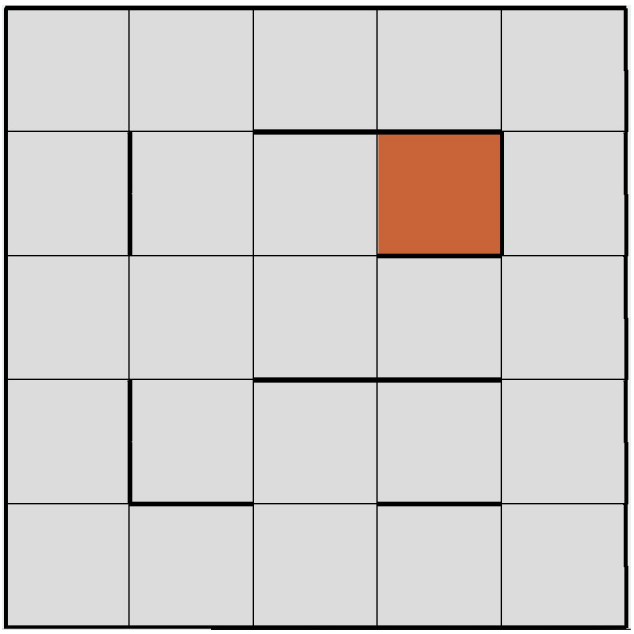
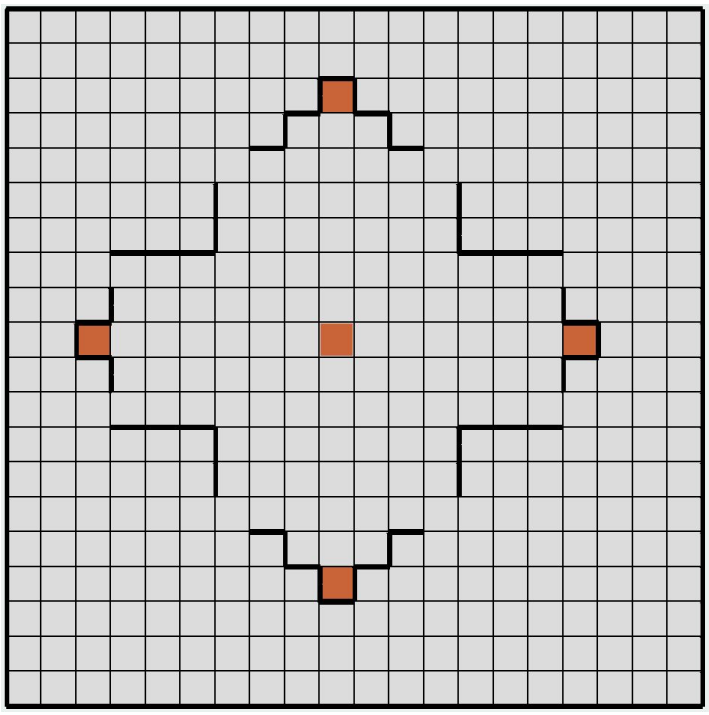
CS 4641 Project 4
Jonathan Hrach

This paper introduces two different Markov Decision Processes (MDP's), one gridworld problem with a small number of states and one gridworld problem with a large number of states. We then use value iteration, policy iteration, and Q-learning to solve the MDP's and compare the performance of each algorithm.

Suppose we have some fully observable environment where some agent must make a sequence of decisions to maximize some sort of reward. This agent can be in some finite set of sets and can take actions, the outcomes of which are probabilistic and lead to other states with associated additive rewards. These transitions from one state to another are Markovian in that only the present state matters in determining the next state. This sort of sequential decision problem is known as a Markov Decision Process. A solution specifies the exact action the agent should take given any particular state and is known as a policy. The best policy, which maximizes the long term expected reward, is known as the optimal policy [1]. Our MDP's will be gridworlds, which are just rectangular grids with an agent in a starting square that takes actions and moves to other squares, usually with the agent trying to get to some goal square.



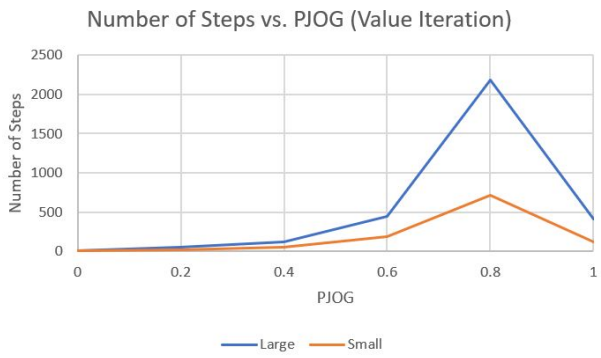
The first gridworld is a 5x5 grid with a single goal state and several walls which are more highlighted. The agent's objective is of course to reach the goal state in as few iterations as possible. The actions the agent has are the standard up, down, left, right. Additionally, the environment is of course stochastic, so there is some probability that an unexpected action occurs. The magnitude of this probability is something we will be experimenting with and seeing how it affects the performances of the different algorithms. The second gridworld consists of a much larger 20x20 grid with 5 goal states with the same set of possible actions and existing probability that some unexpected action occurs. If the agent collides with a wall for this gridworld, however, then it receives a penalty of -50.

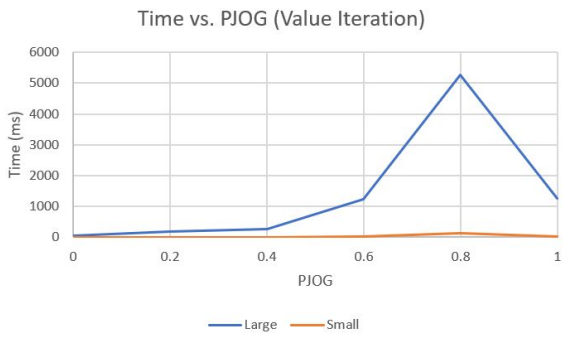


I thought these gridworlds would make for interesting MDP's because they can easily serve as models for real-life scenarios such as with robots. For example, a robot may have to navigate within some fixed amount of space with walls as obstacle to reach some goal, equipped with actuators and sensors that may not be 100% reliable. Also, it will be interesting to see the difference in the agent's actions between the two environments especially with the larger gridworld consisting of 16 times more states than that of the smaller gridworld. Note that for our gridworld experiments, we assume there are infinite horizons for decision making, which implies that our agent can technically live on forever. This means that an agent's preference to take an action in any given state won't really change, so the optimal policy is therefore stationary. Also note that all experiments were conducted with Carnegie Mellon's Reinforcement Learning Simulator [9].

Value Iteration

The first algorithm we will use to solve our two gridworlds is value iteration, which is an algorithm that starts with some arbitrary utilities for each state and performs Bellman updates by looking at utilities of all the other states it can reach, weighing those based on the transition model and then repeatedly taking the action that maximizes expected utility till convergence [2]. We run several trials on each gridworld with varying PJOG values. Note that PJOG indicates that the agent will take an action with probability (1-PJOG), meaning that with a PJOG of 0.1, there is a 90% chance the agent will make its intended action and a 10% chance it won't.

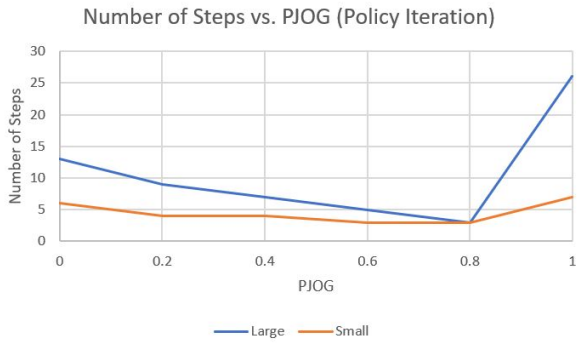




The left graph shows the number of steps taken by the algorithm to find the optimal policy. With lower PJOG values, the number of steps required to find an optimal policy is very low. This is because the environment becomes more deterministic and thus there are less suboptimal moves the algorithm must deal with. With PJOG values like 0.8, the algorithm is constantly making suboptimal moves so it takes a very large number of steps before converging and finding the optimal policy. The decrease in number of steps toward the end of the graph occurs when the agent's intended action will be executed less than 20% of the time. Likely with the 0.8 PJOG value, the agent is exploring too much and struggling to realize it's more likely to find the optimal policy by not executing an action that seems most optimal, while with a PJOG value of 1 it's having an easier time finding the optimal policy by understanding this rather odd methodology. As far as runtime goes, value iteration on the small gridworld ran extremely fast regardless of the PJOG value, which is hardly surprising considering the small amount of states and actions the algorithm must consider. Comparatively, the large gridworld had substantially greater runtimes for PJOG values of 0.6 and 0.8 than the small gridworld due to the sizeable amount of states and unreliable actions to deal with. Both graph's trend for the gridworlds seems very similar to the number of steps taken, which makes sense as the algorithm should run longer as more steps are taken.

Policy Iteration

While value iteration updated utility values to obtain the optimal policy, policy iteration emphasizes updating policies. Similar to value iteration, it starts with some initial arbitrary policy. Alternating between two steps, it first evaluates how good that policy is by calculating the utility of all the states, and then improves that policy at time t+1, which is the policy that takes actions that maximizes expected utility based on the current optimal policy [3]. It is also known to converge, as each step the policy gets closer and closer to the optimal one and there are only a finite number of policies for a finite MDP [4]. We now proceed to run several trials on each gridworld with varying PJOG values.



The above graphs show the number of steps and time, both as a function of PJOG. Immediately, we can see the large difference in the number of steps and runtime taken by policy iteration when compared to value iteration. This isn't surprising, as it is known that for less complicated MDP's, policy iteration is usually the most efficient approach [1]. As far as runtime goes, however, policy iteration can be very time consuming for larger state spaces. Each iteration is supposed to take $O(|S|^3)$ time, where |S| is the cardinality of the set of states. On the other hand, value iteration takes at most O(|S|*|A|) each iteration, where |A| is the cardinality of the set of possible actions and is usually small [5]. So, the gridworlds aren't actually complex enough for value iteration to converge faster. In addition, both value iteration and policy iteration find the same optimal policy, in which policy iteration finds the optimal policy directly through optimizing the current policy every iteration and value iteration finds the optimal policy indirectly through optimizing the utilities for each state every iteration. As far as the differences between the two gridworlds for policy iteration, there isn't a significant difference in the number of steps for reasonable PJOG values. In general, policy iteration is known to often converge in a small number of steps [4], especially with smaller state spaces. For runtime, it's no surprise that despite the number of steps being relatively low for both gridworlds, there are many more states for the algorithm to consider in the larger gridworld and therefore more work to do, which explains why runtime was higher for all PJOG values. Policy iteration has so far proven to perform better than value iteration for our two gridworlds, but now we proceed to apply a reinforcement algorithm, Q-Learning, to solve our model.

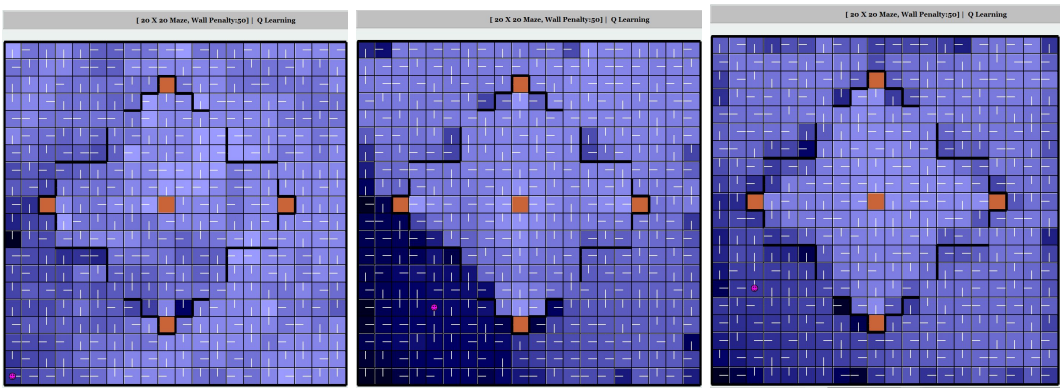
Q-Learning

Q-Learning is a reinforcement learning algorithm that doesn't need a transition model or a reward function during learning or action selection [1]. This is particularly useful, as there are many real-world problems in which the agent has to obtain information through interacting with its environment in order to discover the optimal policy [6]. It uses Q-values, which are related to utility values in order to estimate the total collected reward for (S,A) pairs, and uses the following update rule when some action a at time t is executed in some state s, leading to some new state s_{t+1} [1]:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

[7]

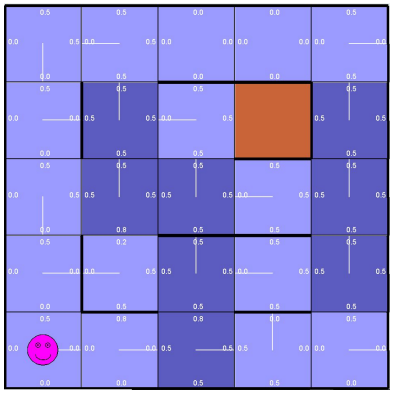
Our learning rate controls how much our updated Q-value moves in the direction of the immediate reward and helps control the amount of exploration and exploitation our agent will be doing, a fundamental tradeoff in reinforcement learning [8]. In addition, it must be decaying over time in order to guarantee convergence [5]. The discount factor controls how important future rewards are. We proceed to run several experiments, varying the learning rate to see which type of exploration strategies work best.



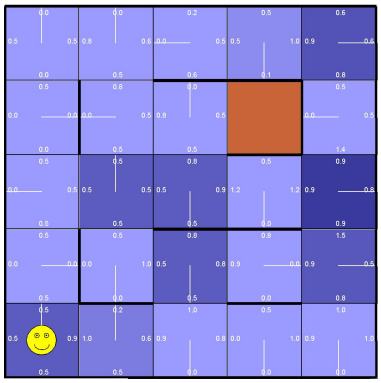
α = 0.9 α = 0.5 α = 0.1

For the larger gridworld, I ran several trials of Q-Learning with a PJOG of 0.2 and learning rates of 0.1, 0.5, and 0.9 with 10+ episodes. Comparing these to the optimal policy found from value and policy iteration for a PJOG of 0.2, I found that the learning rate of 0.5 performed the best, as the policy from above was the closest of all three to the actual optimal policy found from policy and value iteration. However, I wasn't very satisfied with the runtime of any of the trials. For the learning rates of 0.5 and 0.9, it took many hours before visiting the entire state space and generating a policy. This is because all three trials were run with an epsilon value of 0.1, which is another parameter that tunes how much the agent either explores or exploits, where epsilon values near 0 are more exploitative and epsilon values near 1 are explorative. With a low epsilon value, Q-learning will have a harder time finding optimal Q-values, as there is a low chance the agent will end up exploring many of the outer squares in the corner. After all, every state-action pair often has to be explored many times before Q-learning is able to find the optimal policy, so for the larger gridworld the agent should likely be exploring the state space a bit more in order to revisit states as much as possible. Through running additional trials, I found that using an epsilon value of 0.5 was ideal for the large gridworld. However, the policies found again still all seemed to be somewhat suboptimal, so they weren't interesting enough to include. In general, Q-learning didn't seem to work very well with the large gridworld, as it always took a long time for the agent to explore the entire state space and generate a policy, and the final

policies never matched the optimal policies found by that of value and policy iteration. However, the exploration strategy that worked the best was definitely a balance between exploring and exploiting with a learning rate of 0.5 and epsilon of 0.5. As for the smaller gridworld, I ran an episode with learning rates of 0.1, 0.5, and 0.9 with a PJOG of 0.2 and epsilon of 0.1.

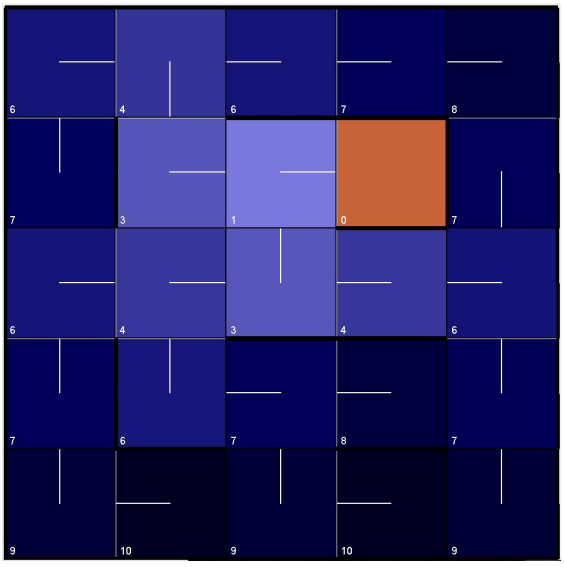


α = 0.5, ε = 0.1

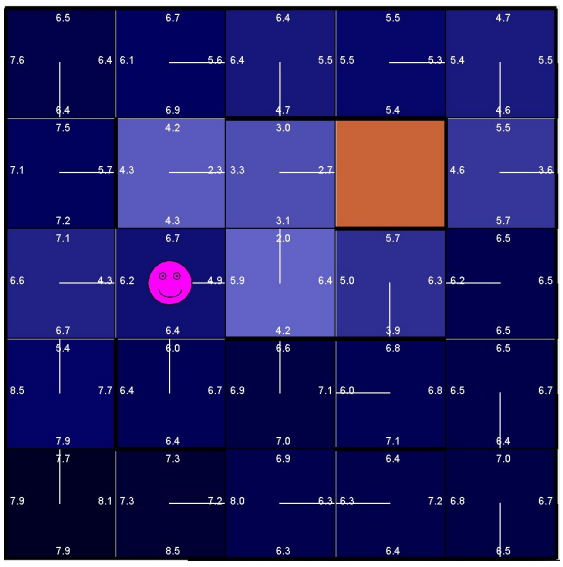


α = 0.5, ε = 0.5

Shown above are two runs of just one episode with a PJOG of 0.2. With a learning rate of 0.5 and epsilon value of 0.5, the state space was explored the most from all the trials I ran. All other episodes with various learning rates and epsilon values of 0.1 did not generate a complete policy and had pretty inconsistent runtimes (ranging from several seconds to several minutes), which can be attributed to the stochasticity of the environment. In addition, the policy recommended is very poor, as it doesn't recommend the logical action for most states, even ones close to the goal. From here, I ran a bunch more episodes with various combinations of learning rates and epsilon values with a PJOG of 0.2 to see which combination would yield the best policy.



Optimal Policy



α = 0.9, ε = 0.1

The left figure shows the optimal policy found by value and policy iteration, while the right shows the most ideal policy found from experimenting with various combinations. I found that a learning rate of 0.9 and an epsilon of 0.1 is most ideal. This combination likely performed best because in such a small state space, there isn't much need for much exploration, so lower epsilon values and learning rates that aren't too high work better. So, while a balance between exploration and exploitation worked best for the larger gridworld, more exploitation proved to be the best strategy on the smaller gridworld.

Wrap Up

Overall, Q-learning didn't perform as well as value and policy iteration in terms of finding the optimal policy. This wasn't too surprising, however, as value and policy iteration are equipped fully with domain knowledge of the environment whereas Q-learning has to slowly gain information about its environment experimentally. Value iteration and policy iteration were able to always find the optimal policies (which Q-learning struggled with especially) and run faster than Q-learning. The biggest issue with Q-learning is finding the correct parameters to control the amount of exploration and exploitation the agent is doing, which affect the algorithm's runtime. Also, too much exploitation can result in getting stuck at local optima, resulting in suboptimal policies. In addition, the stochasticity of the environment, which was controlled by the PJOG value, definitely affected the runtime. I ultimately decided to play around with learning rates and epsilon values with a PJOG value of 0.2, as trials with higher PJOG values often took too long to run. While Q-learning didn't perform as well as I wanted it to, it still has the advantage of being a model-free method and can definitely be applied to more real-life situations. However, when equipped with full domain knowledge, I would choose policy iteration over value iteration for both gridworlds, as it ran faster for all PJOG values.

Sources:

[1] https://dl.acm.org/citation.cfm?id=1671238 (Chapter 17)
[2] https://classroom.udacity.com/courses/ud262/lessons/684808907/concepts/6512308850923
[3] https://classroom.udacity.com/courses/ud262/lessons/684808907/concepts/6512308900923
[4] http://www.incompleteideas.net/book/ebook/node43.html
[5]https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec23.pdf
[6]http://read.pudn.com/downloads93/sourcecode/book/365930/RL_sim/reports/vivek_report.pdf
[7] https://en.wikipedia.org/wiki/Q-learning
[8] https://classroom.udacity.com/courses/ud262/lessons/643978935/concepts/6348990570923
[9] https://www.cs.cmu.edu/~awm/rlsim/