthermore, I have found that Q-learning takes longer than both value and policy iteration in general. In fact, it takes exponential time when compared to the policy and value iteration, so if in the future I had to pick a method for usage in this situation (that is, a situation where I already have a lot of domain knowledge, I would definitely pick one of the iterative methods over Q-learning. This makes sense to what we learned because as you can see from its name, Q- learning is literally a learning algorithm, which means it would be good for using with a situation where you don't have much information so it can make the best guesses with its learning, whereas in that sort of situation, I am confident that value and policy iteration would not do as well. Therefore, which algorithm is optimal depends completely on the situation you apply it to. In the case of the world grid problems I chose, due to their nature, the value and policy iterations performed much better, and policy iteration performed better than value iteration due to how it can prematurely terminate as mentioned earlier in my analysis.