1 Introduction

This project investigates the use of different MDP and reinforcement learning algorithms to solve MDP problems.

1.1 MDP problems

Both of the MDP problems are variants Gridworld. Gridworld is a rectangular board with multiple tiles, and from a certain tile s, the agent can choose to move up, down, left, or right. When choosing an action a from state s, the agent will move to some state s' with a transition probability $T(s, a, s')$. The goal is to find an optimal state-to-action mapping such that the expected reward is maximized over several states.

One of the problems is GridWorld on a 10 x 10 grid, and the other one is GridWorld on a 70 x 70 grid. For the purposes of these experiments, GridWorld is interesting, since the two-dimensional state space allows the optimal action from each state to be readable to the human eye. Thus, the arrangement of the states in gridworld allows one to effectively tell how well one algorithm is doing compared to the other just by eyeballing the actions at every state.

1.2 Experiments

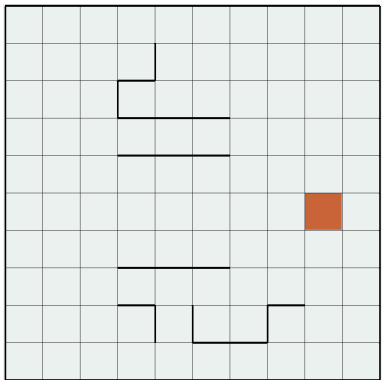
The actions in these experiments have some stochastisity, given a state and action, there is a 70 percent change that the agent will travel to the intended state and a 30 percent chance that the agent will travel to another random state. In the RL-MDP simulation, this feature is controlled by the value of the PJOG parameter, which is set to .3 (which means there is a 1 - .3 = .7 chance of the agent going in the intended direction).

2 Results and Analysis

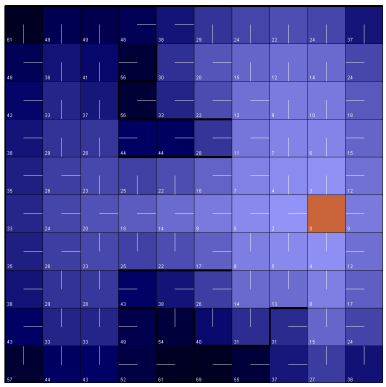
Below are some screenshots of the running of the various experiments. At each tile, there is a line segment, with one endpoint at the center of the tile and the other endpoint in another direction. The direction of each segment at each tile represents the optimal action to take from that particular state. Also, there is a number on each tile, which is proportional to the expected utility that would occur if the particular policy is chosen.

2.1 Maze: Fewer States

Here is the image of the particular maze that was used for the experiment with fewer states:

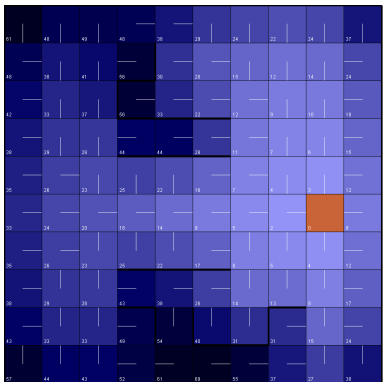


2.1.1 Value iteration



With a precision parameter of 0.001, PJOG = 0.3, and wall penalty of 50, this algorithm took 61 steps to converge, taking a total of 69 ms.

2.1.2 Policy Iteration



The iter. limit parameter (the maximum number of times for the policy to be evaluated) was set to 10, and the value limit (the maximum number of value updates within each policy iteration) was set to 5000. The algorithm converged within 6 policy evaluation iterations, taking a total of 19 ms.

Here, policy evaluation worked more quickly than value iteration and still gave similar result on the maze tile values. Of course, each policy iteration took much longer than each value iteration, since the number of steps in the algorithm was equal to $S^2 * 10$, but there were much fewer iterations in the policy iteration algorithm.

For each state in each value iteration, the algorithm needed to consider what the expected utility by moving up, down, left, or right. This was unlike what happened in policy iteration, where the algorithm only considered the result of one action, allowing for fewer computations.

Also, each value iteration involved observing what happens when moving either direction only once, while policy iteration involved observing what happens when moving in a fixed direction several times. This made the value iteration algorithm more prone to noise from taking each action from a state, since each action is taken only once; since policy iteration takes that fixed action ten times from that state, it is able to disregard stochastic irregularities for a particular action.

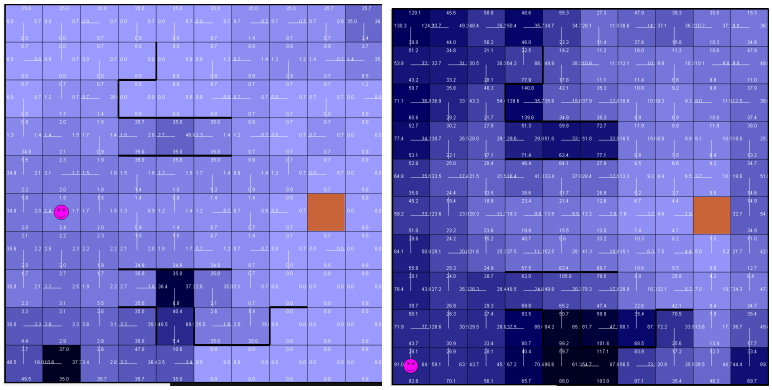
The policy iteration experiment discussed here mentions a 53 ms run that is caused by 10 inner iterations in policy evaluation. However, this can be sped up even more by decreasing the number of inner iterations. The total number of policy iterations doesn't increase here, since the expected number of tiles that random noise would affect is smaller with this small maze state.

The faster time for fewer inner policy iterations and the fixed number of total policy iterations can be confirmed by the table below (which measures policy iterations and time for varying numbers of inner iterations):

Inner iterations	Policy iterations	Time
1	6	4ms
2	8	8ms
5	8	12ms
10	6	19ms

2.1.3 RL

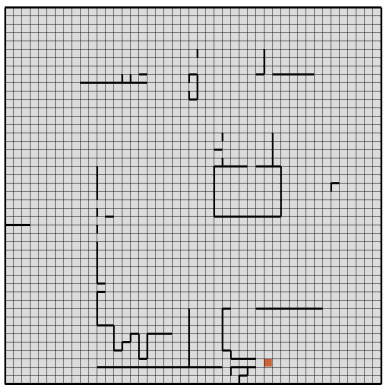
On this MDP, Q-learning was done for 1000 episodes.



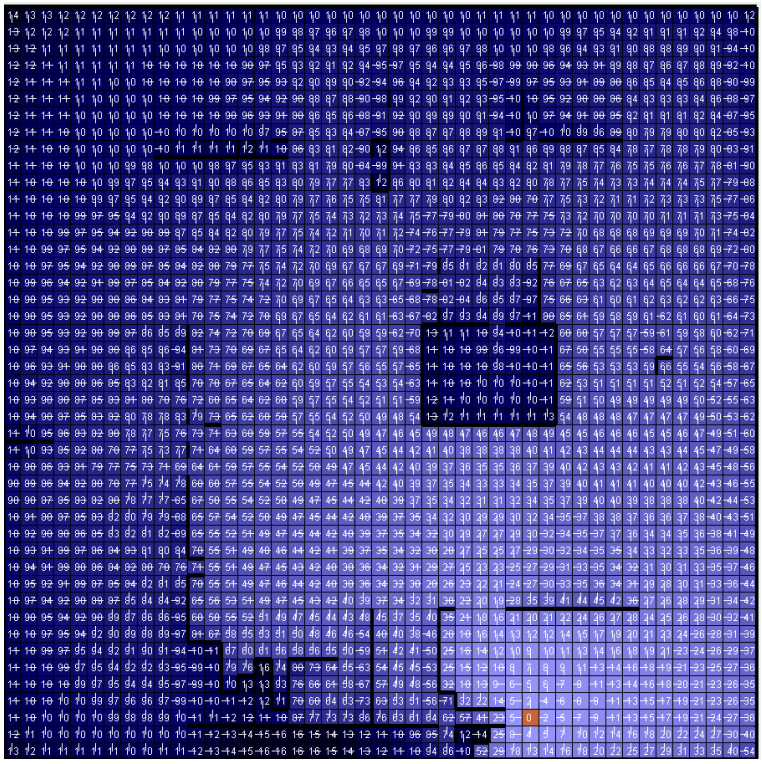
There were much more iterations done for Q-learning than for the other two algorithms, but the policy is less clear.

2.2 Maze: More States

Here is the image of the particular maze used for a large number of states:



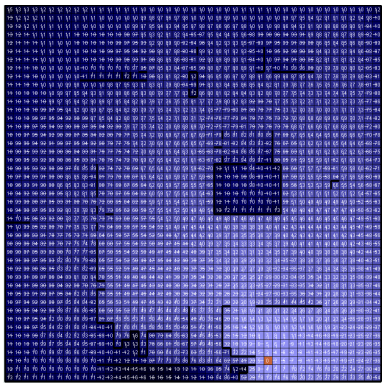
2.2.1 Value iteration



The image above is the final distribution of the values and policies

The value iteration converged after 181 steps, and the algorithm took 10,871 ms.

2.2.2 Policy Iteration



The policy iteration converged after 18 steps for 10 inner policy iterations, and the algorithm took 2,841 ms, which is much faster than the value iteration algorithm.

However, using a smaller number of value updates for policy iteration here produces a different result when compared to using a smaller number of updates in the small maze, as shown below:

Inner iterations	Policy iterations	Time
1	104	3516ms
2	55	2908ms
5	8	12ms
10	18	2841ms

First of all, the number of policy iterations decrease as the number of inner iterations increases. This makes sense since there exist several squares that are very distant (at least 5 squares away) from any walls. Thus, for fewer inner iterations like one iteration, it takes multiple value updates for the information that a wall exists to propogate to certain squares; thus, more policy iterations are needed for fewer inner iterations.

Also, the algorithm takes more time to run with fewer inner iterations, unlike what occurred with a smaller maze. Even though each policy iteration is faster due to fewer inner iterations, there are so many policy iterations that the total time is ultimately increased.

Another reason for the longer runtime with fewer inner iterations is that there is a very large number of tiles, so more tiles are expected to be sensitive to noise due to action stochastisity. As explained previously, having fewer value updates causes certain policy estimates to be vulnerable to noise, so there would be a delay in the runtime of the algorithm.

2.3 RL Algorithm Results

The reinforcement learning algorithms have to undergo an exploration/exploitation tradeoff, which is not present in the MDP algorithms, since the MDP algorithms already have perfect information about the rewards and the transition functions. Thus, the reinforcement learning algorithms ended up taking a significantly longer time than the MDP algorithms. Even in the 10 x 10 maze, where the state space is quite small, there are still 100 total states and almost 400 possible actions, which the agent meeds to try one at a time in order to achieve the information that an MDP has.

3 Sources

[1] https://www.cs.cmu.edu/ awm/rlsim/