Harsh Patel

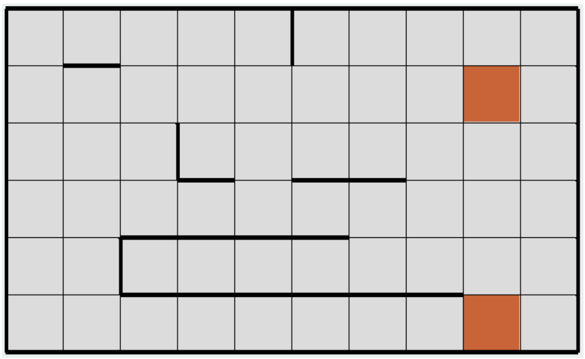
CS 4641 – Machine Learning

Assignment 4: Reinforcement Learning

Introduction + Selected MDPs

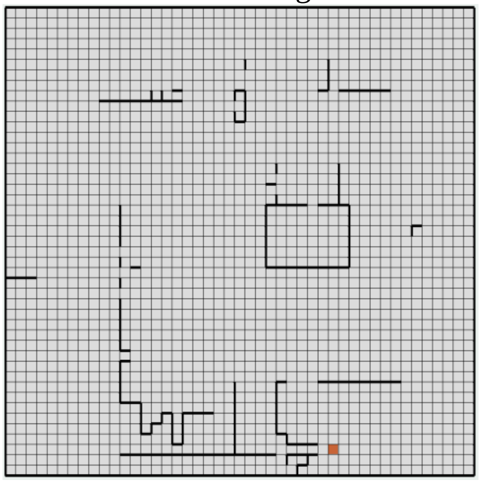
Markov decision processes (MDPs) provide a mathematical and computational framework for decision making in scenarios where decision outcomes are somewhat random, but also somewhat under the control of the agent. MDPs are usually solved using algorithms often classified under Reinforcement Learning, specifically Value Iteration, Policy Iteration, and Q-Learning. To both dive deeper into solving MDPs and the algorithms listed above, I selected the following two MDPs (both gridworld problems) and used RL_Sim (a reinforcement learning software) to explore the impact of different hyperparameters and also if different algorithms would converge to the same solution if the MDP was the same:

1. multipleGoals1.maze:



The gridworld problem to the left is an MDP with two goal states (indicated in orange) and quite a few obstacles / walls (indicated in bold square edges) that could possibly prevent an agent from reaching one of the goal states. This MDP is interesting because, quite simply, having more than one goal state increases the overall complexity of the problem (think search trees for possible moves knowing that there are two winning outcomes, that tree grows in size very, very quickly). The agent must now consider which goal state is optimal to strive for and how to reach that goal state while avoiding the walls. Furthermore, in states that are relatively close to both goal states, the agent must select actions that minimize overall penalty with respect to at least one of the two goal states.

2. big.maze:



The second MDP I selected is shown to the right. Unlike multileGoals1.maze, big.maze only has one goal state (indicated in orange; bottom right). What is glaringly different about this gridworld problem is the sheer number of state spaces. With 2025 (45x45 grid) possible states, the agent must navigate through this massive world in order to arrive at the goal state, all while minimizing penalties received (from either bumping into walls or staying alive too long). These massive problems are very useful to explore because they more or less help model real world MDPs where there are lots of possible states and our agent must be able to successfully select the correct series of actions to achieve its task.

Note: Each gridworld in RL_Sim, by default, has the same penalty for hitting a wall (50) and the "reward" for transitioning from one state to the next is a penalty of 1. The agent is trying to solve the MDP with the lowest score possible representing the lowest penalty one can earn if acting optimally.

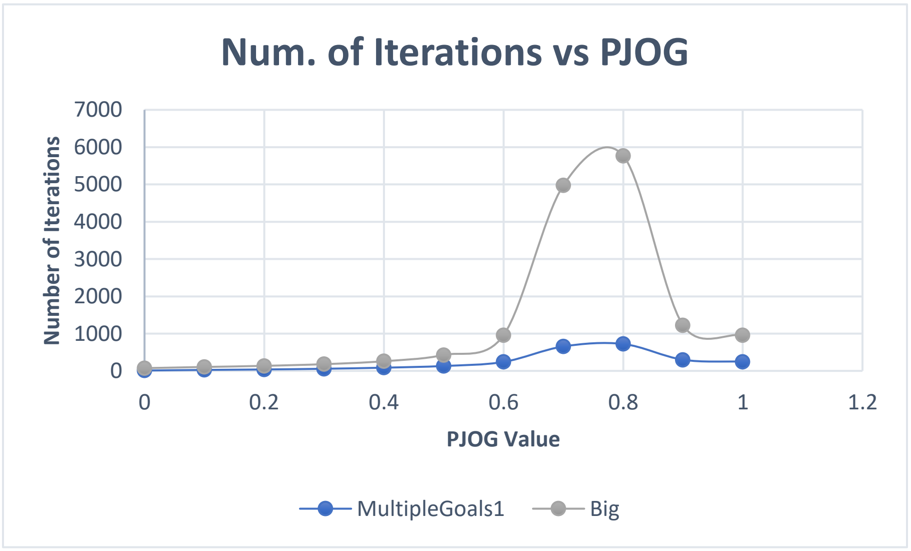
Value Iteration

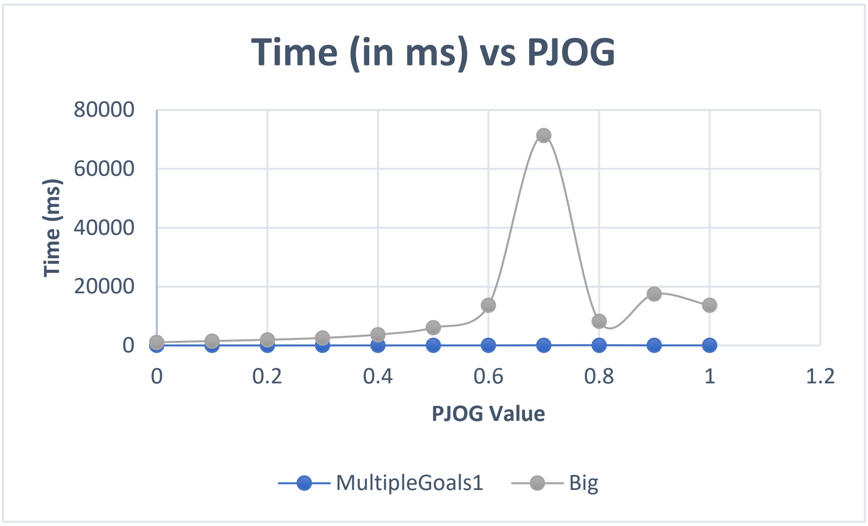
Value Iteration is a reinforcement learning algorithm that strives to maximize the value of a given state through repeated measurement of gained rewards with respect to an agent's actions. The pseudocode for RL_Sim's value iteration algorithm is as follows:

```
initialise $V(s)$ arbitararily
loop until policy good enough
    loop for $s \in S$
        $V_{t+1}(s) = \min_{a \in A}\{1 + \sum_{s'=S}(P(s'|s,a) \cdot x_{ss'})\}$
        $\pi_{t+1}(s) = \arg\min_{a \in A}\{\sum_{s'=S}(P(s'|s,a) \cdot x_{ss'})\}$
        where,
            $P(s'|s,a) =$ Prob. of transition from $s$ to $s'$ after action $a$.

            $x_{ss'} = V_t(s')$, if transtion from $s$ to $s'$ is safe
                $= Penalty + V_t(s)$, if transtion from $s$ to $s'$ is not safe

    end loop
end loop
```

In RL_Sim, as the value of a state is actually the total penalty an agent would get if starting at that state and acting optimally till the goal state, RL_Sim's value iteration algorithm aims to select an action that minimizes penalty incurred. To explore the different aspects of the algorithm with respect to the two MDPs discussed in the previous section, I loaded both gridworlds into RL_Sim and manipulated the following hyperparameters: PJOG (transition probability: 1 – PJOG value is the probability that the action taken by the agent is the action the agent wanted to take, where 1 – (1 – PJOG) or simply PJOG represents the probability of the agent taking a random action in an undesired direction; i.e. PJOB of 0.3 and selection action North → agent will go North 70% of the time, and either South, East, or West 30% of the time), and Precision (converge parameter that measures difference of the old value of the state and the newly measured value in order to determine when the algorithm should stop as the values have converged).

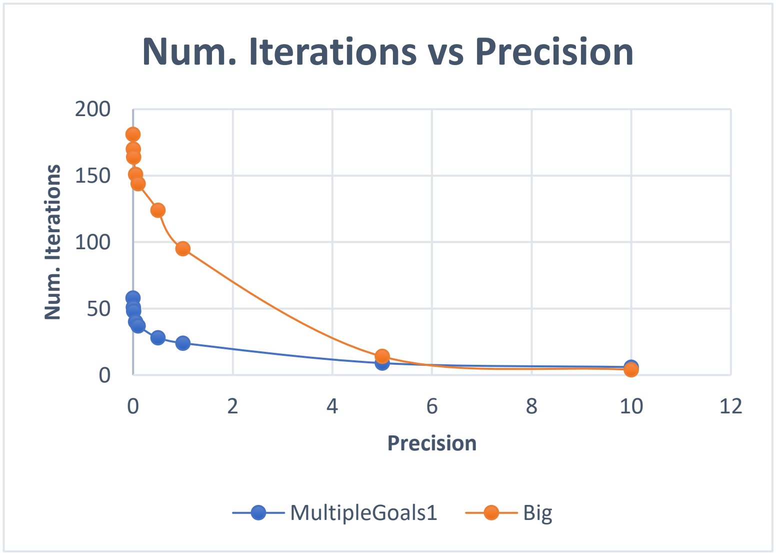
Looking first at changing PJOG (from 0 to 1 in increments of 0.1), while keeping precision at 0.001) for each MDP, we arrive at the following curves:

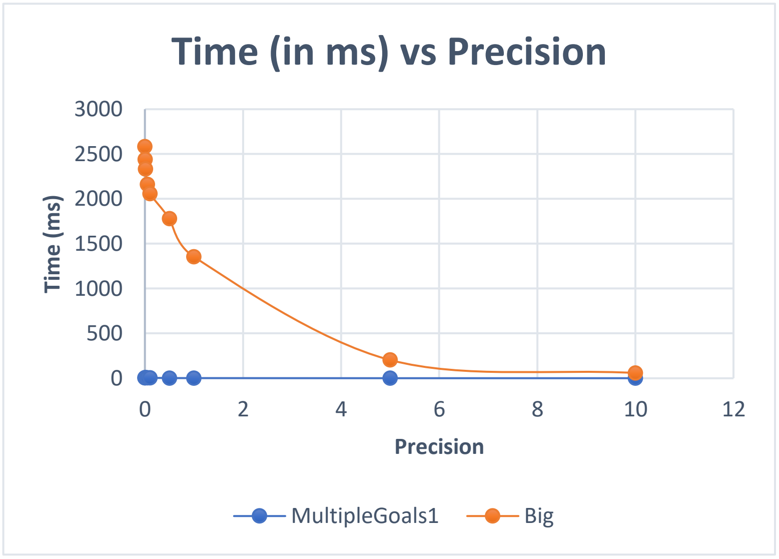




Looking first at the number of iterations versus PJOG chart, we see a similar trend for both MDPs: a general increase in the number of iterations as the PJOG value increases (with a peak around 0.8) and then a sudden drop around PJOG values of 0.9 and 1. Between PJOG values of 0 and 0.8, we can explain the rise in the number of iterations Value Iteration requires till converge due to the increase in randomness of the agent's actions. Even if the agent knows the optimal path to the goal state, as its actions are partially random, the higher the PJOG value, the more likely it is for the agent to take an incorrect action. From PJOG value of 0.9 onward, the agent is more or less making suboptimal actions every time it decides to move and as a result, the correct path is never followed. In order to minimize total error, the agent's policy converges quickly to incorrect actions as then the agent can randomly select the correct action with a probability equal to the PJOG value. Comparing the two MDPs, we see a pretty straightforward trend, in both the Num. Iterations vs. PJOG chart and the Time (in ms) vs. PJOG chart, where it takes longer and requires more iterations to solve big.maze than multipleGoals1.maze. This observation is more or less expected due to the larger state space of big.maze. The more states the algorithm must consider, the more time the algorithm will take to explore all the states and the more iterations required till convergence.

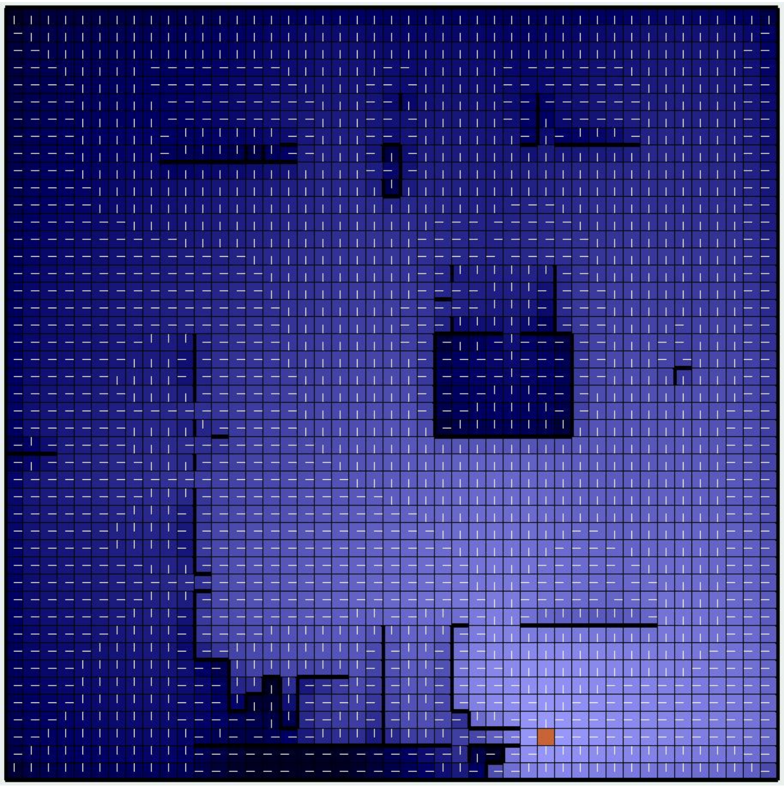
Moving on the how precision influences number of iterations and time taken till convergence, I changed the value of precision (0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10), while keeping PJOG at 0.3, and that resulted in the following curves:

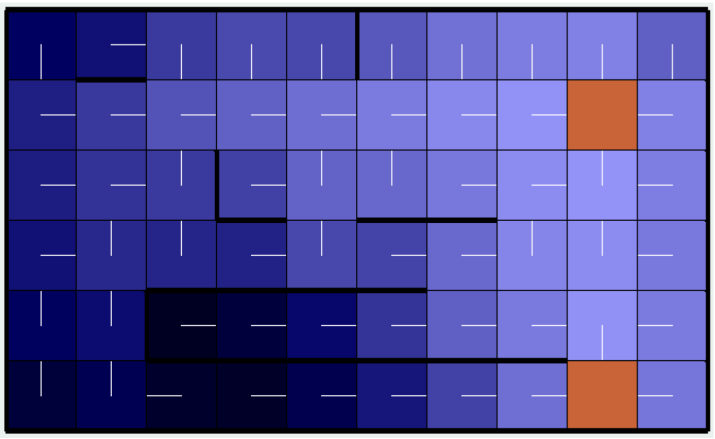




From the curves above, we see a similar trend across both the number of iterations and time with respect to precision: There is an overall decrease in num. iterations and time as the precision value increases for both the multipleGoals1.maze and big.maze. The big.maze, in both curves, has a higher value than the multipleGoals1.maze for low precision values, but seems to converge to values similar to that of multipleGoals1.maze for high precision values. Logically, the lower the precision value, the smaller the difference required between the old value and the new value of a state for the algorithm to decide that converge has occurred. Larger precision values allow for more leniency and thus the algorithm converges with less iterations and less time. Big.maze generally requires more iterations and time due to the large number of states, but what is really interesting is how the curves for big.maze converge to the curves for multipleGoals1.maze at high precision values. This tells us that having high precision values results in faster convergence regardless of the RL_Sim gridworld MDP selected. We can attempt to explain this phenomenon by noting the similarities in how the value of a state is determined between both MDPs, as both gridworld problems aim to minimize total penalty incurred till goal state with a penalty of 1 per transition and penalty of 50 every time the agent hits a wall.

Lastly, let's discuss the optimal policy Value Iteration converged to for each of the MDPs. Keeping PJOG at 0.3 and Precision at 0.001, RL_Sim's value iteration resulted in the following policy for each state of the two MDPs:

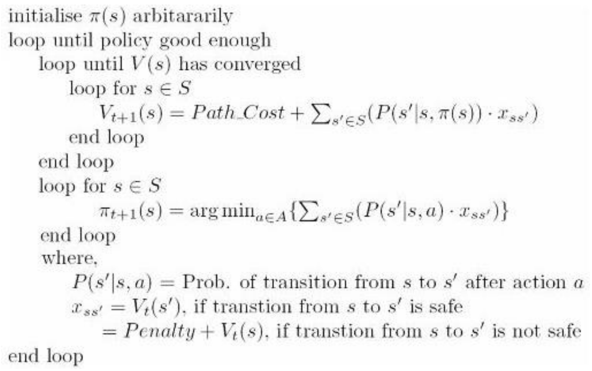




Although hard to see for the big.maze, we can easily see how states near the goal state have greater value (smaller penalties; indicated by light blue shading compared to dark blue shading) and the policy is generally the action that directly leads to the goal state. We will discuss the above policies in greater detail when we compare with the policies generated by Policy Iteration in the next section of the report.

Policy Iteration

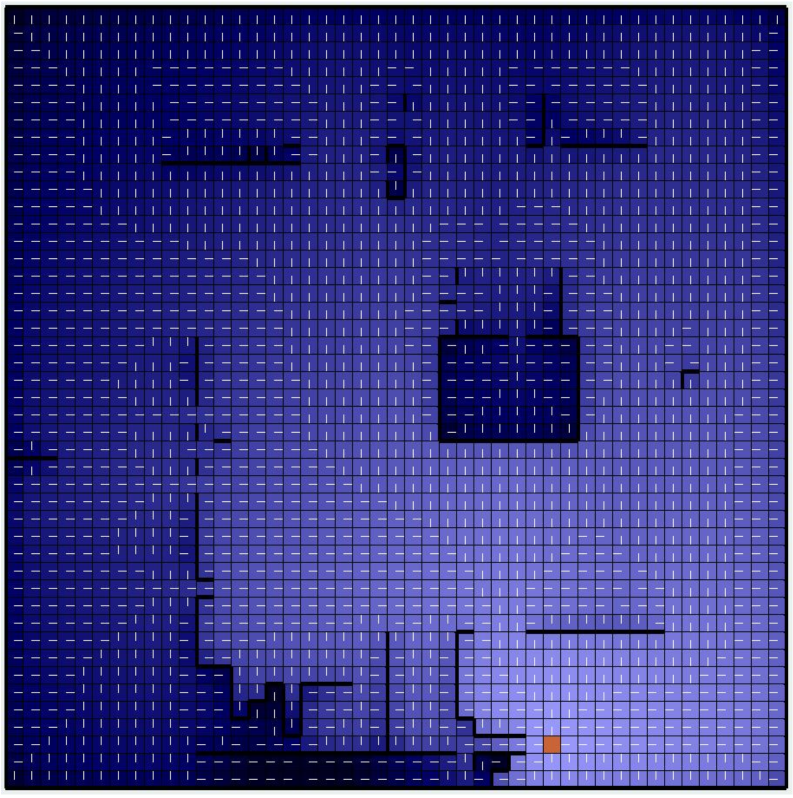
Policy Iteration is a reinforcement learning algorithm that strives to converge to the optimal policy by first initializing a random policy and then, through repeated measurement of gained rewards with respect to an agent's actions, updates that policy to reflect maximized state values (or in RL_Sim's case: minimized state values). The pseudocode for RL_Sim's policy iteration algorithm is as follows:

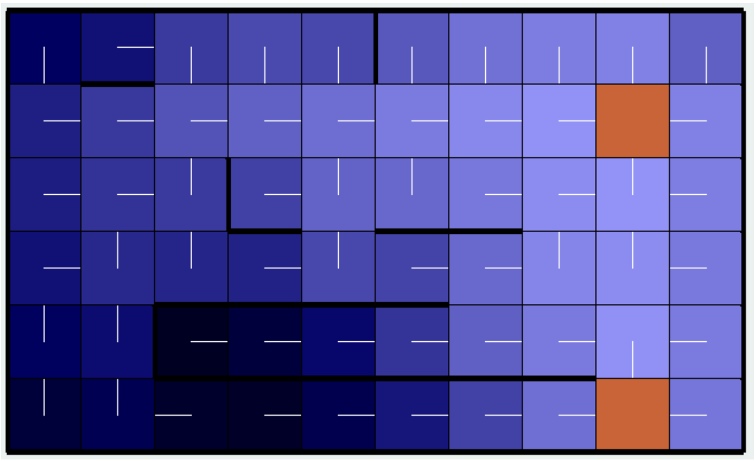


```
initialise $\pi(s)$ arbitararily
loop until policy good enough
    loop until $V(s)$ has converged
        loop for $s \in S$
            $V_{t+1}(s) = Path\_Cost + \sum_{s' \in S}(P(s'|s, \pi(s)) \cdot x_{ss'})$
        end loop
    end loop
    loop for $s \in S$
            $\pi_{t+1}(s) = \arg\min_{a \in A}\{\sum_{s' \in S}(P(s'|s, a) \cdot x_{ss'})\}$
    end loop
    where,
        $P(s'|s, a) =$ Prob. of transition from $s$ to $s'$ after action $a$
        $x_{ss'} = V_t(s')$, if transtion from $s$ to $s'$ is safe
            $= Penalty + V_t(s)$, if transtion from $s$ to $s'$ is not safe
end loop
```

There are several hyperparameters we can tune with respect to policy iteration. They include: Value Limit (the limit placed on how large the value of a particular state can reach; useful to prevent infinite loops), Iteration Limit (how many times each run of policy iteration should evaluate and update the individual value of each state), PJOG, and Precision (both discussed in the previous section).

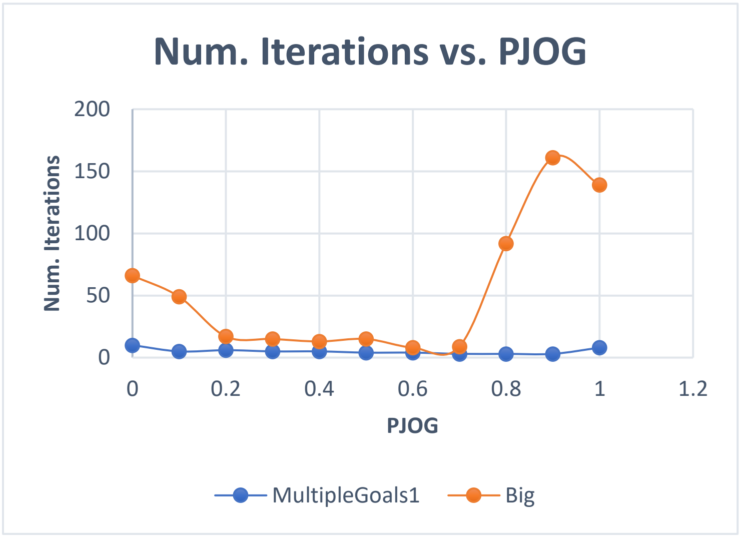
Before we jump into changing those hyperparameters, let us simply look at the optimal policy given by Policy Iteration for both of the MDPs and compare with the optimal policy given by Value Iteration (as that is still fresh in our minds and to avoid unnecessary scrolling up and down to compare figures). Optimal policies given by Policy Iteration (with PJOG = 0.3, Precision = 0.001, Value Limit = 5000, Iteration Limit = 500):



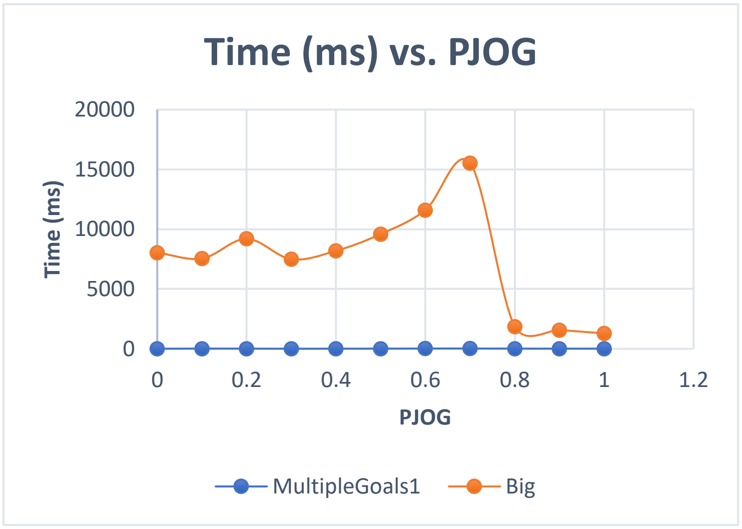


Comparing the optimal policy for multipleGoals1.maze returned by Value Iteration and Policy Iteration, we see that both algorithms have returned the exact same policy. This only solidifies the optimal policy and also enables us to select either algorithm to run with respect to that particular MDP. Looking at the policies for big.maze, we see a similar phenomenon where both Value Iteration and Policy Iteration have both given the same (mostly similar) policy for the gridworld problem (Note: While it was easy to check if the policies were the same for multipleGoals1.maze, I only verified policy similarities for key parts for big.maze (near the goal state, inside the wall box [center right], and near the edges) and subsequently concluded that the policies were similar enough). As expected, states near walls have darker shading indicating higher average penalty values even if the agent were to act optimally from then on to the goal state. Since policy iteration and value iteration resulted in the same (highly similar) policies for both MDPs, we can tune the hyperparameters of Policy Iteration to see if we can reduce number of iterations or time till convergence will still maintaining the optimal policy, thus resulting in a highly tuned and efficient algorithm for these specific MDPs.

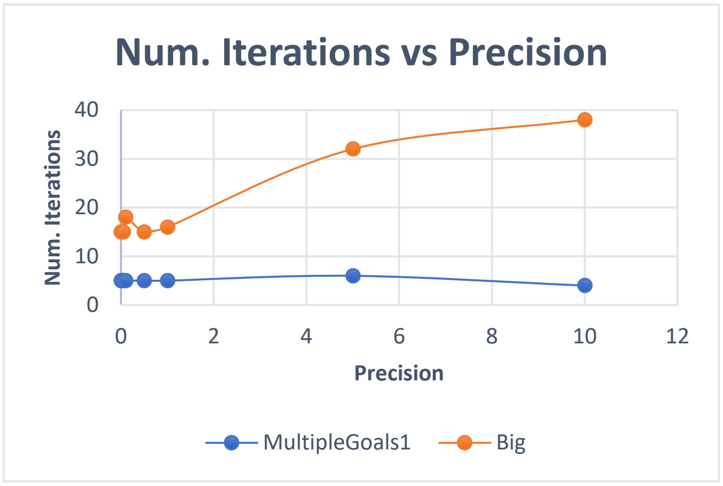
Tuning Value Limit (values of 100, 500, 1000, 5000, 10000, 25000, and 50000), Iteration Limit (values of 1, 50, 100, 500, 1000, 2500, and 5000), PJOG (from 0 to 1 in increments of 0.1), and Precision (values of 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, and 10), we arrive at the following curves: (Note: When tuning one hyperparameter, the rest were set as their default values, underlined above, PJOG default = 0.3)

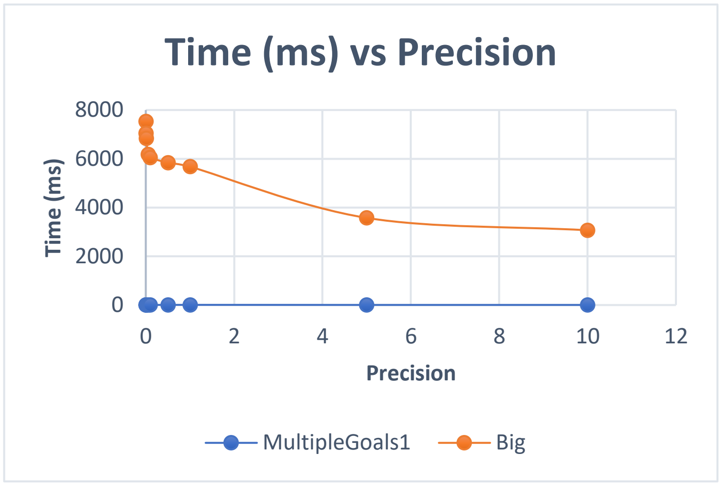


While changing PJOG, value limit was set at 5000, iteration limit was set at 500, and precision was set at 0.001. From the curve to the left, we see that the number of iterations is generally higher for big.maze than multipleGoals.maze, but between PJOG values of 0.2 and 0.7, we see similar number of iterations till converge for both MDPs. This is probably due to the higher value of the Iteration Limit (as each state is evaluated 500 times per cycle of Policy Iteration) thus allowing for less overall Policy Iterations.

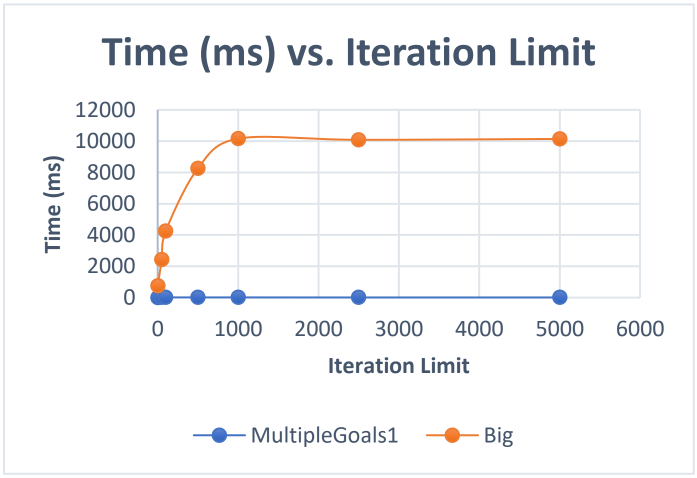


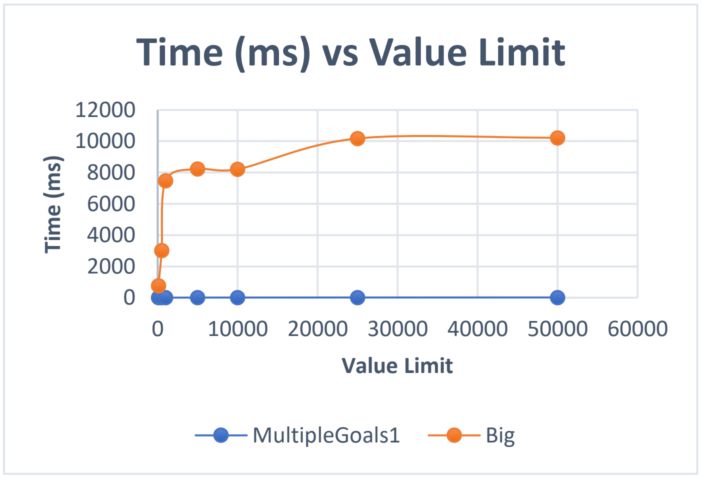
Looking at the time till convergence, we see that big.maze took more time for every PJOG value, but had a drastic reduction in time till converge after PJOG of 0.8 (we discussed a similar trend in the previous section of value iteration). Comparing the time with value iteration, it seems that policy iteration usually takes longer (on the magnitude of three times as long) to converge even if the policy at converge is identify for both MDPs.





Looking at Precision (curves above), we see a similar trend in time till convergence as we saw with value iteration, where the time generally decreases as the precision value increases. However, we see a very different trend when observing the number of iterations. When we ran value iteration and tuned precision, we observed the number of iterations decrease as precision increased, but for Policy Iteration, specifically looking at big.maze, we see the number of iterations actually increases. One explanation for this could be that allowing more leniency in concordance with states whose values change every iteration (confusing states near edges of the gridworld or surrounded by walls) is making it harder for the algorithm to converge. One possible solution would be to simply lower the level of leniency.



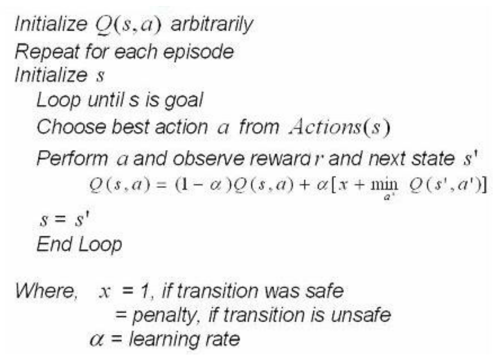


Looking at just the time (ms) vs both Iteration Limit and Value Limit for multipleGoals1.maze and big.maze, we see a similar trend: a sharp rise in the time till converge and then a relative plateau. Looking first at the Iteration Limit, it makes sense that the higher the iteration limit, the more time it will take the algorithm to run and converge. For each state, we are increasing the workload by a factor of the Iteration Limit which then increases the total time expended. Looking next at Value Limit, if we limit the value each state can have, then it makes for the algorithm to converge much faster at smaller state value limits than larger state value limits. What is interesting is the plateau region for both hyperparameters as it allows us to simply select the value for Iteration Limit and Value Limit that best results in the optimal policy without having to worry about increases in algorithm run time.

Overall, Policy Iteration returns the same (or highly similar) policy for both MDPs as Value Iteration, but it does generally take longer for the algorithm to run and converge. Knowing this, we can make an informed decision on which algorithm to select when solving our MDPs. After selecting the algorithm, we can tune the appropriate hyperparameters to minimize algorithm run time while balancing policy changes. Let us now look at solving MDPs if we do not know or cannot control the transition function or the reward function.

Q-Learning

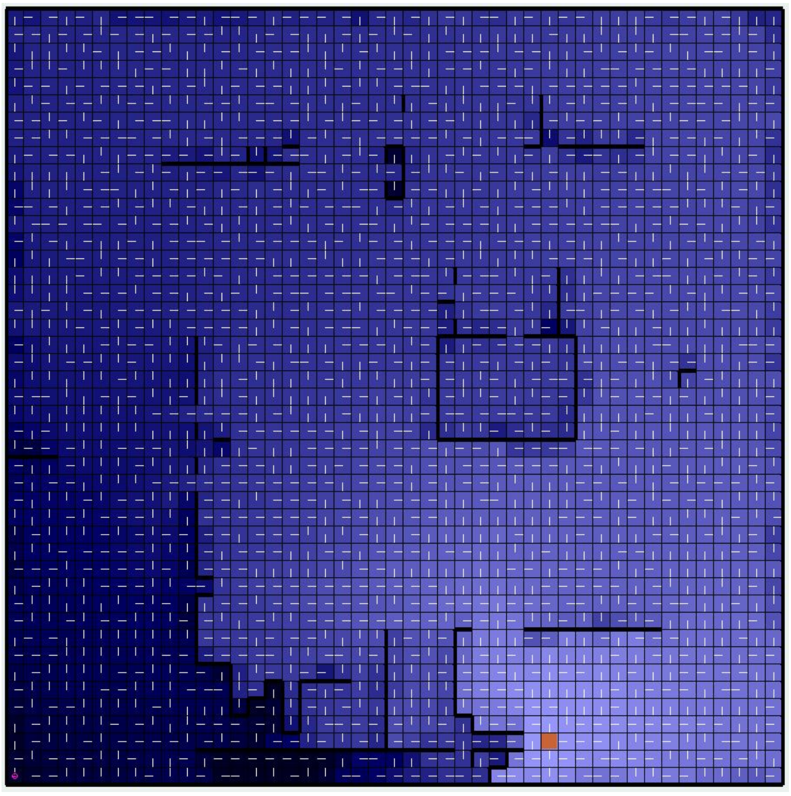
Q-Learning is a reinforcement learning algorithm used when the transition function and reward function are unknown. Values for these two functions are constantly estimated and updated by taking samples and recording observations as the agent traverses through the MDP. RL_Sim's Q-Learning algorithm is shown here:

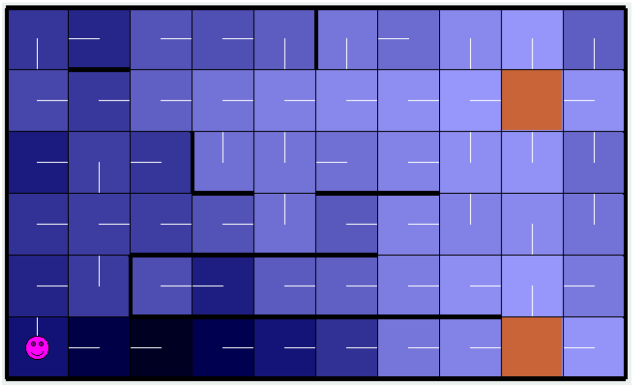


```
Initialize $Q(s,a)$ arbitrarily
Repeat for each episode
Initialize $s$
    Loop until s is goal
    Choose best action $a$ from $Actions(s)$
    Perform $a$ and observe reward $r$ and next state $s'$
        $Q(s,a) = (1-\alpha)Q(s,a) + \alpha[x + \min_{a'} Q(s',a')]$
    $s = s'$
    End Loop

Where,  $x$ = 1, if transition was safe
          = penalty, if transition is unsafe
        $\alpha$ = learning rate
```

Some things to note: Q-Learning allows for agent exploration with some probability ε, where the agent may take a suboptimal path to the goal state to explore its surroundings, and also includes a learning rate α, which decides by how much to update the old Q-value of a state, action pair.

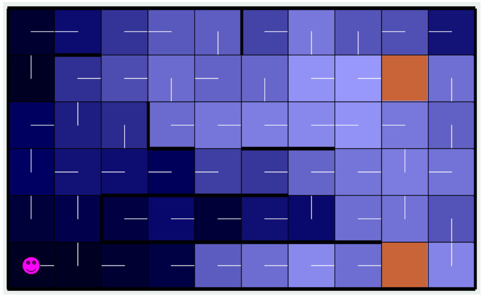
Setting PJOG at 0.3, Precision at 0.001, Learning Rate at 0.1, and Exploration Probability, ε, at 0.1, we arrive at the following policies after 1000 episodes of running Q-Learning for each of the two MDPs:



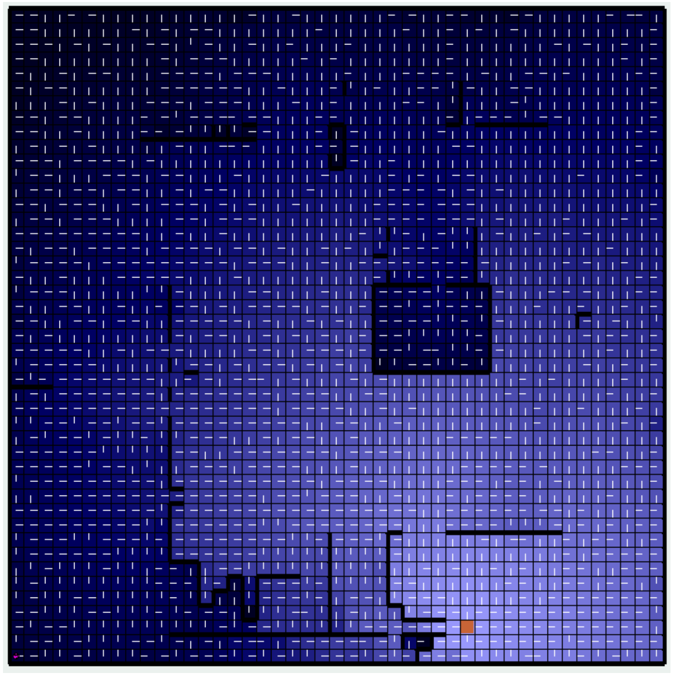


Comparing the policies generated by Q-Learning and those generated by both Value Iteration and Policy Iteration, we see some similarities for both gridworld problems, multipleGoals1.maze and big.maze, however, there are also differences that are worth exploring. First, looking at multipleGoals1.maze, we can observe shifts in the policies of states in between the two goal states. In both Policy Iteration and Value Iteration, the policy of the state exactly in between the two goal states points North, whereas the policy for that same state points South for Q-Learning. Furthermore, we see changes in policy near the edges of the gridworld, and running Q-Learning multiple times changes the policy in these states. This is just a by-product of the algorithm, as it does not know the transition function or reward function ahead of time and is using samples to estimate those values. Allowing Q-Learning to run for more episodes changes the policies in these “questionable” states, but the policies near the goal states are more or less fixed. We see a similar result when looking at big.maze, where the Q-Learning policy differs from the Value Iteration and Policy Iteration policies in states near walls and boundaries, but is similar, with some noise, near the goal state. Ways we can improve Q-Learning include: allowing the algorithm to run for more cycles (more cycles lead to better estimation of the transition and reward functions), tune the exploration and learning rate parameters (discussed below), and add more agents (more Q-Learners) to speed up algorithm convergence time.

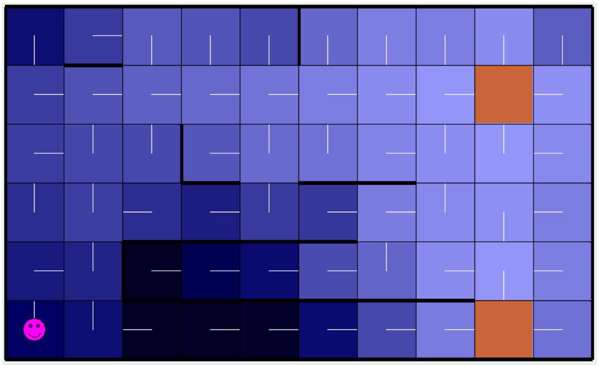
To explore the impact of hyperparameters, I tuned the learning rate (default at 0.7, decaying) and the exploration probability, ε, and report some of the results below (I do not report all the results to maintain brevity of the analysis). When disabling decay of learning rate, we get the following policy for MultipleGoals1.maze:

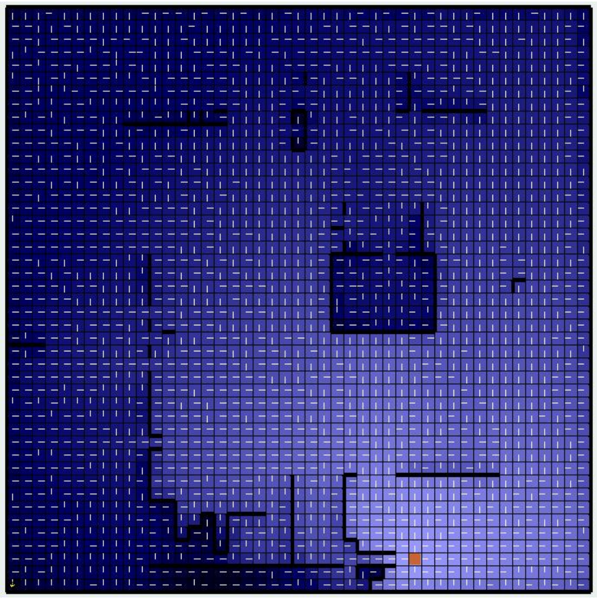


Comparing the policy to the left with the policies from Value Iteration and Policy Iteration, we see differences in how the agent should act at states near the top and right boundaries. The learner must not simply had enough data to decide to move down and closer to the goal state and instead moves left and right to explore its neighborhood. Also some states near the goal state, especially the one directly below the top goal state, have odd policies, once again due to the lack of enough cycles.



The big.maze policy shown to the left is a result of setting the exploration probability to 0.99, thus allowing the agent to explore more than it exploits the state-value gradient. Interestingly enough, it performs very well near the goal state and arguably outperforms Value and Policy Iteration near walls, observe the policies of states near the single horizontal wall right above the goal state. One possible explanation for this could be that exploration is more valuable than simply taking the right action when dealing with obstacles impeding an agent's progress. Combining a high exploration probability (0.99) and a low, decaying learning rate (0.1), we get the following policies for multipleGoals1.maze and big.maze respectively:





Comparing these policies to those of Value Iteration and Policy Iteration, we see lots more similarities than we saw from just the basic Q-Learning policy. Looking first at the multipleGoals1.maze, we observe the top and right boundary policies are more in congruence with the Value and Policy Iteration policies and the states adjacent to the goal states have policies that directly instruct the agent to the goal state, something we struggled with in basic Q-Learning. Looking next at the big.maze, we see similar trends, where the policy gathered with high exploration and low learning rates resulted in a similar outcome to when we ran Value and Policy Iteration. Note the similarities in policies in the wall box (center right) and the policies near the single goal state. A possible explanation for this phenomenon could be that exploration is key in these gridworld problems with lots of walls and having a low learning rate means that we reduce the impact of each observation on our overall Q-value for each state, action pair, this reducing the impact of noise on our function approximation for both the transition and reward functions. Finding the optimal exploration probability and learning rate would be crucial in optimizing the Q-Learning algorithm for any particular MDP. It is also important to note that Q-Learning was able to achieve comparable results with Value and Policy Iteration even without knowing the transition or reward function. This allows us to utilize these reinforcement learning techniques in real-world applications, where we often do not know the underlying Markov Decision Process governing the problem we are studying.

Conclusion

Overall, I gained a deeper understand of MDPs and some of the algorithms we can use to help us solve such problems. Value Iteration and Policy Iteration, before this project, appeared very similar to me, but after messing with the hyperparameters and seeing their influence on the overall policies generated, I was able to extract key differences between the two algorithms, specifically the impact of the arg-max / arg-min function in Value Iteration that is absent in Policy Iteration. As for Q-Learning, I was able to appreciate how an algorithm was able to generate an effective policy even without knowing the two core functions of an MDP. As with all ML-problems, we need more data and more time to fully understand the big picture.