Utkarsh Garg
CS 4641 - Machine Learning
Assignment #4
4/18/2015

Markov Decision Processes Report

The Two MDPs

The MDPs chosen for this assignment are both grid worlds. Grid worlds in general are interesting MDPs, because they can model a lot of the stochasticity of the real world, with very little complexity. Rewards and penalties can be modeled either into actions or states or actions corresponding to states. Restrictions can be imposed simply by adding walls and there is always at least 1 start and goal state. The grids can be manipulated to encourage exploration or exploitation or a combination of both.

State and Action Definitions: A state in this context is simply defined as the square of the grid the agent is in and an action can be up, down, left or right. Note: the agent is not allowed to stay in the state it is currently in.

The grids chosen for this report are as follows:

Small MDP: The smaller grid world is a 4 X 4 grid, with just **16 states**. The start state is the bottom left square and the goal state is the top right square. The walls have been arranged in such a way that initially, while exploration, the agent is prone to get stuck. The penalty for colliding with a wall i.e. for an illegal movement is -50. The grid has been arranged in such a way that the agent should soon learn to not get stuck in the two traps and should avoid them altogether. Also, there are two optimal policies in the maze. It will be interesting to see whether the agent choses one over the other or whether they are equally weighted. It will also be interesting to note the difference in the values of the two paths as a function of the number of iterations; with *sufficient* iterations, do the two paths ultimately come out equal? Both policies are uniform as they is literally no difference in them in terms of reward/penalty.

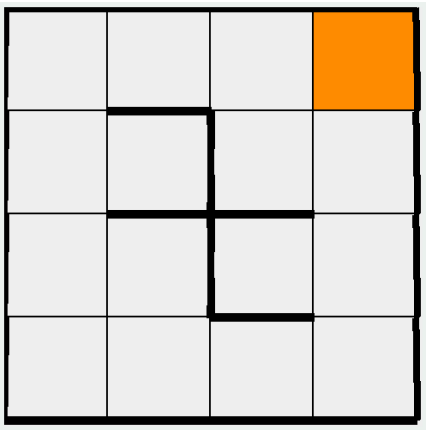


Figure 1.

To make things more interesting, some of the experiments have been performed on variations of this grid. There are 4 versions of the world:

1. The one described above; with 2 optimal policies and 1 goal state. (Figure 1)
2. With only 1 optimal policy and 1 goal state. The agent should learn to always go right in the beginning. (Figure 2)
3. With 2 goal states and one optimal policy for one and 2 for the other. (Figure 3)
4. With 2 goal states, each with only 1 optimal policy. (Figure 4)

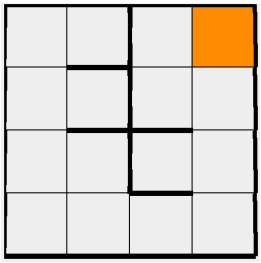


Figure 2.

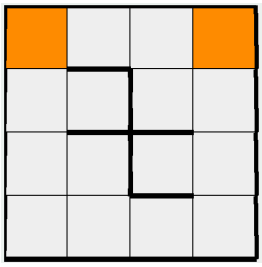


Figure 3.

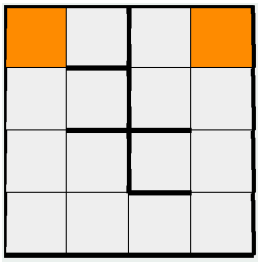


Figure 4.

Large MDP: This grid world has **225 states**. The state definition is the same as for the grid above. There are 3 goal states and the start position is the bottom left square. One of the goals has no walls around it, one goal has just one and the other is surrounded with walls. The traps in this grid are much more sinister than the ones in the small MDP, they can lead to multiple steps of wasted work, with high penalty. Reaching the topmost goal can either have the agent go through the big open space, or get stuck in the trap near it. The goal state just below that is surrounded by walls and the agent might find the penalty too much to attempt. The goal on the bottom right on the other hand is completely open, and seems to be the easiest for the agent to converge to. It will be interesting to compare the performance of the same parameters in different algorithms for the small and large MDP. It will also be interesting to see whether or not the optimal policy found is the simple path of going all the way left to the goal.

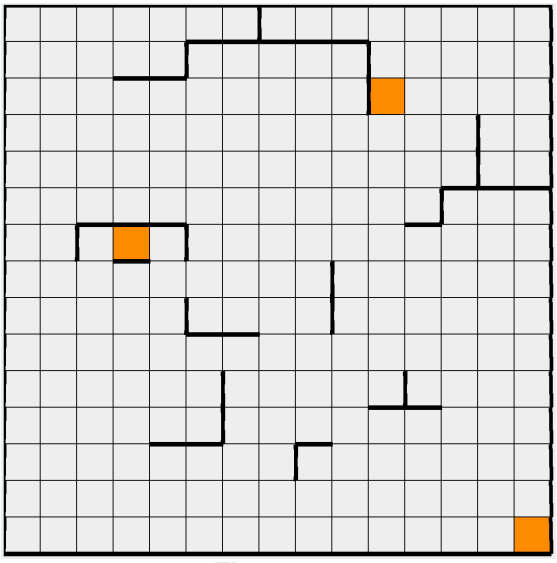


Figure 5.

Again, to make it even more interesting, there are a total of four variations of this grid:

1. The original one described above (Figure 5)
2. With only 1 goal state and less walls(Figure 6)
3. With only 1 goal state and many walls (Figure 7)
4. With 3 goal states and less walls (Figure 8)

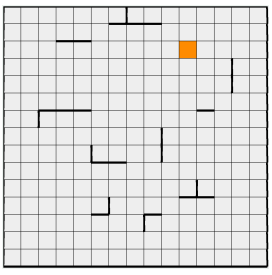


Figure 6.

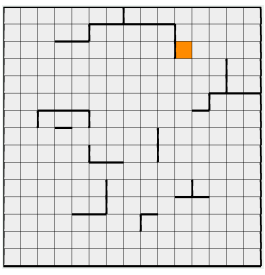


Figure 7.

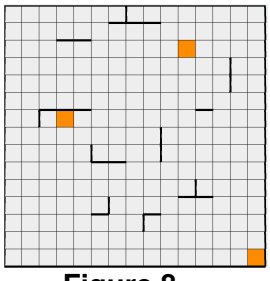


Figure 8.

Overall, the two MDPs represent very differently sized state spaces with the same action space. Some scenarios are intuitively easy while others might take a while to learn. It will also be interesting to see the variance in performance based on varying levels of stochasticity.

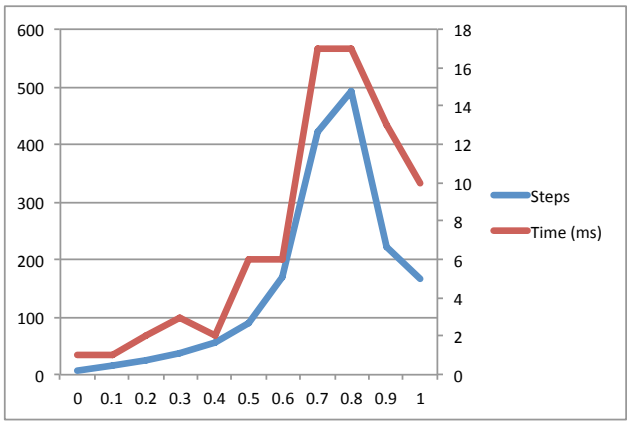
Value Iteration and Policy Iteration

Two experiments were performed for each, value iteration and policy iteration:

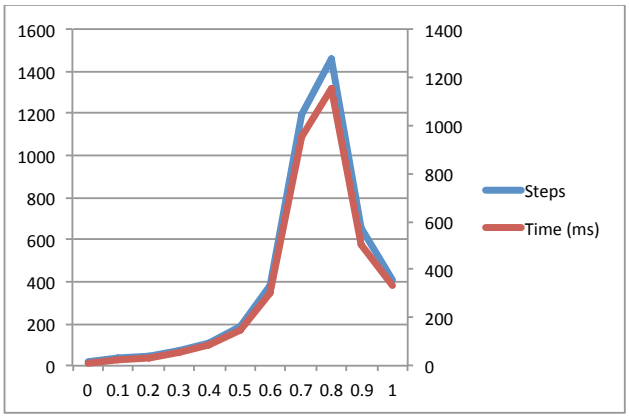
Experiment 1 - Varying stochasticity: The first experiment varies the non-determinism of the agent from 0-1 in steps of 0.1. A stochasticity of 0.1 means that the agent will execute the required action with a 90% probability, and 10% of the time, it will perform one of the other actions. A stochasticity of 0 means that the actions are deterministic and there is no noise in the required and executed motion; a stochasticity of means that the iterations will never execute the required action. I expect this to result in a completely random policy with low performance.

Value Iteration: The goal of value iteration is to find the optimal value function V^* for the discounted infinite horizon problem satisfying the Bellman equation. This optimal value is the convergence condition (which is reached by setting a precision value, 0.001 in this case). This optimal value is then used to find the optimal policy for the agent.

Results: Both, the small MDP and the large MDP showed a very similar pattern. As the stochasticity increased (increased randomness/noise), the number of steps required to find increased up to a certain point; the point of 70%-80% stochasticity. After that however, in both mazes, the steps and time to convergence started decreasing. The increase in the number of steps and the time is intuitive - more noise leads to more errors, and thus it takes a longer time to find the optimal value. The decline at 80%-100% randomness however is intriguing.



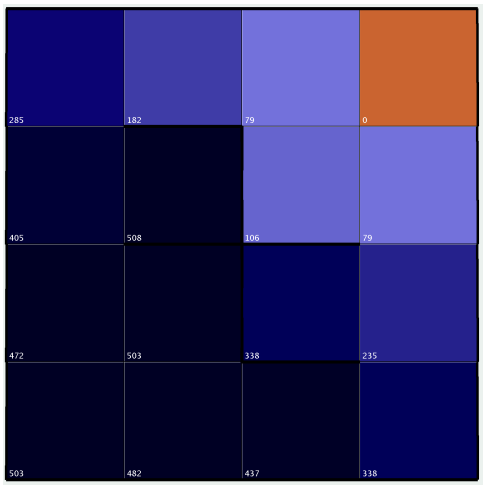
Small MDP



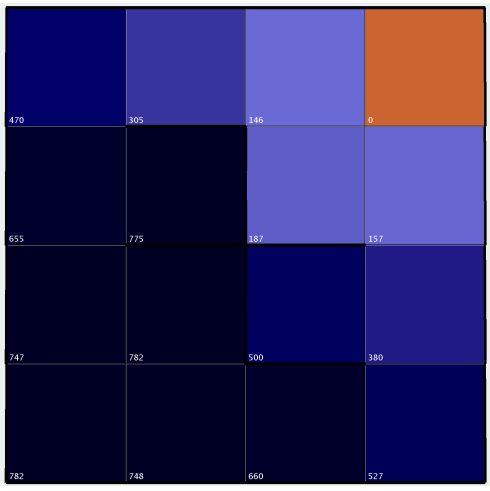
Large MDP

In the two plots below, the red square is the goal state; the darker blue a grid square is, the higher the penalty for going to that state is. From the two plots, it seems like the entire trap area has been blacked out and the squares in the direction of both optimal policies are very dark. This means that a lot of penalty has been accumulated over the iterations.

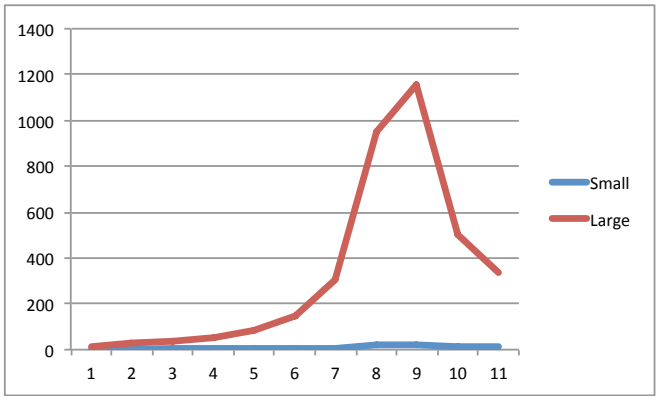
90% stochasticity



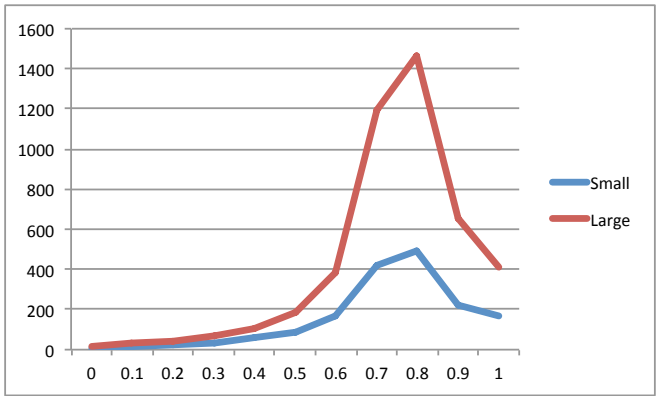
100% stochasticity



From the values in the squares, and the two line plots right above them, it is clear that though increasing the noise almost to a 100% decreased the steps and time taken to converge significantly, the penalty accumulated is very high. This is because the agent bounced around so much in each iteration that it gather much more penalty than in the partially deterministic cases, and thus had a higher information gain. Thus, the agent took longer to complete each iteration, but because of the amount of information gained in each, it took less number of iterations to converge. The same is true for the large MDP. This is partially possible because both MDPs have a very limited number of states (16 and 225). Had there been 20,000 states, we would have seen a similar trend in the number of iterations, but not in time; there would be very high information gain in each iteration, but it would also take very long.



Time comparison (ms)

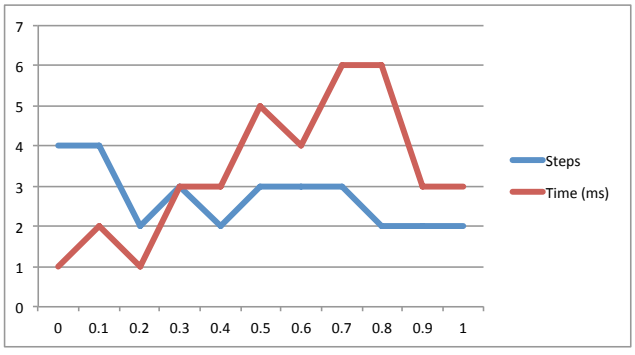


Steps comparison

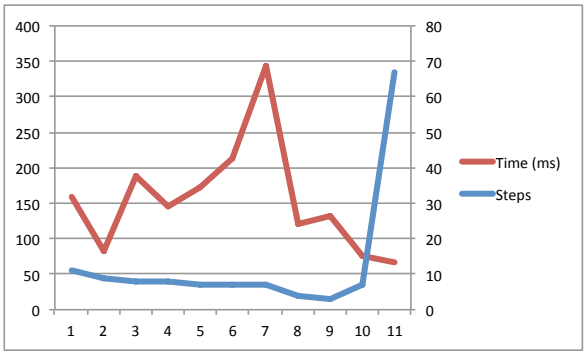
The two graphs above show the variance in the time and the number of steps, as functions of the stochasticity; it is a comparison of the big and the small MDP. Compared to the large MDP, the small one converges much faster, and that is expected; the pattern of increase in time however is very different. In the small MDP, increasing stochasticity does not increase time by much; in the large one however, it increases by several magnitudes. This signifies that the number of states greatly influences the performance of value iteration. From the graph on the right, both the small and the large MDP show an increase in the number of steps, AND follow the same pattern; the magnitudes however are very different.

Policy Iteration: Policy iteration's goal is not to find the optimal value, but to find the optimal policy. It is basically value iteration iterating over policies instead of individual state values. It makes more sense simply because the gol is to find the optimal policy. It computes utilities for states, given a policy, updates the utilities and subsequently, the policy. It iterates until values converge (convergence specified by a precision value, 0.001 in this case).

Results:

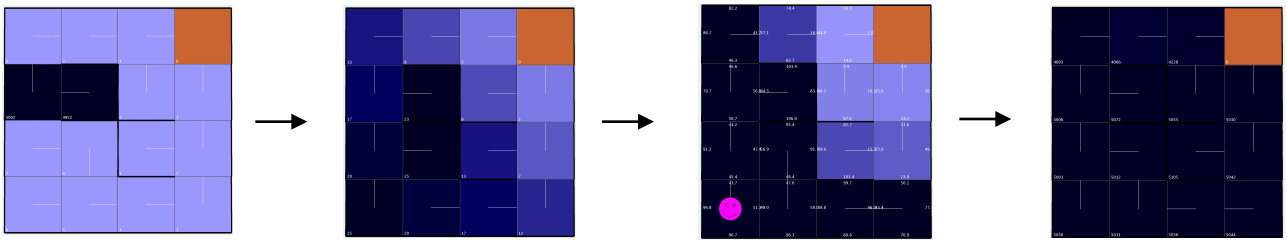


Small MDP



Large MDP

From the graphs above and the grid plots below, for a small MDP, increasing stochasticity in general increases the the time, while the number of steps taken remains fairly constant. From the plots below, it is clear that with with little randomness, policy iteration finds the optimal policy without accumulating any significant penalty. In the deterministic case, after the policy suffered penalties for going back and forth in the maze, it seems to not go there anymore and simply follow the optimal policy. In 100% randomness however, it seems like the policy has accumulated close to the maximum possible penalty for each state before converging to the optimal policy. Again, as in value iteration, the number of steps and time taken both decrease at 100% randomness. For the large MDP, the pattern is not quite the same. In the large MDP, though the time peaks at a noise:policy iteration imbalance, and decreases from there, the number of steps shoots up by a magnitude of 7. This shows that, though policy iteration does not require additional time, it does need a lot more exploration (using a random world to arrive at an optimal policy) to converge. This makes sense because the goal here is to find the optimal transition function and in a random world with 100% uncertainty, the algorithm needs to run many times before it can definitively arrive at the optimal function.

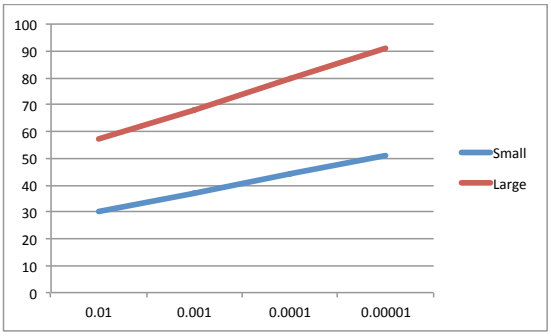


0% randomness **10% randomness** **70% randomness** **100% randomness**

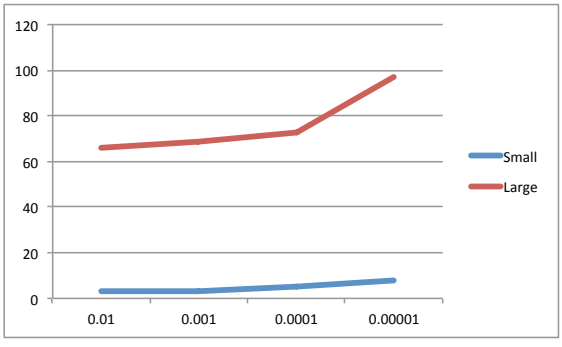
Note: The peak at around 70% noise and a decline in the amount of work required to converge after that can be explained in another way. with 80%+ randomness, it might be fair to say that the algorithm being used it a randomized optimization and that the 10%-20% of the VI or PI is basically noise. This explains perfectly the results as e see an improvement; the improvement however is not as good as a deterministic world, and that is to be expected, since the algorithm being run is a randomized optimization. This is also analogous to how despite its completely random nature, RCA is able to find some valuable components to achieve a performance comparable to PCA and ICA.

Experiment 2 - Varying precision: In this experiment, the precision i.e. the condition for convergence was varied. The values ranged from 0.01 to 0.00001, in increments of a magnitude of 1/10th.

Value Iteration Results: With increasing precision, the number of steps required to converge expectedly increased; a harder convergence condition means that the change in the value of state has to be even less for termination. The amount of time also increased because of increasing number of iterations. The magnitude of increase however is more for the large MDP, both in terms of time and steps. This reinforces the observation that value iteration is dependent on the number of states.

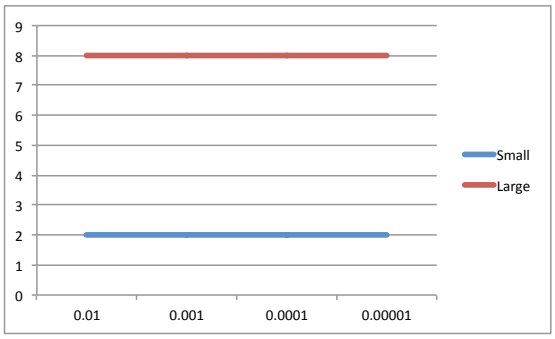


Steps

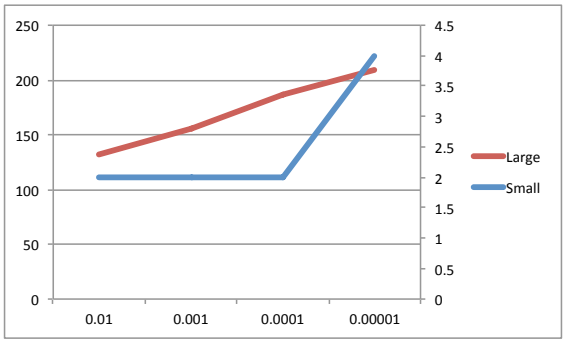


Time

Policy Iteration Results: The number of steps, both for the small and large problem remained constant while the time increased; this means that there was more exploration per iteration with increasing precision, to obtain the optimal policy.



Steps



Time

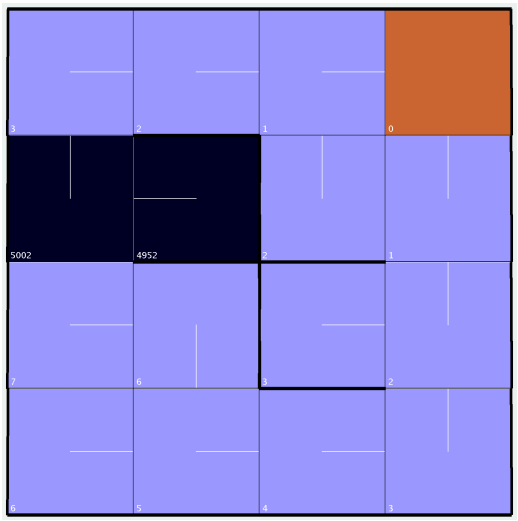
Comparison - Value Iteration and Policy Iteration

Performance: In terms of performance, policy iteration performs significantly better than value iteration in terms of the number of steps but is significantly slower in terms of time (for the large MDP, there is no difference in the small MDP); with increasing or decreasing randomness or precision, policy iteration is always takes less iterations, but a longer time to converge.

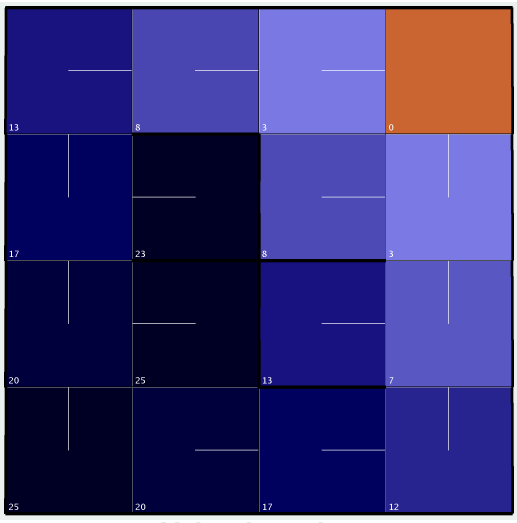
Standard Parameters (0.3 stochasticity and 0.001 precision)	Value Iteration	Policy Iteration
Small MDP (Time (ms))	3	3
Large MDP (Time (ms))	53	146
Small MDP (Steps)	37	3
Large MDP (Steps)	68	8

Effect of number of states: In terms of steps, policy iteration is unaffected by the number of states however, takes much longer to converge. Value iteration on the other hand sees a significant jump, both in time and the number of steps. This behavior is expected since value iteration optimizes the value of every state while policy ` iteration aims to find the optimal policy for the entire MDP.

Optimal Policy - Small MDP:



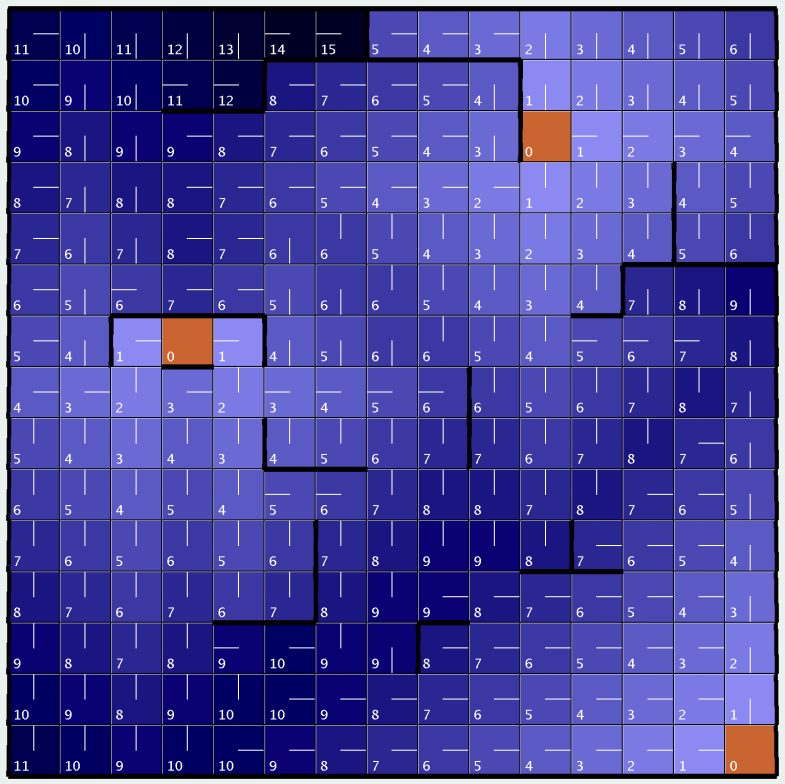
Policy Iteration



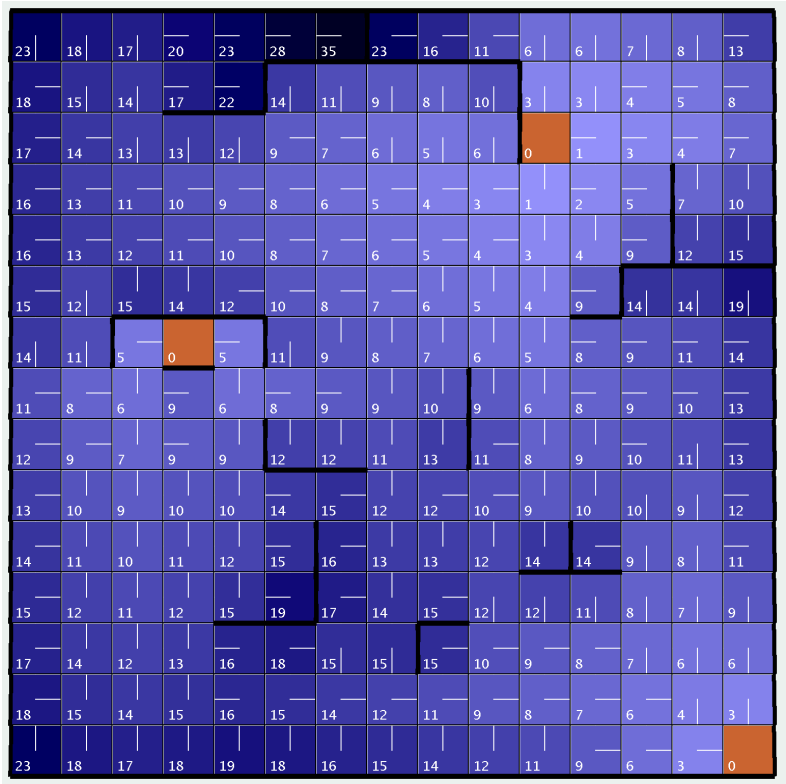
Value Iteration

In the two grid plots above, the white line in each state represents the optimal policy. As is obvious from the images, both PI and VI come up with the same optimal policy for almost all states. From the color of the squares however it is clear that value iteration suffered much more penalty, which is expected since VI took so many more iterations than PI. There are two possible optimal policies, and both algorithms found both of them, it just took VI many more tries to do so. In PI, there is one very dark area which seems like the agent went back and forth a lot, gathered penalty and never went there again, simply found the optimal policy and followed it.

Optimal Policy - Large MDP:



Policy Iteration



Value Iteration

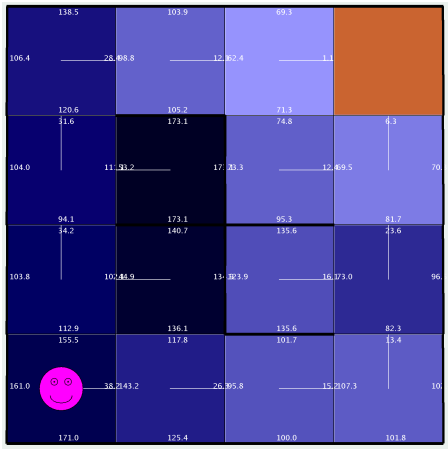
In the case of the large MDP, there are 3 goals and several optimal policies. The one with the least penalty is the one surrounded by the maximum walls; this is in contrast to the expected result. The optimal policy I expected was for the agent to just go right and each the goal closest and easiest to reach. the policy in policy iteration for that goal is very close to this expected policy, but in value iteration this only happens very close to the goal. The reason for both approaches not going towards the intuitive obvious optimal policy is perhaps the complexity of the maze. The penalties near the goal states are minimal and those on traps are quite high. Looking at the values in the grids, it is obvious that value iteration has had much more penalties than PI, simply because of the higher number of iterations, and thus higher source of error. In summary, both algorithms seem to find similar optimal policies, provided the same parameters.

Reinforcement Learning: The reinforcement learning algorithm picked for this report is Q-learning. I have worked with various variations of Q-learning in the past, with Dr.Isbell and Dr. Thomaz, so it was a natural choice.

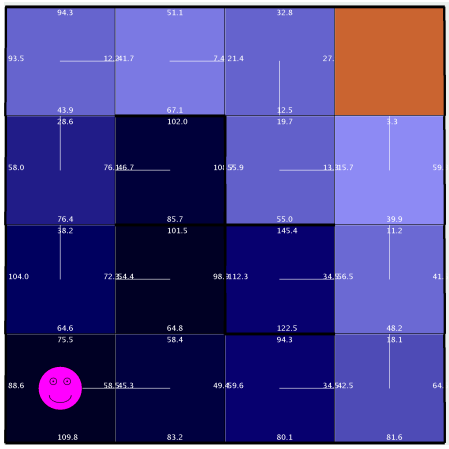
Q-learning: Q learning is a model-free reinforcement learning algorithm. It deals with the limitation of policy iterations and value iteration that requires the agent to have domain knowledge. In Q learning, the agent simply needs to know the states and the possible/legal actions in them. Each state has its own Q-value, which can be considered as the reward for being in that state. It's update equation updates as the Q-value of the current state, plus the discounted Q-value of the next state, resulting from an action. Q learning has an additional parameter epsilon which determines he probability of taking the best action. The best action is defined as the one that maximizes the Q value update function. For example, if epsilon is 0.1, it means that the algorithm will choose the best action 90% of the times, and another random action the rest of the 10% times. This is done to balance exploration and exploitation and also to introduce additional, agent based stochasticity into the environment. Q learning tries to maximize the expected reward.

Experiment - Varying Epsilon Values: In this experiment, the epsilon values used in Q learning were varied from 0 (always choose optimal action => greedy epsilon approach) to 1 (never choose best action), in increments of 0.1. The learning rate is maintained at a constant 0.7. Varying the epsilon values should greatly affect the performance, given the same number of cycles. If the number of cycles for high epsilon is increased, there should be no difference, the output should be the same optimal policy; but, for comparison, the number of cycles is maintained at a constant 800.

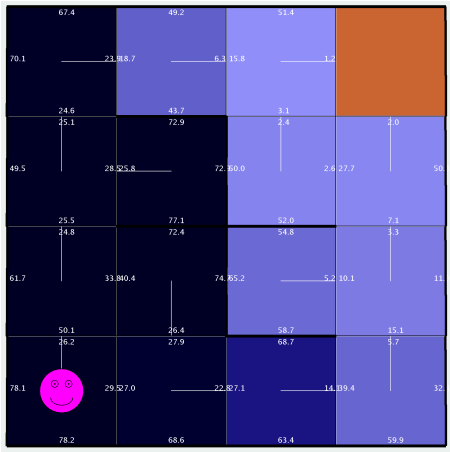
Results:



Epsilon = 0

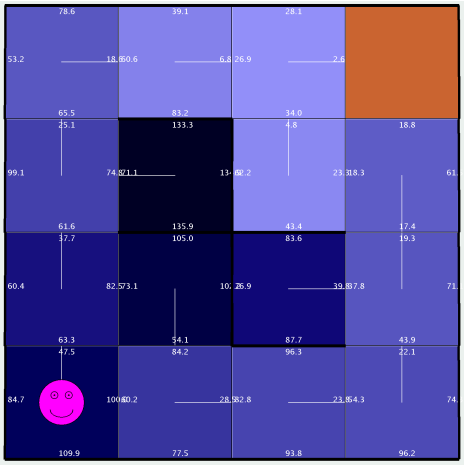


Epsilon = 0.6



Epsilon = 1

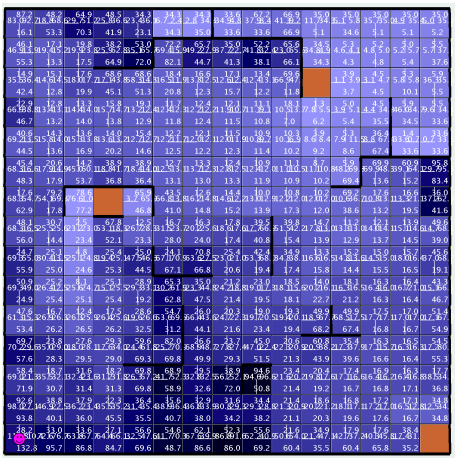
The epsilon 0 approach finds both optimal policies. On increasing epsilon gradually, it finds only one optimal policy and only partially finds the other one (only the last 3 units of the optimal policy RRRUUUU are found). When epsilon is 1, Q learning is still able to find one optimal policy, but accumulates much more penalty and much more spread out penalty (the penalty with epsilon 0.6 though more than epsilon 0, is still concentrated to a region). For the epsilon 1 policy, half of the grid has very high penalty, yet there is only a partial optimal policy. This is simply because the approach is random and is not dependent on the Q values, thus voiding Q learning altogether. From the data collected, epsilon=0.3 was found to be the most optimal (found both optimal values and had least accumulated penalty) for



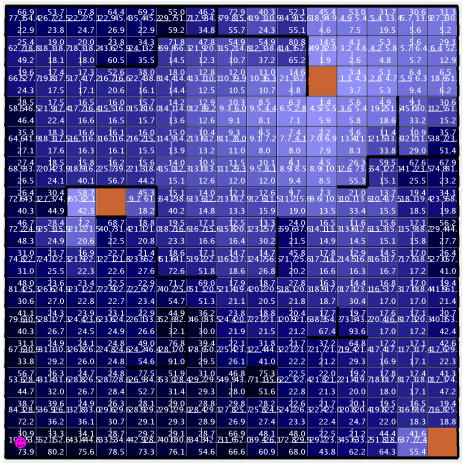
Epsilon = 0.3

this maze.

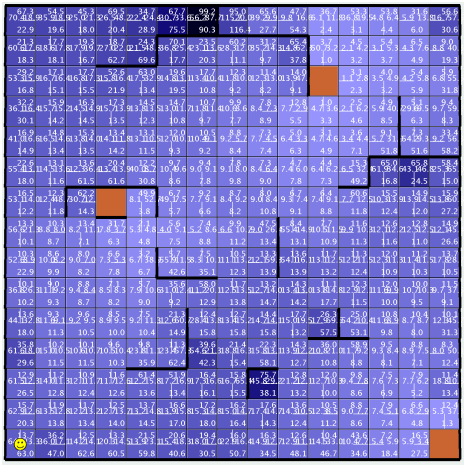
For the **larger MDP** the results were very different. Surprisingly, epsilon 0.9 was the optimal, in terms of having the least penalty and finding an optimal policy to all three goals. Epsilon 0, the greedy approach performed very poorly, accumulating a lo of penalty and finding only partially optimal policies to the goals. As epsilon increased, at epsilon 0.3, it seemed to accumulate the maximum amount of penalty. But there on, as epsilon increased, penalty decreased. It seems like the complexity of the MDP requires more exploration before converging to an optimal policy. Epsilon 0.9 performed the best, its randomness seems to provide the *optimal* exploration of all values experimented with; a greedy approach, or even a partially greedy approach seems to fail miserably for this maze.



Epsilon = 0



Epsilon = 0.3

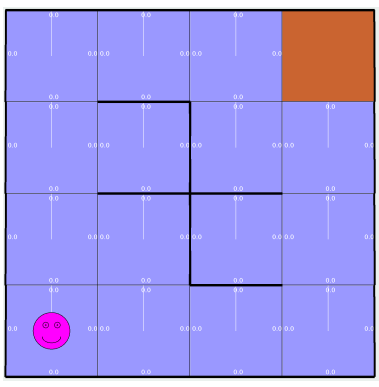


Epsilon = 0.9

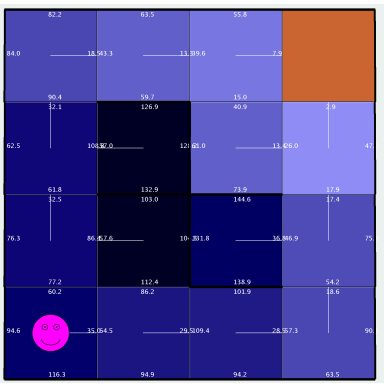
Note: The odd performance for the large MDP might be attributed to the fact that this was also run only on 500 cycles and the increased number of states require an increased number of cycles for appropriate results.

Experiment - Varying the Learning Rate: In this experiment, the learning rate is varied between 0 (only the current state matters, maximize only current reward) and 1 (the next state matters as much as the current state, maximize the combination), in increments of 0.2. A learning rate of 0 should result in a greedy approach, with the algorithm trying to maximize only the current Q value, thus leading to no learning; a learning rate of 1 is not expected to have the best outcome, a balanced learning rate is required, that adds the discounts the next state by an *appropriate* amount.

Results: For the both MDPs, like expected, learning rate of 0 results in no policy, as the algorithm does not learn and tries to maximize only the current state, thus treating each state as independent from one another. The optimal learning rate for the small MDP turned out to be 0.5, as it found both optimal policies and localized penalty the maximum i.e. the rest had penalties scattered throughout the domain. Increasing the learning rate anymore



Learning Rate 0



Learning Rate 0.5

seemed to give too much value to the next states and only one of the two optimal policies were found.

For the **larger MDP**, an intermediate learning rate seemed to perform better (optimal = 0.4). A lower learning rate didn't seem to be able to grasp enough relations between next states to make optimal transitions and anything more seemed to be obstructed by the complexity of the MDP problem. In conclusion, optimal learning rates for both problems seem to be pretty close, though the reasons are not exactly the same.

Experiment - Varying the Number of Cycles: In this experiment, the number of cycles is varied from {500,1000,10,000,100,000}. The purpose of this experiment is to analyze the improvement in the performance of the algorithm with number of iterations.

Results: For the **small MDP**, the optimal performance (finding both optimal policies with least penalty in states) is in the minimum number of cycles in the test set i.e. 500 cycles. Increasing the number of cycles also found both optimal policies but increased penalty in the states. This is probably just because the stochasticity has been set to a default of 30%; thus even though Q learning might have already found the optimal policy, it accumulates error because of the inherent randomness in the algorithm. Note: Though the penalty did increase, it did not increase substantially.

In contrast to the above, the **large MDP** had the best performance with 100,000 cycles; with increasing cycles, the penalty declined and an optimal policy to all three goals was achieved. There was a big performance jump from 500 to 10,000 cycles and then a slight improvement with 100,000 cycles. This means that though 10,000 cycles are a good estimate, a little more will benefit the algorithm, but if pressed for resources, going all the way tp 100,000 cycles is unnecessary in this case. The large MDP requires many more cycles than the small MDP because of its increased complexity, not only in a larger state space, but also with an increased number of goal states and larger, more elaborate traps.

Comparison - Value Iteration and Policy Iteration vs Q learning: In conclusion, policy iteration always took less iterations than value iteration. Q learning took magnitudes more iterations than policy and value iterations, but also resulted in a better performance, with a tweaked epsilon and learning rate. ! learning requires a lot of decisions on the algorithm designers part (these decision can be made using experiments as above and don;t have to be guesses, but are simply additional initial work) and Q learning is more computationally intensive. However, the it was able to find optimal policies to multiple goals in a complex maze, while keeping penalties less than those of policy and value iterations. We noticed that increasing the number of steps unnecessarily in PI and VI only increased significantly the penalty, however, in Q learning, with the correct valued parameters, this does not happen, and the designer need not worry about it.