# MARKOV DECISION PROCESS

Tahsin Ekram

## Introduction

This analysis explores the performance and behavior of different reinforcement learning techniques on simple discrete markov decision problems.

## Problems

### Windy Frozen Lake

The windy frozen lake problems is a variation of the frozen lake problem that is part of the open gym ai library. The windy frozen lake is comprised of the same elements as the original frozen lake which include a starting point, holes, frozen areas of ice and the end goal. The agent obtains a reward of 1 when it reaches its goal cell and it is given a zero on all other transitions. In addition to that the windy version of this frozen lake consists of specific locations in the grid which are windy. Thus whenever agent falls in the windy area it is setback by 1 unit in a random direction which could be up, down, left or right. Furthermore this problem adds stochasticity by injecting a slippery term where the agent after taking a certain action can succumb to the slipperiness of the frozen sections and not end where it initially intended to. The algorithms for this problem are compared based on the average reward obtained. First each algorithm is trained and then the optimal policy generated from the training phase is passed to a testing phase where it explores the environment until the optimal policy reaches the terminal state. The cumulative reward is calculated during this testing phase and average reward is calculated. Thus the expectation is for the policy to consistently reach its goal and achieve a reward of 1 and the number of times it was able to do that across all the times it played is a good measure.

The interesting aspect about this problem is that the unpredictability injected by the windy cells makes it a difficult path to navigate. Thus it will be a challenging environment for the algorithms to solve which may eventually provide a greater perspective into what parameters are key drivers of success in resolving these types of problems with these algorithms

### Gambler's problem

This problem is taken from the Reinforcement learning book by sutton. The problem is comprised of a gambler who has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal set by the problem, or loses by running out of money. On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP.
The state is the gambler's capital, $s \in \{1, 2, \ldots, 99\}$. The actions are stakes, $a \in \{0, 1, \ldots, \min(s, 100 - s)\}$. The reward is zero on all transitions except those on which the gambler reaches his goal, when it is +1.

The problem can be posed as an interesting one by creating a biased coin where a probability of flipping heads is low thereby creating a difficult situation for agent to win at each flip. The interesting thing to observe here it would be how and when the bets of the agent will be radical vs conservative.

## **Value and Policy iteration Windy Frozen Lake**

The following bullets highlights the key observations and their explanations based on the results from running policy and value iteration for the windy frozen lake problem.

1. Based on figures 1 and 2  both policy and value iteration yields a final optima policy and its associated value which are identical This highlights one key aspect of mdp problems. the absolute value of utility of states does not matter that much; what matters is the relative rankings of utility between states. While determining  the policy for both cases we perform an argmax function whose return only depends on the ranking not on the absolute values. Thus no matter what the magnitude of these values are as long as the correct choice has the largest value we will get the same optimal policy.
2. Policy iteration converges faster with lower number of iterations compared to value iteration. According to figures 3 and 4 the overall amount of iterations taken for value iteration is higher than policy iteration. An interesting thing to note here is that policy iterations has a large reduction in number of iterations as discount factor increases whereas the iteration rises with discount factor with value iteration. This can be attributed to how both approaches in finding the optimal policy. Policy iteration keeps improving the value function at each iteration until the value function converges. Then at the policy improvement step the policy is redefined and the value is computed according to the new policy. Thus in the event when the policy converges before the actual value function policy iteration is able to yield the optimal policy at a lower iteration.
3. Figures 5 and 6 show the average rewards for different state spaces for  value iteration. One key observation would be that for the larger state space the average rewards is the highest for the highest discount factor whereas for the lower state space the rise in rewards with discount factor is more gradual. Value iteration is based on the assumption that no matter what the policy is we are certain that the utility of a state is the sum of its immediate reward and the utility of its successor state with a discounted factor. Thus value iteration computes this utility for each state across all the possible actions. Thus whenever we are at a certain state s' we can be sure that it is the best outcome among all possible paths given that value iteration deemed it to be with the highest value. So with a larger state space the value of a state is dependent on a large set of successor states and with a lower value of the discount factor the value iteration is dominated more by the immediate reward which makes it choose states that are not in the best interesting of producing the highest result in the future. As a result with a high discount factor the future rewards are given more weight which allows it to choose a better next state.

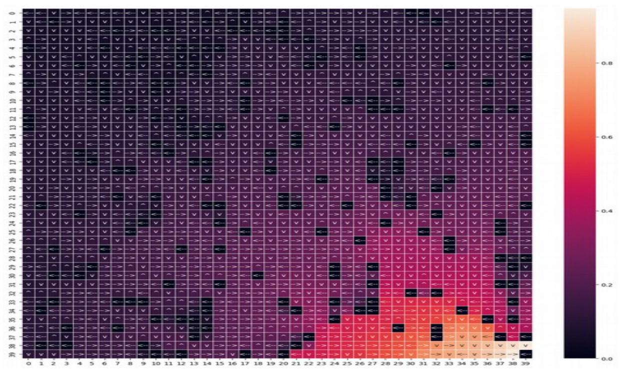


Figure 1 Value iteration policy heatmap

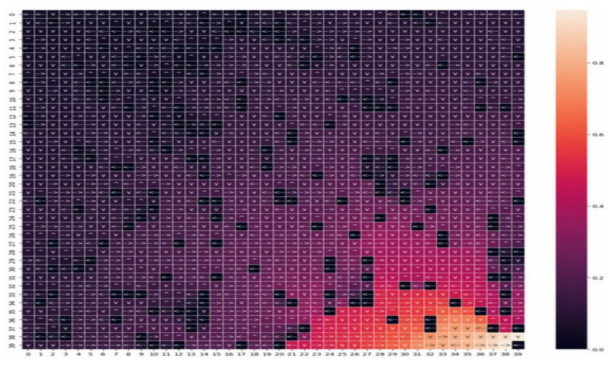


Figure 2 Policy itertaion policy heatmap

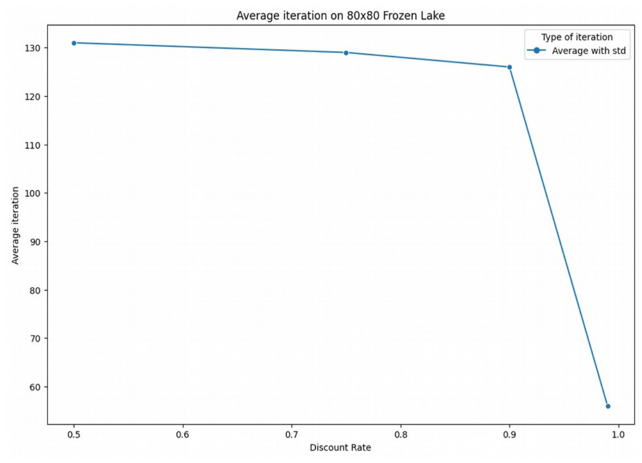


Figure 3 Policy iteration average iterations

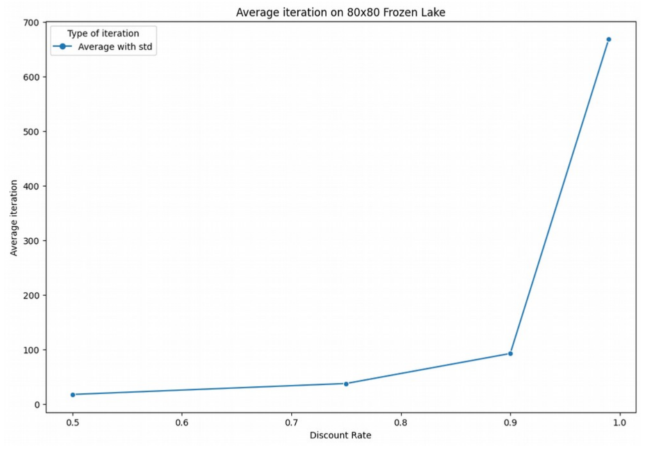


Figure 4 Value iteration average iterations

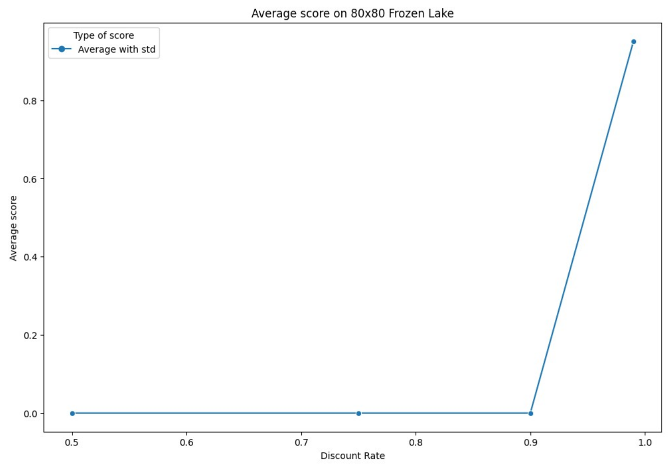


Figure 5 Value iteration rewards 80x80

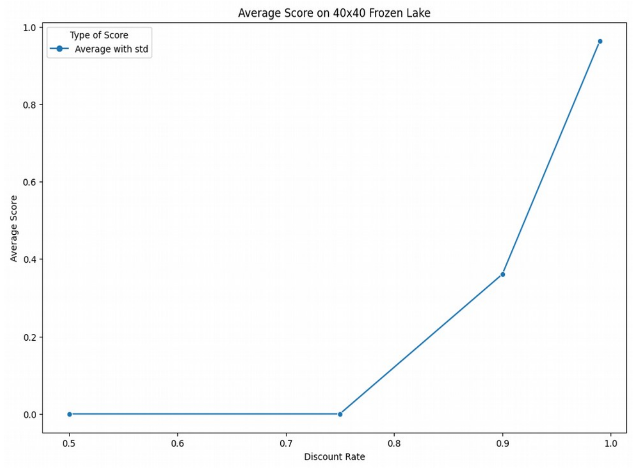


Figure 6 Value iteration rewards 40x40

## Q-learning on windy frozen lake

Figures 7,8 and 9 summarizes the relationship between average rewards and discount factor, learning rate and decay rate from applying q learning to the windy frozen lake problem. Based on the figures we can observe that the average reward for Q learning is below 0.8 whereas for the value and policy iteration it is nearly 0.9. This brings out a key difference between the two types of the algorithm. Value and policy iteration being a dynamic programming algorithm is required to have full knowledge of the transition probabilities which it utilizes to visit every possible state and action before they can find an optimal policy. However, Q learning utilizes the concept of temporal difference which allows the agent to learn optimal policy by exploring and interacting with its environment via action. Despite this ability to interact and explore Q learning seemed to have produced a lower average reward. One reason could be due to the nature of the grid navigation aspect of the problem. The windy frozen lake not only contains holes in the grid but adds a level of unpredictability by introducing windy blocks where agent experiences a set back in a random direction if it lands on it. Thus given the finite length of episode to run the q learning and the large state space it is unable to reach its goal and achieve a reward of +1 and wanders the grid aimlessly earning a reward of 0. However, value and policy iteration handles this better as it has full knowledge of all the states and whenever it comes up with a policy it already has accounted for these areas of holes and unpredictability. Figure 10 and 11 highlights the rewards between varying state spaces while keeping the number of episodes same. It can be observed that the lower state space yields a better average reward.

I found this very interesting as q-learning is equipped with the ability to perform one step lookahead which is similar to projecting one step forward and ask what is the next best action. Previous attempts estimated the expected return for each state using only knowledge of that state. Equation 3-5 allows the agent to look ahead to see which future trajectories are optimal. This is called a rollout. The agent rolls out the states and actions in the trajectory (up to a maximum of the end of an episode, like for MC methods). The agent can create policies that are not only locally optimal, but optimal in the future as well. The agent can create policies that can predict the future. Thus it can be deduced that the number of episodes is a key factor of q learning. Given enough episodes the action-value estimates approaches the correct expected return.

Furthermore, figures 7, 8 and 9 highlight some key concepts of q-learning. Given that q learning has the ability to control how much faith it will put on the newly learned information vs the old information by the learning rate it plays a significant role in the final average reward. With a lower learning rate as seen in figure 9 there is a higher reward as it allows the q learning to take the new information with a grain of salt and incorporate knowledge of the previous path taken into account. The decay rate as shown in figure 8 is responsible for controlling the exploration vs exploitation tradeoff. With a lower decay rate the algorithm stays in the exploration phase for a longer time which allows it to randomly choose an action which can sometimes lead the agent in to a new path that may lead to an optimal policy.

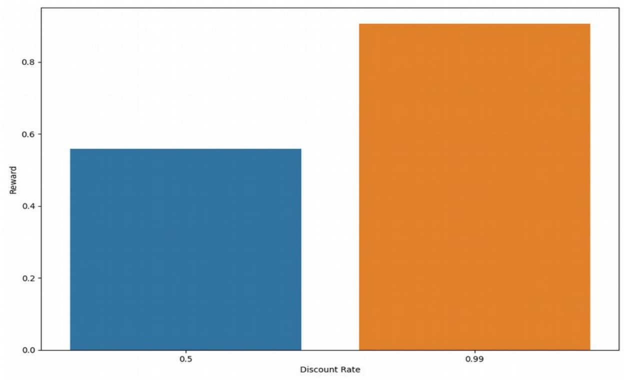


Figure 7 Discount vs rewards

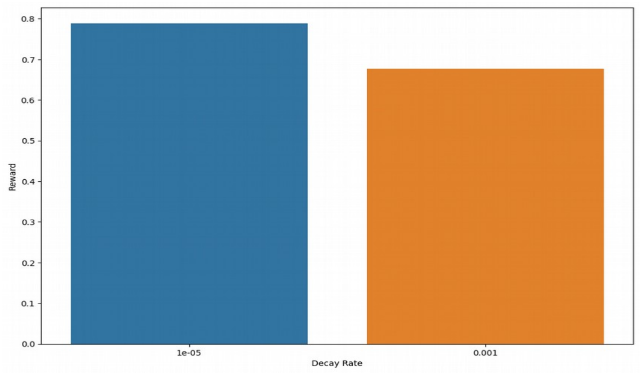


Figure 8 Rewards vs Decay rate

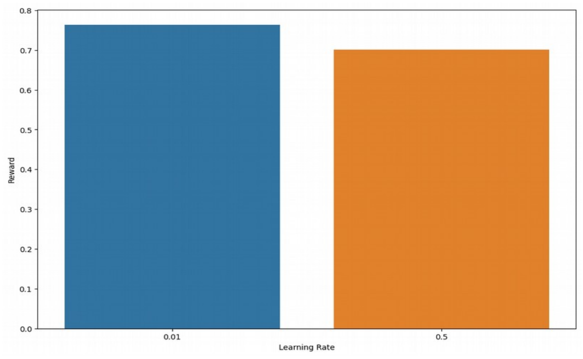


Figure 9 Reward vs Learning rate

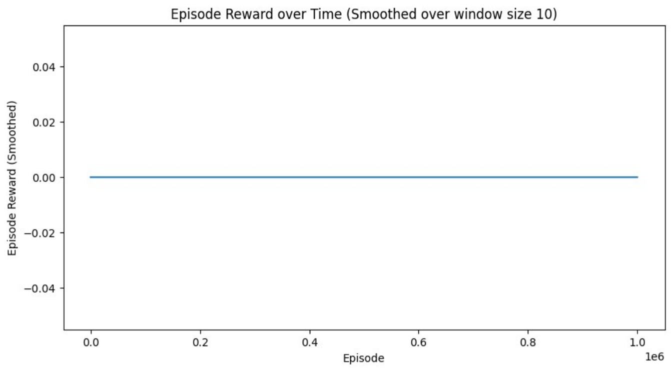


Figure 10 Qlearning large state space

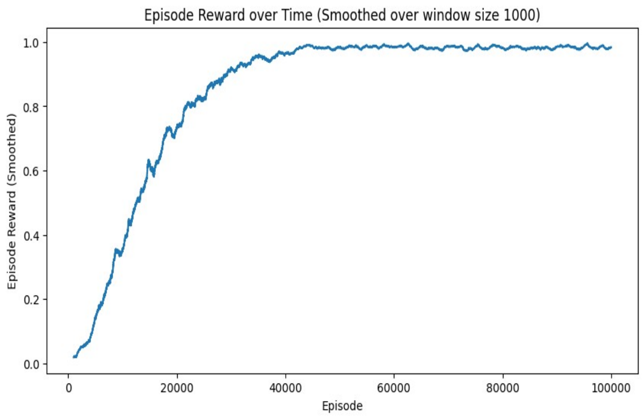
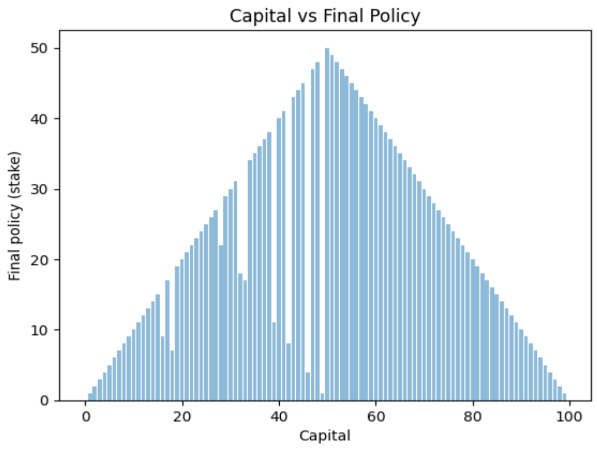


Figure 11 Q learning small state space

## Value and policy iteration for gambler problems

Figures 12 and 13 summarizes the results for the gamblers problem where the intended final state of capital is $100 and the probability of heads or winning is 0.35, which is low. We can see that both value and policy iteration yields a policy that is able to control conservative vs radical betting. We can notice that once it reaches a capital of halfway it is smart enough to bet conservatively to reach its goal of $100. It cautiously holds back on betting a large portion of its capital to risk losing all of it or cross the limit of $100 since its goal is to exactly land on $100 where it receives its reward. This can be again attributed to the fact that both policy and value iteration is fully aware of its environment and the exact bet or action to take in a particular state.



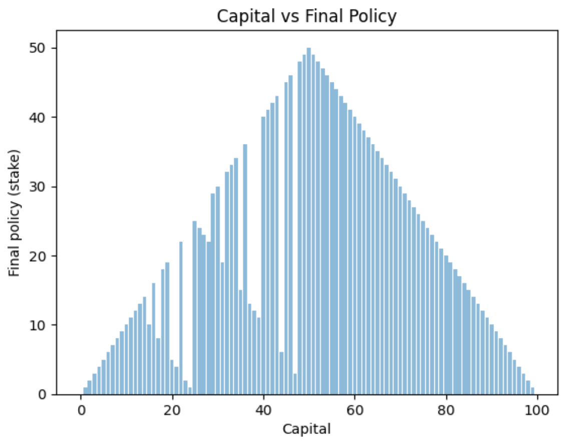


Figure 12 Policy Iteration

Figure 13 Value Iteration

## Q-learning for gambler problems

The figures 14 and 15 highlight the results from the policy generated by Q learning with varying learning rate. There is a slight difference in the policies in the initial stages. The policy with the low learning rate was able to place bets in a more organized manner than the one with the higher learning rate. Instead of gaining capital it was very conservative and didn’t bet more or less than a certain amount for a while unlike figure 15 where the policy bets an amount proportionate to the capital it gained from the previous bet. This is makes sense as a higher learning rate is more inclined to ingest the new feedback and consider the previous knowledge less whereas a lower learning rate keeps track of the previous knowledge to make a more informed decision.

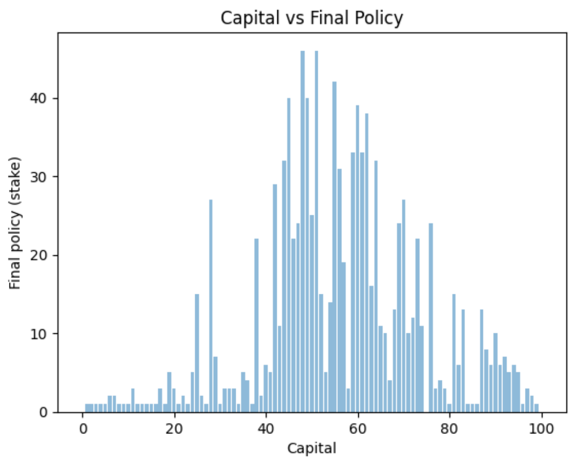


Figure 14 Learning rate 0.5

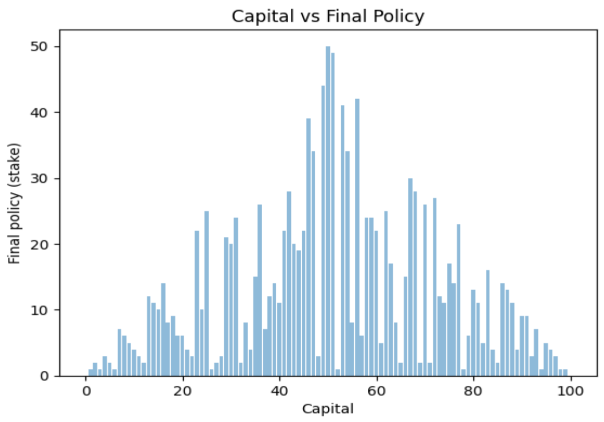


Figure 15 Learning rate 0.01

Figure 16 and 17 shows the episode length for varying learning rate. The average episode length for is higher for the one with the higher learning rate. Given that learning rate is the key determinant of how much past and new information will be ingested into the qtable update a process discarding past information will have a hard time getting to an optimal policy since the sequential nature of a MDP deems it highly dependent on sequence of past information.

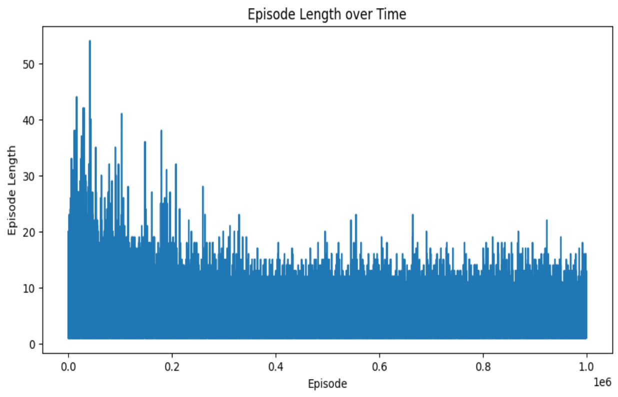


Figure 16 Learning rate 0.01

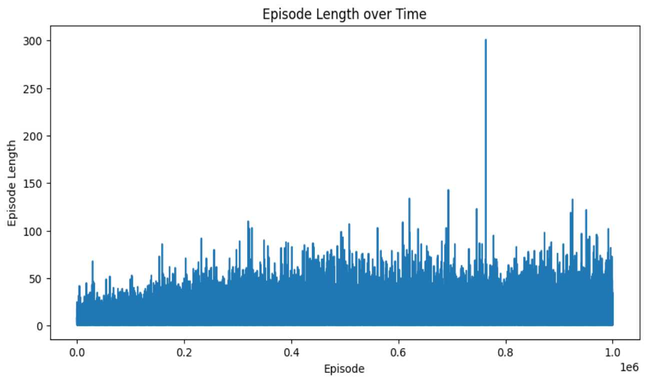


Figure 17 Learning rate 0.5