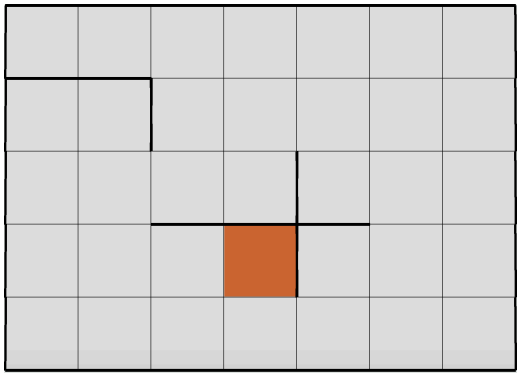
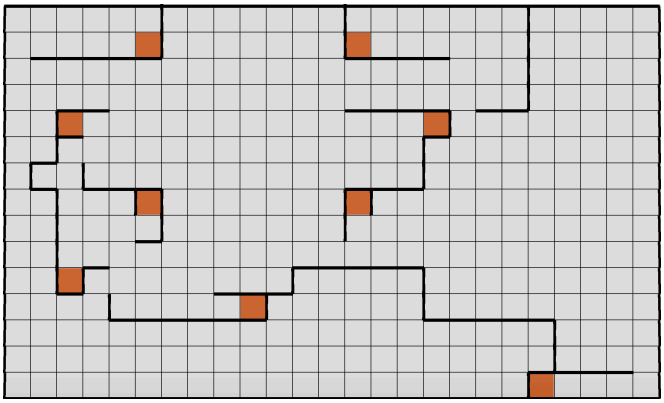
Sanjana Kadiveti
CS 4641

Analysis of MDPs and Reinforcement Learning

In this paper, I compare and contrast the performance of Policy Iteration, Value Iteration and Q-Learning on two different mazes: one relatively small, simple maze and one larger, more complex, multiple goal state maze. I discuss what parameters make each of the three algorithms perform better or worse, and which scenarios each of the three are better suited for over the others. This paper uses the CMU Reinforcement Learner Simulator for every experiment performed.

My MDPs





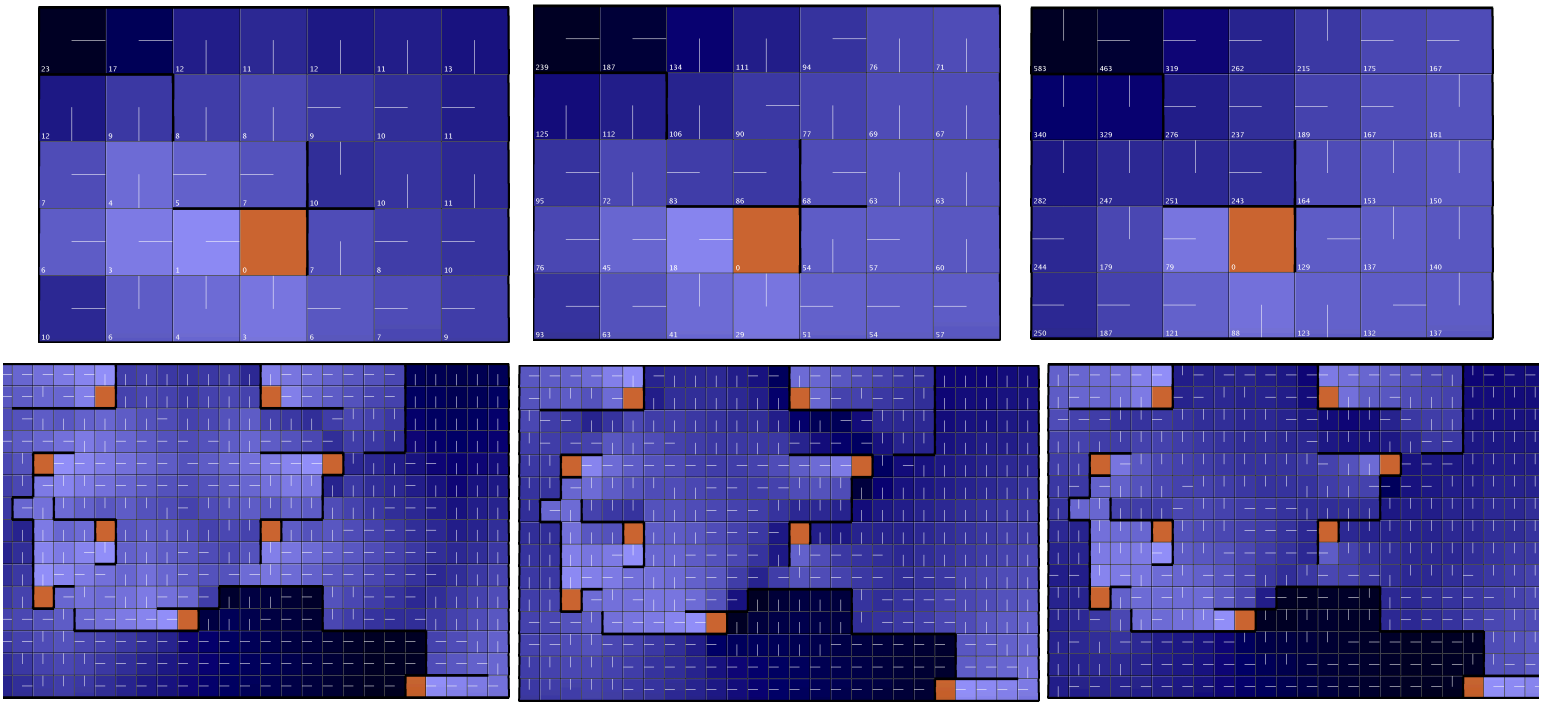
The above figures represent my two MDPs. Orange squares are goal states, black lines are walls or blocked pathways, and gray squares are all the regular states. The penalty for hitting a wall throughout this paper is 50. The MDP above to the left is the smaller one, consisting of just 35 total states (7x5 grid) with one goal state. With this maze, I thought it would be interesting to see how the algorithms perform with walls blocking the one goal state on half of its sides (faces), and whether on certain non-goal states, the agents would prefer the rather narrow pathway to the left face or the seemingly less blocked but further down bottom face.

The second maze is a 15x25 grid with 375 states and 9 goal states. With the larger maze, I wanted to see how the algorithms would perform with several goal states, and once again, how they would perform given that almost all of the goal states are blocked by walls on at least 2 sides, and are placed in the corners of narrow corridor-like structures. This means that the agent would have to traverse straight down a narrow path with a high likelihood of hitting walls in order for it to reach any of the goal states. I also made it so that some goal states are impossible to get to without entering the main central region, such as the bottom right goal.

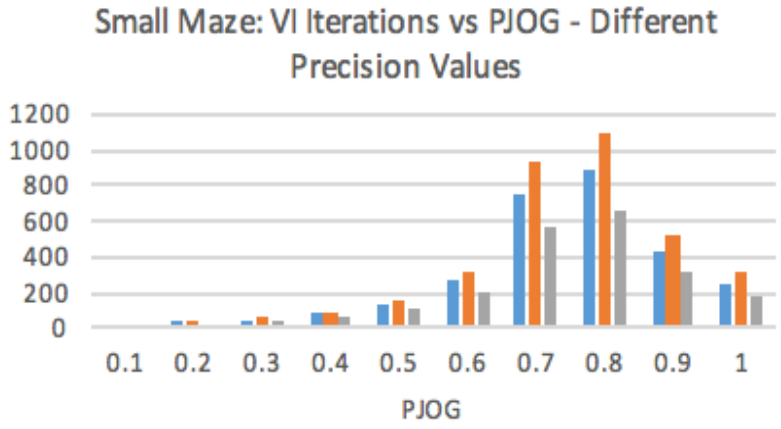
Value Iteration

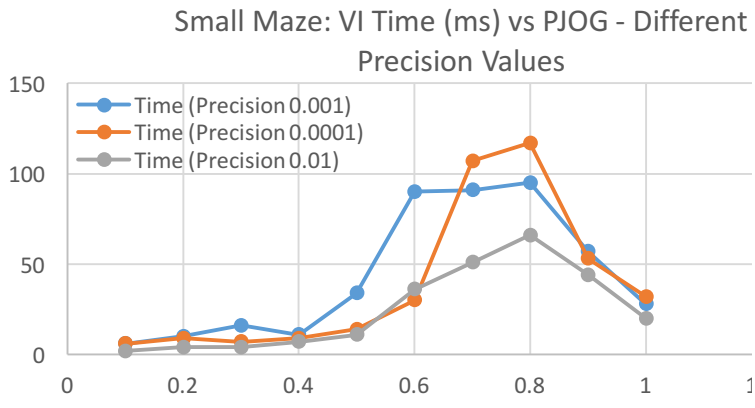
Value Iteration is an algorithm that calculates expected utility with the value function equation, and it does this for each state and its possible next state [1]. The algorithm selects the action at every step that maximizes this value, converging when these values for all states don't change. In RL-Sim, the parameter PJOG represents the probability of taking a desired action. (1-PJOG is the probability of making an unintended move). Essentially, PJOG makes the MDP environment

stochastic rather than deterministic. It makes the algorithms discover unexplored territory, which as we know from Randomized Optimization and Simulated Annealing, is usually a good idea and reduces bias.

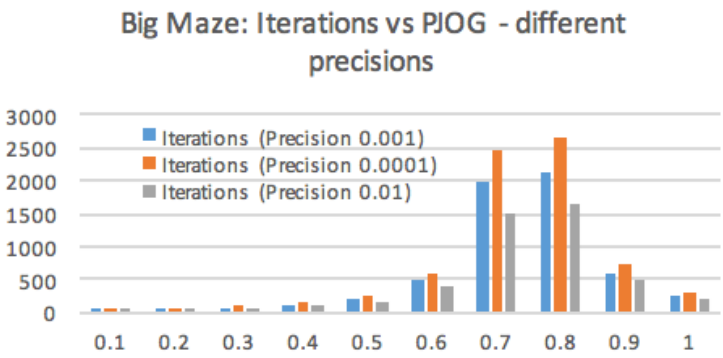


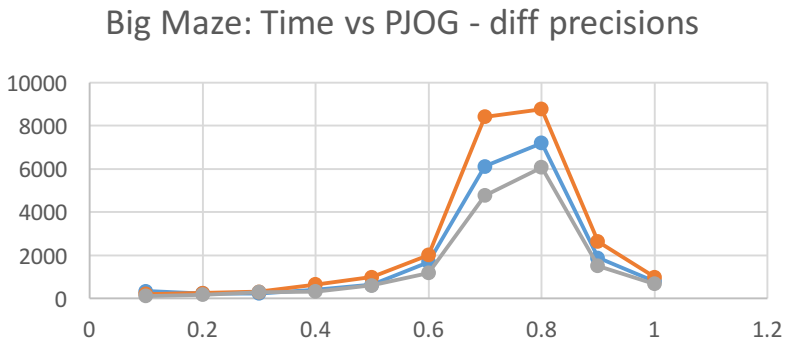
The above figures show converged states for both mazes with PJOG values 0.1, 0.5 and 0.9 respectively, from left to right. For both mazes, between 0.1 and 0.5 we don't see many differences in each state's actions, except more of the states nearer to the sides (on the right and left) and in the corners were explored more in the 0.5 outcome. In the 0.5 outcome of the small maze, the states on the right seem to prefer the bottom path to the goal over the left one. Overall however, the biggest change we see is with 0.9 PJOG. For both mazes, in the 0.9 convergence state, several states close to the goal states have flipped actions from 0.1 and 0.5's. This is expected, since we are very likely to not follow the desired path with PJOG 0.9 and therefore would succeed more by setting it to a different policy altogether.





Sanjana Kadiveti
CS 4641





Value iteration on the small MDP with PJOG 0.1 converged in just 23 steps and 18ms. Upon repeated runs, I found that this time unit remains in a certain bracket, between 6ms to 23ms, and obtaining different time values is likely due to probabilistic nature of the environment; we know the probability of unexpected action, but don't know exactly how often unexpected actions will occur. The more it occurs, the more time spent.

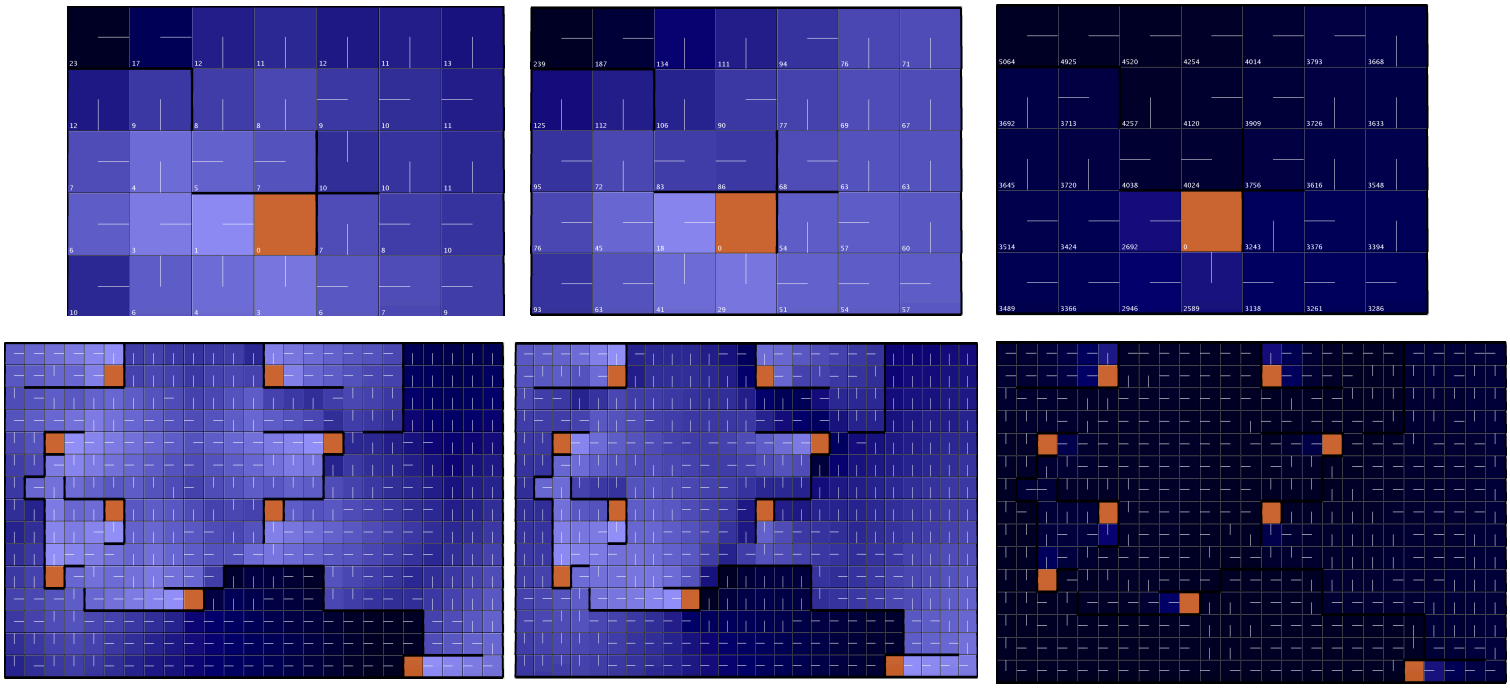
The above graphs show the variation in number of iterations (steps) and time taken to converge as PJOG, probability of taking an unexpected action, increases. They show this for 3 different values of precision, which is what the algorithm uses to determine whether it has reached a state of convergence. We notice that, for both mazes, as PJOG increases, there is a general increase in number of iterations, but past 0.8 we see them decrease. This is likely due to the fact that the more often we take unwanted paths, the more steps the algorithm requires to get back on track and arrive at the goal state. Past a 0.8 probability of taking undesired actions, we see a decrease in both time and iterations needed, and I hypothesize that this is due to the fact that the earlier we hit a wall and obtain a penalty of 50 points, the quicker we discover the optimal path and goal state. This is in contrast to the probabilities less than 0.8, when it is likely that the agent traversed a large number of states before colliding with a wall, which is a more time consuming series of steps.

Between precision values, we notice that the lower the precision, the greater the number of iterations and time needed, and this is the case with both mazes. This is likely because decided the values have converged becomes trickier with lower precisions, and so value iteration continues for longer. The only exception we notice is when 0.0001 precision takes longer than 0.01 only past a value of 0.6 for PJOG. This anomaly could have been caused because with lower PJOG values, uncertainty decreases and the values near convergence are not as complicatedly similar. Between the big and the small maze, we notice that the larger maze takes a lot longer than the small one to arrive at convergence. In most cases the increase in time is more significant than the increase in iterations, besides the peak states of PJOG values 0.7 and 0.8. This is expected however, because an iteration on the big maze takes longer than an iteration on the small maze. Interestingly, PJOG values 0.7 and 0.8 are the two highest points for iterations and time in both mazes. Perhaps this says something inherent about value iteration and its seemingly poorer performance at these two uncertainty ratios.

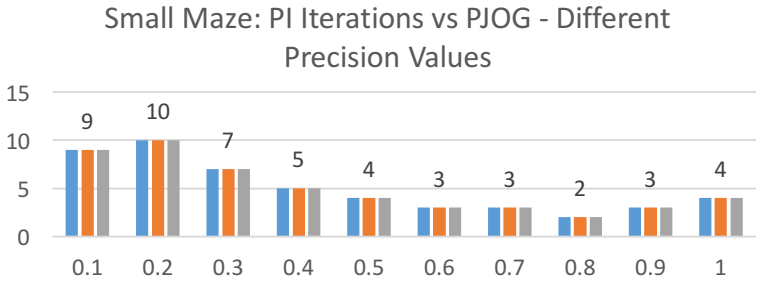
Sanjana Kadiveti
CS 4641

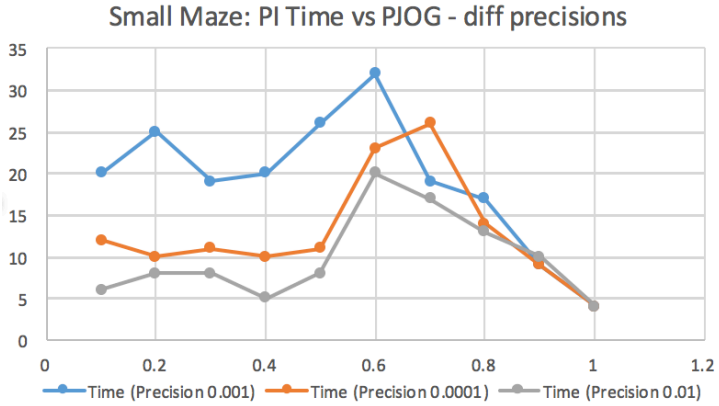
Policy Iteration

Policy Iteration's algorithm works by computing a policy instead of value function estimates at every state, and using this policy, finds the current utility value for each state [1]. Policy Iteration converges when the policy stops changing, rather than the values themselves.

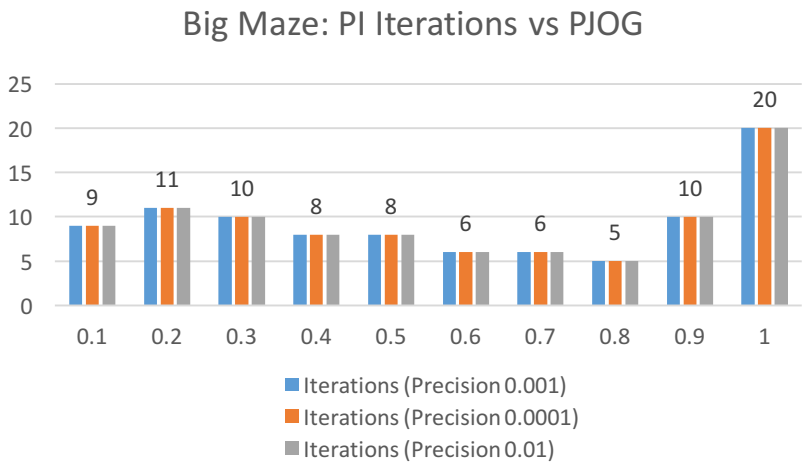


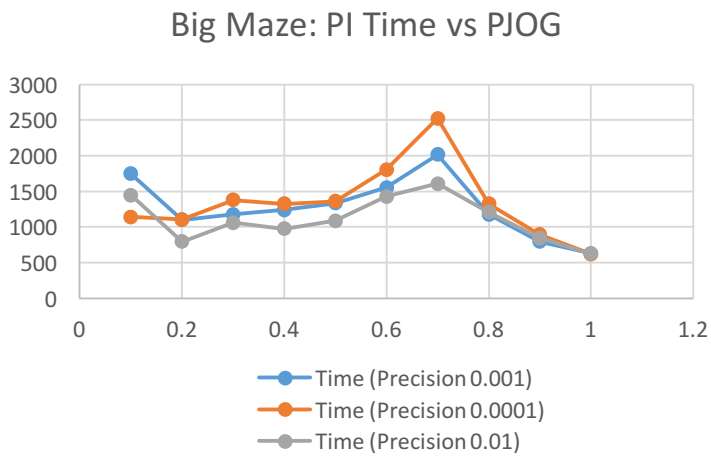
The above figures show the mazes converged with PJOG values 0.1, 0.5, and 0.9 respectively from left to right. We notice that there are minor differences between the 0.1 maze and the 0.5 maze, and in particular the 0.5 maze utilizes the bottom entrance to the goal state more than the 0.1 maze does. This was the case with value iteration 0.5 PJOG as well, but we see even more states preferring the bottom path here with policy iteration. We also notice that the 0.9 PJOG maze very different from the other two. Once again this is expected because the MDP of 0.9 PJOG takes into consideration that the unexpected action is much more likely to occur, and so it works best if the policy were not the optimal action. Interestingly however, the actions taken at states near the goal states are less different between 0.9 and 0.1/0.5 PJOG; we see more differences in the states far away from the goal states, which is unlike what we





Sanjana Kadiveti
CS 4641





saw with Value Iteration. Perhaps this means Policy iteration is more sturdy and resistive to non-determinism than Value Iteration.

The plots above represent the mazes' changes in iterations and time over changing PJOG values for three different precision values. We immediately notice that precision doesn't impact number of iterations for policy iteration. I hypothesize that this is because the algorithm attempts to find one optimal policy, which would not be impacted by precision, unless the MDP is truly extremely complex with several more pathways and actions to select from. Which is not the case here. One could argue that there are differences in convergence times between precision values, especially because, with the small maze, we see that until a PJOG value of 0.6, precision 0.01 convergence times are higher than the other two. Interestingly however, all three precision results have the same convergence time at PJOG 1 for both mazes. In addition, the difference between the times for any of the PJOGs isn't very significant, and this difference could be attributed to the probabilistic nature of PJOG. This is reinforced by the fact that there doesn't seem to be a trend going from high precision to low precision in terms of time taken.

We see similar trends in the number of iterations needed for each maze. With the small one, they start off high, with 9 and 10 iterations for 0.1 and 0.2 PJOG values, and decrease to 2 at 0.8, past which we see small increases again. This could potentially mean that the nature of the policy iteration algorithm makes exploring undesired states useful, to a certain degree, past which the exploration becomes counter-productive. With the larger maze, we see similar starting values for iterations at around 9 to 11, a decrease to 5 at 0.8 PJOG, and much sharper increases in iterations for 0.9 and 1 PJOG. This reinforces my argument that there is a happy middle between exploitation and exploration with policy iteration, and it is important to maintain it in and around it, because too much exploration can be very detrimental, particularly with more complex MDPs.

One particularly interesting aspect to note is that, with fewer iterations, we notice an increase in time taken to converge. This is an inherent property of policy iteration as well. It generally requires less iterations to reach convergence, but is computationally more expensive.

Sanjana Kadiveti
CS 4641

Comparing Value Iteration and Policy Iteration
Small Maze, Precision 0.01

PJOG	Value Iteration		Policy Iteration	
	Time	Iterations	Time	Iterations
0.1	18	23	21	9
0.5	23	137	26	4
0.9	11	427	9	3

Big Maze, Precision 0.01

PJOG	Value Iteration		Policy Iteration	
	Time	Iterations	Time	Iterations
0.1	332	42	1746	9
0.5	639	215	1336	8
0.9	1876	614	794	20

Both policy iteration and value iteration are run with the assumption that the agent knows the nature of the environment, such as the reward function, the transition function, etc. The difference between the two lies in how they search for the optimal state. Policy iteration iterates until the policy itself converges, whereas value iteration iterates until a value function converges, from which it extracts an optimal policy. Usually, value functions take longer to converge than policies; with value iteration, the policy itself may have already converged but value iteration doesn't recognize it as such, because it waits until the value function converges.

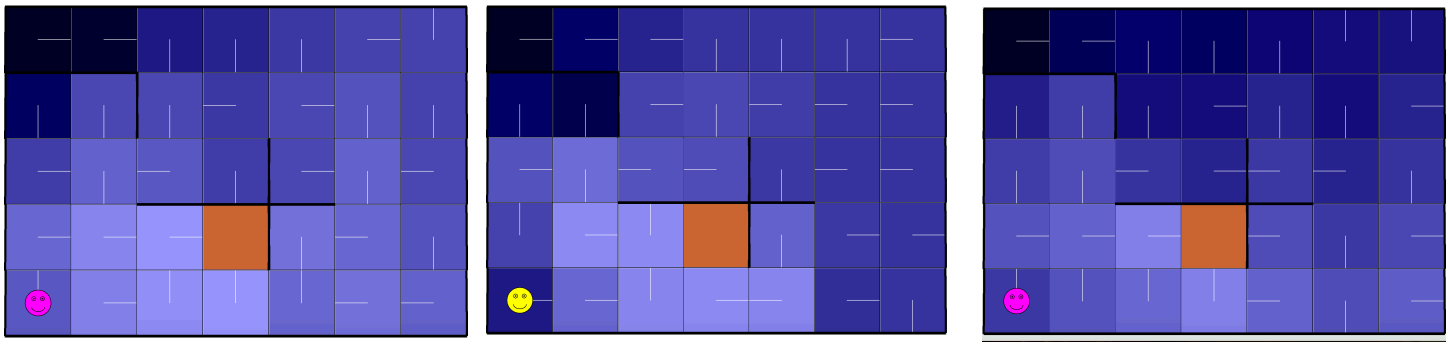
From the tables above, we can see that for a PJOG value of 0.1 and 0.5, Value iteration takes less time to converge, but with 0.9, it takes longer than policy iteration. This is likely caused by the fact that, when the agent is more likely to perform the desired action, the value function involved can converge and be evaluated faster, and therefore value iteration works fast in this case. With a PJOG value of 0.9, the agent performs unexpected actions and the value may continue alternating between possible states of slightly different value, even though the policy itself has converged. This is why policy iteration takes less time in this case. In terms of number of iterations, we consistently see that value iteration takes a larger number of them to converge. We also saw that there is a drastic increase in iterations for value iteration at around 0.7 / 0.8 for PJOG. This is likely due to the same reason as mentioned above; it reaches a certain point at which there are several similar states to explore, and the variety of them makes it difficult for utility to converge.

In terms of their outcomes, we saw that they returned very similar paths for PJOG values 0.1 and 0.5, but with 0.9, Value Iteration's states near goals were more sub-optimal, usually pointing away from the goal or towards a wall. Policy Iteration states behaved as such as well,

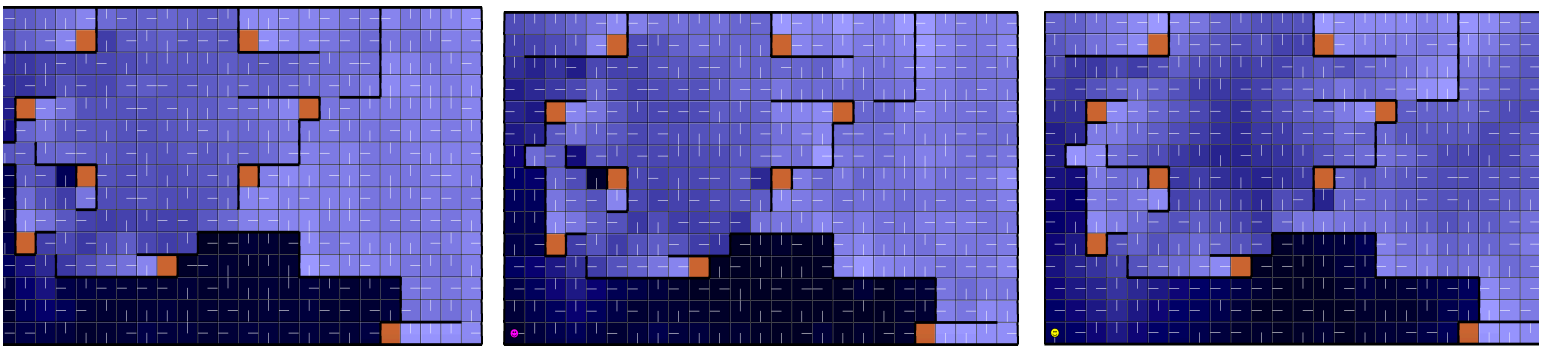
but more so in states far from the goal. This, as I mentioned previously, could mean that Policy Iteration's algorithm is better at sticking to a policy in high reward regions and perhaps more resistive to uncertainty.

Q-Learning

Q Learning is an algorithm that uses Q, a 2-D table of immediate and discounted reward sums for each state and action [3]. For each state, it simply chooses the action with the largest value in the table for that state, and consequently updates the table's values. It does this repeatedly until convergence to the optimal Q table. Unlike policy iteration and value iteration, Q learning is model-free, which means it does not know the rewards and states beforehand; instead it learns it the model through iterative exploration and exploitation. In the RL-Sim UI, there is a value called epsilon, which represents the probability of taking the optimal action in each state. I alter this variable to change exploration strategy. The higher the epsilon value, the less explorative the algorithm becomes.



The images above show convergence for Q learning with epsilon values 0.1, 0.5 and 0.9 respectively, from left to right. My unchanged parameters were a learning rate of 0.7 with decaying, 1000 episodes, PJOG of 0.3 and precision 0.001. With an epsilon value of 0.1, the states on the left half lead to the goal state, but states on the right half have action that are not as optimal. In particular, the third state from the right on the bottom row has an action pointing upwards, which it should point left. This state is required for any of the states above or to the right of it to reach the goal state through the bottom row. With an epsilon value of 0.5, we see that states close to the goal, particularly the states to the left of and below the goal, point to suboptimal directions. States far from the goal however have mostly optimal actions. With an epsilon of 0.8, we see cases in which states' actions are to hit walls, but generally, the states



close to the goal state point optimally towards the goal. For these reasons, I'd say overall 0.1 epsilon performed the best on the small maze.

The images above show convergence for Q learning with epsilon values 0.1, 0.5 and 0.9 respectively, from left to right on the large maze. My unchanged parameters were a learning rate of 0.7 with decaying, 1000 episodes, PJOG of 0.3 and precision 0.001. One thing to note is that 1000 episodes on the large maze took quite long to converge, especially with higher epsilon values. With 0.1 epsilon and less exploration, we see that with some goals, the paths at their neighbors point away from the goal state; this represents those goal states not being explored well enough. With 0.5 epsilon, we see slight improvement where some goals are explored better, such as the bottom right goal. Generally, however, the optimality of these actions does not improve by much, even with this additional exploration. The outcome of 0.9 epsilon doesn't fix the problem, even with 1000 episodes, and I would argue that there are more cases of states pointing to walls than in 0.1. I hypothesize that this means I needed to have run the algorithm for more even more episodes. Generally, I would say 0.5 epsilon worked best on the large maze of all three values.

Concluding Comparison

Q-learning did not perform well on the big maze; it took a very long time, and even with 1000 episodes and all three epsilon values, there were many seemingly suboptimal actions. On the small maze however, it does an impressive job, especially considering it's a model-free algorithm. Therefore, number of states plays a role in its performance. Value Iteration and Policy Iteration work effectively on both mazes. Neither take a long time, but policy iteration takes less time with higher uncertainty (PJOG between 0.7 and 0.9). The downside to these is that both of them function on the assumption that we know the environment and its reward and transition functions, which in practice is hardly ever true. As blind as the Q-Learning agent seems, and as long as Q-Learning takes to complete, with increased exploration and more episodes, Q-learning will find different paths to the goal, despite taking several non-optimal actions along the way, which is exactly how a blind agent **is expected** to behave. Without the time it needs with episodes and the exploration factor however, Q-learning will not discover paths as quickly as Value iteration and Policy iteration, and will only find sub-optimal paths.

So overall, in a perfect world, with not much uncertainty (more control over actions), Value Iteration is recommended. In a perfect world where there is uncertainty, policy iteration is recommended, but with some amount of exploitation and not just exploration. In a world where we don't know the state space, Q-learning will have to be used, but again there will have to be an intelligent combination of both exploration and exploitation.

Sanjana Kadiveti
CS 4641

References

[1] Medium. (2019). *Deep Reinforcement Learning Demysitifed (Episode 2) — Policy Iteration, Value Iteration and…*. [online] Available at: https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa.

[2] *Mitchell, T.M. (1997). Machine Learning (1st ed., McGraw-Hill Series in Computer Science). McGraw-Hill Education.*

[3] "*Machine Learning for Trading*" *Udacity, n.d. Web.*

[4] "*Intro to Machine Learning*" *Udacity, n.d. Web.*