Introduction

For this assignment, I chose two Markov Decision Processes (MDPs). As a small MDP, I chose forest management from PyMDPToolbox. For the large MDP, I adopted the Frozen Lake problem from the OpenAI gym toolkit. I used Value Iteration (VI), Policy Iteration (PI), and Q-Learning to solve the problems.

The forest management problem is a non-grid world problem with 10 states. Each state represents a year. At the end of the year we are provided with two actions: wait, or cut. There's a chance that the forest catches fire. In the case of cutting the trees or the fire burning the forest, we are thrown back to the first (youngest) state. This is an interesting problem as it could model how we make a decision under uncertainties (fire).

The Frozen Lake problem consists of a 30 by 30 grid world. Each cell in the grid represents a state. The goal is to reach the bottom right corner cell without falling into the holes. In order to get the Q-learning algorithm to converge, I had to modify the provided class to add intermediate rewards. I find this problem interesting as it could model a simple warehouse problem where a robot aims to pick up a delivery and drop it off at a predetermined location.

I find these problems interesting together because they present different dynamics. The forest management problem presents the agent with only two actions, so going to a previous state is not possible unless the agent starts over. This limitation in choices makes exploitation much more profitable than exploration. On the other hand, the frozen lake problem the goal is so far away that without enough exploration the agent was not able to find a valid policy. Therefore, these problems display the weaknesses and strengths of different algorithms effectively.

Summary of key findings

This section summarizes important observations and results. The following sections will dig deeper into each of these.

1. **VI/PI**
    a. **Convergence Criteria**. I looked at different metrics and chose maximum reward gain to determine convergence for both VI and PI. I defined a threshold based on the values of gamma and epsilon. I used consistent values so the VI-PI comparison is valid.
    b. **Convergence Behaviour**. In both MDPs, PI converged much faster than VI. The reason appears to be that VI's stopping criterion checks the changes in the reward even though the algorithm may have found the policy earlier.
    c. **Policy Comparison**. In the case of forest management MDP, both algorithms find the same optimal policy. But they differ in the frozen lake problem. This could be due to the difference between the two MDP in the number of states.
    d. **Impact of State Size**. The number of states also impacted the number of iterations required to find an answer.
2. **Q-Learning**

a. **Performance.** Overall, Q-learning showed an inferior performance compared to the previous algorithms inevitably due to the lack of a model.
b. **Exploration VS Exploitation.** For each MDP, I looked at the tradeoff from two different angles: epsilon value and decay function. For each epsilon starting value, I used a linear decay function as well as a non-linear one. Interestingly, in the case of the small MDP, the linearly decaying epsilon with the smaller starting value displayed the best performance. On the contrary, the non-linear decay function performed better in the case of the frozen lake problem.

Value Iteration

The threshold for VI's convergence can be defined in terms of gamma and epsilon. One goal is to show how the VI converges under different hyperparameters. So I used a fixed value for epsilon to isolate the impact of gamma on the algorithm.

$$threshold = \quad \times (1 -) /$$

where ε is the probability of random action in ε-greedy policy, and γ is the discount-rate parameter.

The candidate metrics that I used for convergence included

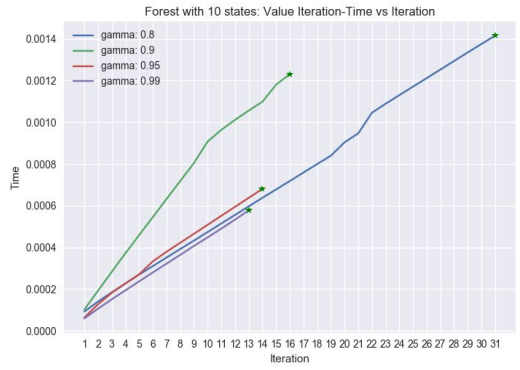
- V-diff (a.k.a delta): the sum of changes in the value-function (V) across all states
- V-avg: the mean of changes in the value-function (V) across all states
- V-sum: the sum of the values for all states
- V-max-diff: the largest change in the value-function among the states
- V-variation: the largest change in the value-function among the states minus the smalles change

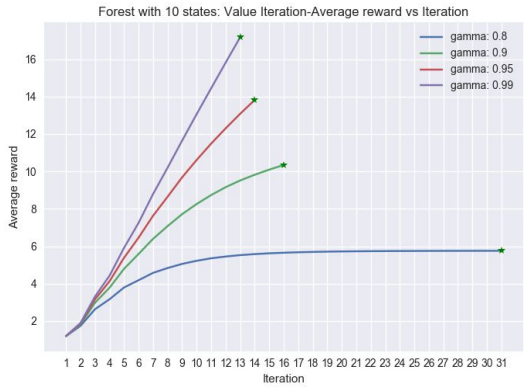
When I tried out these metrics on the forest management MDP, they more or less converged within the same range of iterations. Though, in the case of Frozen Lake, results were more consistent with V-variation as the convergence metric. This can be explained by the fact V-variation takes into account the effect of negative rewards for the intermediate steps as well as the holes in the grid world. I used both V-variation and V-avg to capture the trends in VI convergences.

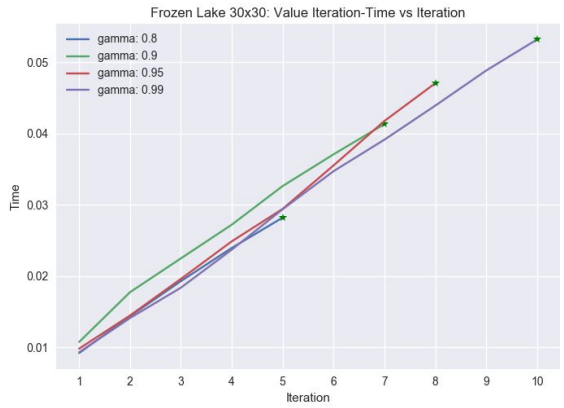
As depicted in the graphs, VI converges in both problems fairly quickly. A very interesting observation can be made with regard to the effect of gamma. In Forest Management problem, the VI with the lowest gamma converges the slowest, whereas, in Frozen Lake 30 x 30, the one highest gamma is the slowest. This stems from the difference between these problems in terms of the underlying structure. In the latter problem, the values of the states at the beginning are small negative rewards, so using a high value for gamma means we put more values on the long term gains. The former problem has a much smaller state space, which makes reaching a convergence with fewer iterations more likely.

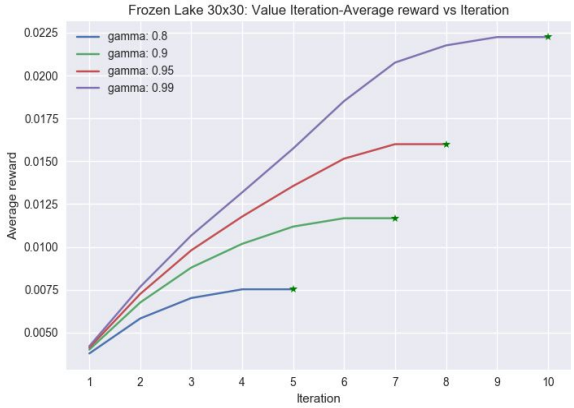
In terms of time, it appears that both problems display an almost linear curve. This may seem contradictory to our intuition as the VI algorithm uses three nested loops, but we need to take into consideration the fact that it takes only a few iterations for the VI to converge. Also, an

interesting observation is that gamma doesn't seem to heavily impact the slop of the execution time vs iterations.









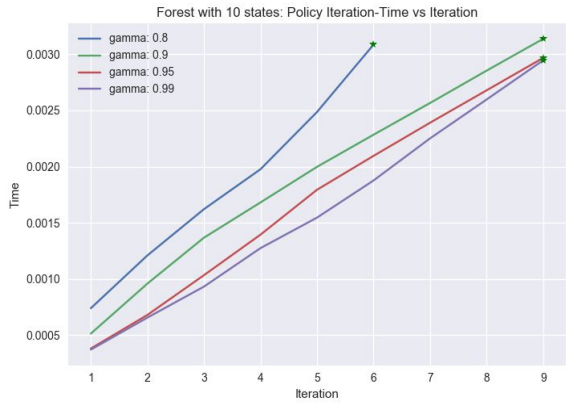
Policy Iteration

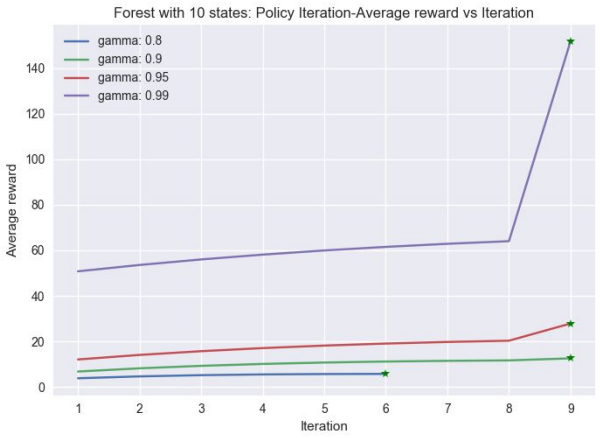
In addition to the convergence metrics explained earlier, I also looked at the number of changed actions at each iteration. Ultimately, I find that the benefit of looking at the same metrics outweighs using a different metric, as it facilitates comparing PI with VI.

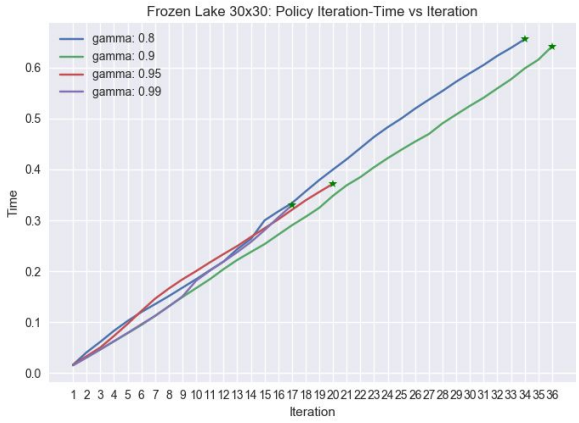
Contrary to VI, PI stops as soon as it finds the optimal policy. Overall, PI takes fewer iterations compared to VI. Though, since at every iteration PI solves a linear equation, each iteration may take longer than that of a VI. For example, when solving the forest management problem, it takes up to .6 unit of time while VI takes only up to .003.

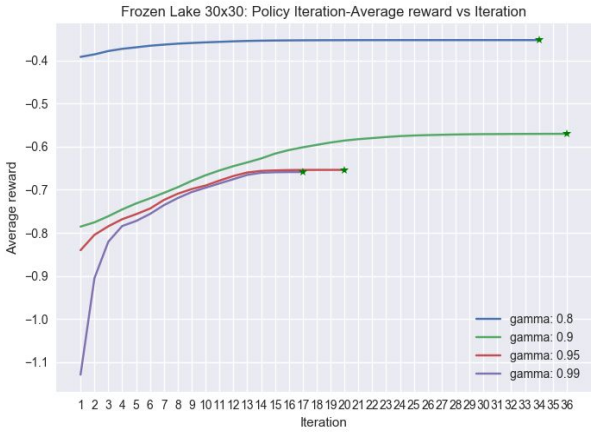
Comparing the PI's performance on the small MDP to the large one, one can see PI took fewer iterations when solving the former. It's worth noting that here the highest gamma took as many iterations as the next two values, whereas in the frozen lake problem the smallest gamma took more iterations. This is probably because, with the small future reward, the algorithm has less incentive to change an action.

I decided to not include the grid world maps as they take up too much space without providing much insight. Looking at the policies, one thing I find interesting is that PI and VI converge to different optimal policies for the frozen lake problem but they converge to the same policy on the small MDP.









Q-Learning

The q-learning algorithm responded really well to small Frozen Lake problems, but as I increased the size of the grid world, it started falling into holes. Despite my best efforts in hyperparameter tuning, I was not able to get good results. So I had to modify the rewards corresponding to different types of states. Once I made those changes, the q-learning started converging again. In other words, the q-learning algorithm is highly sensitive to the number of states. This is in contrast with the VI and the PI algorithms we examined earlier.

In order to perform the exploration-exploitation tradeoff analysis, I examined two aspects: the initial value of epsilon, and the decay function. My goal was to see how qlearning algorithms perform under each exploration-exploitation strategy.

For the forest problem, I chose initial epsilon values of .7 and .2. For each value, I used a linear as well as a non-linear decay function. The linear decay function simply deducts a fixed value at every iteration. The non-linear decay function decreases the epsilon logarithmically using the following equation

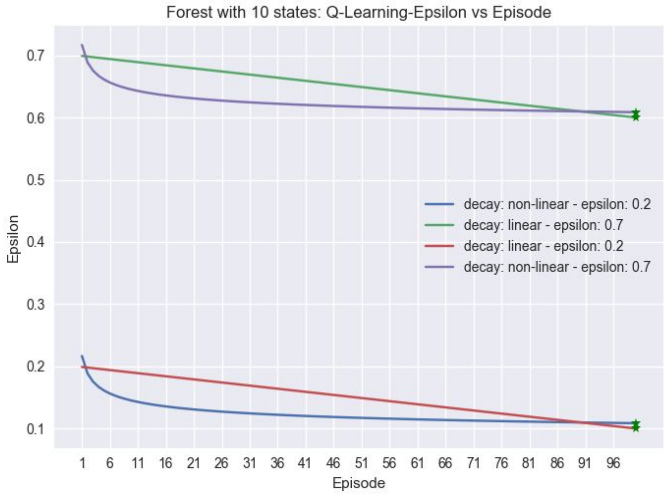
$$= 1/logn + c$$

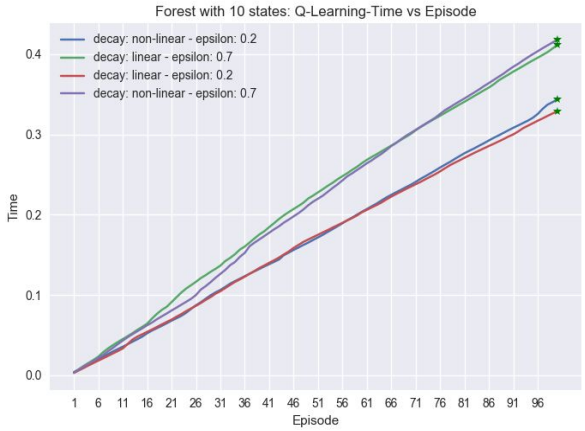
Where n is the iteration number and c is a constant.
Similarly, I used the following non-linear decay function for the Frozen-Lake problem

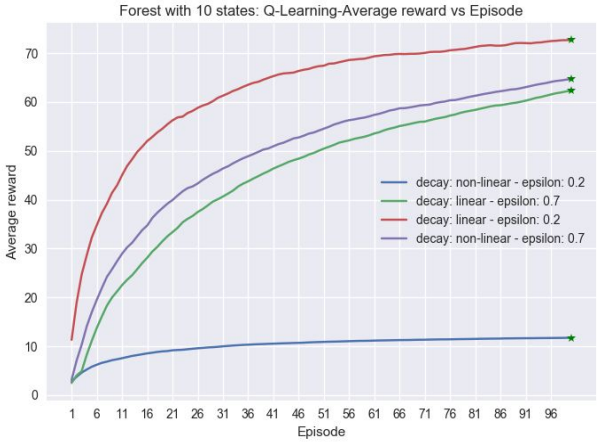
$$= \times d$$

Where d is the decay rate. As depicted in the Epsilon VS Episode graphs, the values are chosen in a way that for each pair starting at the same value, they also intercept at the end. That way we can isolate the effect of linearity/initial value for each pair.

Let's look at the forest management problem. While there is not a significant difference in terms of time, the obvious winner is the linearly decayed epsilon starting from the lower value (.2). This supports our understanding of the problem structure: for the smaller number of states, with a small number of possible actions, exploiting the current knowledge of the problem is more beneficial. But interestingly, the q-learning algorithm performs better when the rate of exploration is decreased linearly rather than exponentially. This is probably because even though the state space is small, there is still some uncertainty (catching fire). It's worth noting that the average reward curve looks smoother compared to the frozen lake, but that's just because in the mdptoolbox q-learning function I extracted the average over the steps for each episode





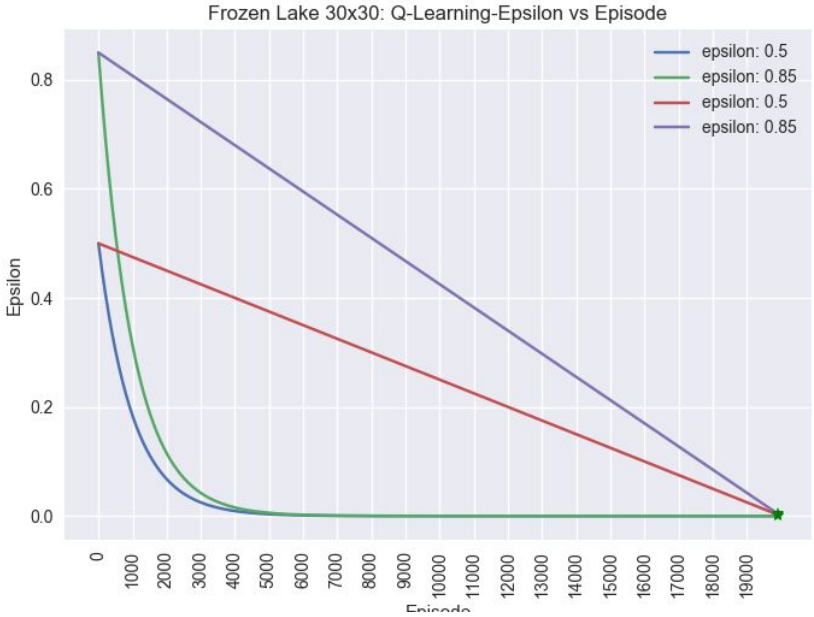


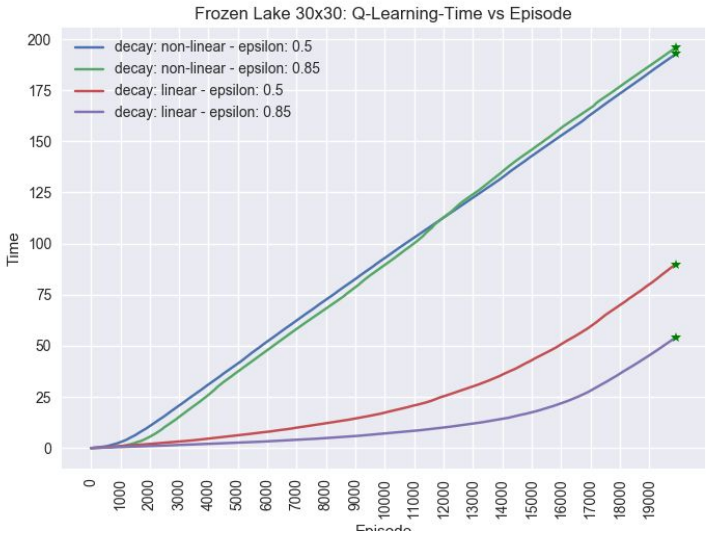
After experimenting with different epsilon initial values for the frozen lake problem, I chose .85 and .5 as these captured the underlying exploration-exploitation tradeoff.

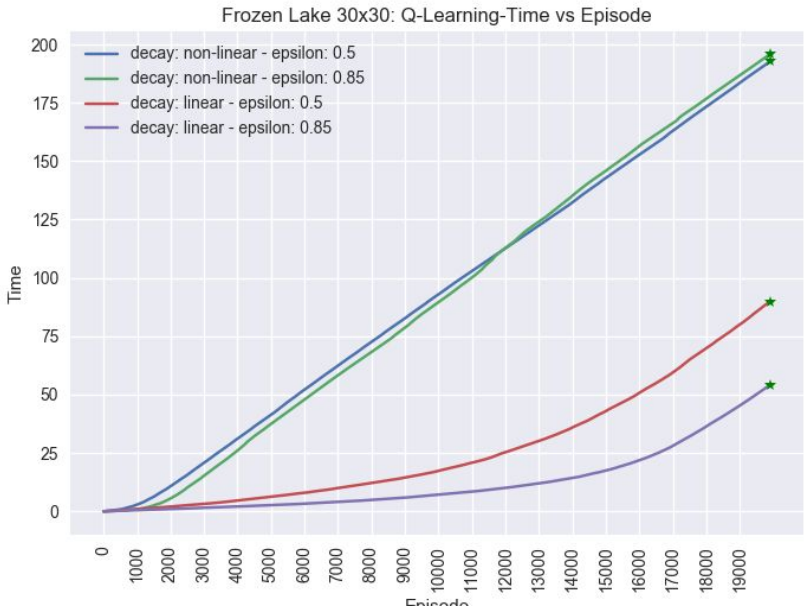
Although the non-linearly decayed algorithms took much longer to converge, they show a clear privilege when compared to the linearly decayed ones. The average reward curves for the latter are much noisier, which probably indicates falling into holes. This aligns with our understanding of the epsilon curves. The area under the epsilon curve corresponds with how slowly we move from exploration to exploitation. Comparing the area under curves of the linear function to the non-linear one, we can see that the linear functions are favoring exploration almost everywhere. Even when we compare the two linear functions we see that the one starting from .85 (therefore more exploration oriented) performs worse both in terms of average reward and accumulated discounted reward.

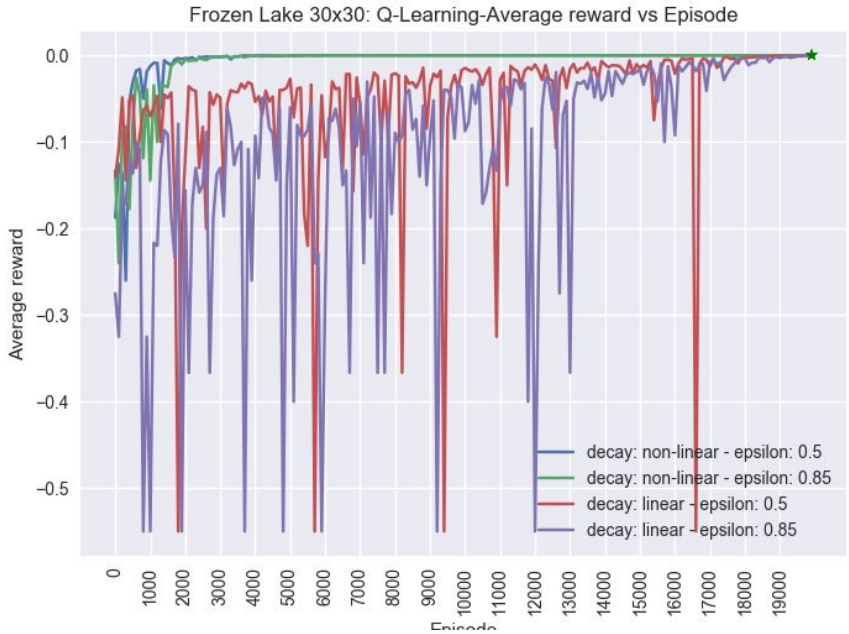
The q-learner that starts from 0.5 and is linearly decayed, in other words, the one with the least tendency to explore (thus, leans toward exploitation the most), has the best performance. Despite some spikes at the beginning, it converges the fastest and the highest reward. This could probably mean that other strategies had overly favored exploration, resulting in falling into holes more often.

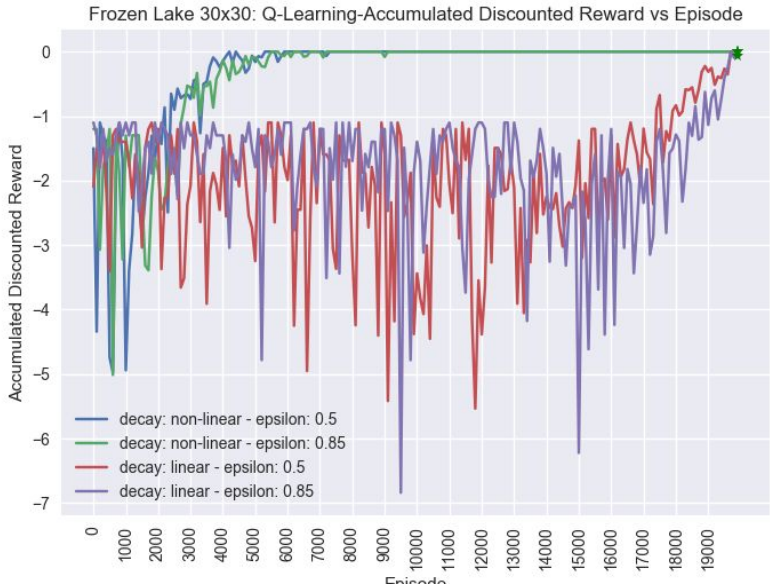
The reason for choosing average reward as well as accumulated discounted reward as metrics for q-learning was they captured the nuisances of the tradeoff better than other metrics.






Frozen Lake 30x30: Q-Learning-Time vs Episode
decay: non-linear - epsilon: 0.5
decay: non-linear - epsilon: 0.85
decay: linear - epsilon: 0.5
decay: linear - epsilon: 0.85
Time
Episode


Frozen Lake 30x30: Q-Learning-Average reward vs Episode
Average reward
decay: non-linear - epsilon: 0.5
decay: non-linear - epsilon: 0.85
decay: linear - epsilon: 0.5
decay: linear - epsilon: 0.85
Episode


Frozen Lake 30x30: Q-Learning-Accumulated Discounted Reward vs Episode
Accumulated Discounted Reward
0
−1
−2
−3
−4
−5
−6
−7
decay: non-linear - epsilon: 0.5
decay: non-linear - epsilon: 0.85
decay: linear - epsilon: 0.5
decay: linear - epsilon: 0.85
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000