Solving the Frozen Lake Problem using Markov Decision Process assumptions and Reinforcement Learning

Daniel Pak

***Keywords*— Value Iteration (VI), Policy Iteration (PI), Policy Gradient Optimization (PGO), Markov Decision Process (MDP), frozen lake problem, reinforcement learning**

I. INTRODUCTION

Markov Decision Process (MDP) is one version of Makov models where the state is affected by a control input and is completely observable. For MDP problems, reinforcement learning can be used to determine the ideal policy, which is essentially a look-up table for what action to take at a given state. Value iteration policy iteration, and policy gradient optimization are the three different types of reinforcement learning explored in this project. The details of the models will be discussed in the technical approach section.

II. PROBLEM FORMULATION

The MDP problem that we tried to solve is the frozen lake problem. The state space is discrete and defined on a grid map, with each grid representing one of the following: start position, frozen/solid spots, holes, and goal (Fig. 1 and 2). The main objective is to reach the goal from the start position without falling into a hole. Upon reaching the goal, the agent is given a reward of 1, and the episode of traveling terminates. If the agent ends up in a hole, the episode terminates and is given a reward of 0. The length of an episode is not limited, meaning the agent must keep traveling until it reaches a hole or the goal; therefore, it is an infinite horizon problem. The control is also discrete, defined as direction of traveling (up, down, left, right). The main twist to the problem is that a given control input is affected by noise - an agent trying to go straight may end up in other grids nearby (characterized by "slip" probabilities). As previously mentioned, the goal of this project is to figure out the ideal policy for a given frozen lake grid map and slip probabilities.

III. TECHNICAL APPROACH

A. Value Iteration

Value iteration is the simplest of the three reinforcement learning methods, where the general idea is to update, for every iteration, the "value" of each state with the equation below, and also choose the policy that maximizes each state value.

$$V_{k+1} = \max_{u \in \mathcal{U}(x)} \left[r(x,u) + \gamma \sum_{x'} p(x'|x,u) V_k(x') \right]$$

where V = state-value function, k = iteration number, x = state, u = control, γ = discount factor (set to 0.95), and r = reward. The reward is calculated as the expected reward for a given state and action pair, and thus $r(x,u) = \sum_{x'} p(x'|x,u) \cdot r(x')$. The discount factor is needed in this case to prevent value functions going to infinity, as the frozen lake problem is an infinite horizon problem.

One thing to note is that the value function is calculated starting from the goal and backwards towards the start position, because calculation of reward is much simpler when we only have to figure out one action at a time. The state-value functions were all initialized to 0, and calculated iteratively according to the value iteration equation with 20 iteration steps. At the end of each iteration, the ideal policy according to the current state-value function was calculated by taking the actions that give the best state values at each state. The exact equation is as follows:

$$\pi_k^*(x) = \arg\max_{u, \mathcal{U}(x)} \left[r(x,u) + \gamma \sum_{x'} p(x'|x,u) V_k(x') \right]$$

All results related to value iteration were obtained using the two provided equations.

B. Policy Iteration

Policy iteration is a more thorough version of value iteration, where the general idea is the same, but the value functions are calculated completely for a given policy. Instead of truncating the policy evaluation step after a single backup, it is calculated via linear programming in policy iteration algorithm, exploiting the fact that the state-value functions form a set of linear equations in this step. The equation for policy evaluation is as follows:

$$V^\pi = R^\pi + \gamma P^\pi$$
$$(I - \gamma P^\pi) V^\pi = R^\pi$$

where

$$R^\pi \in \mathbb{R}^{|\mathcal{X}|} \text{ with } R_i^\pi := r(i, \pi(i))$$
$$P^\pi \in \mathbb{R}^{\mathcal{X}x\mathcal{X}} \text{ with } P_{i,j}^\pi := p(x' = j | x = i, u = \pi(i))$$
$$V^\pi \in \mathbb{R}^{|\mathcal{X}|}$$

Using these equations and numpy's linalg.solve method, the value functions were calculated for each iteration step. Then, the policy that results gives the best value can be calculated using the same $\pi_k^*(x)$ equation as in value iteration. The reason we were able to use linear programming for policy iteration and not for value iteration is that the max function used to calculate the value functions in value iteration makes the process non-linear, whereas the value functions for policy iteration are calculated iteratively with linear operations.

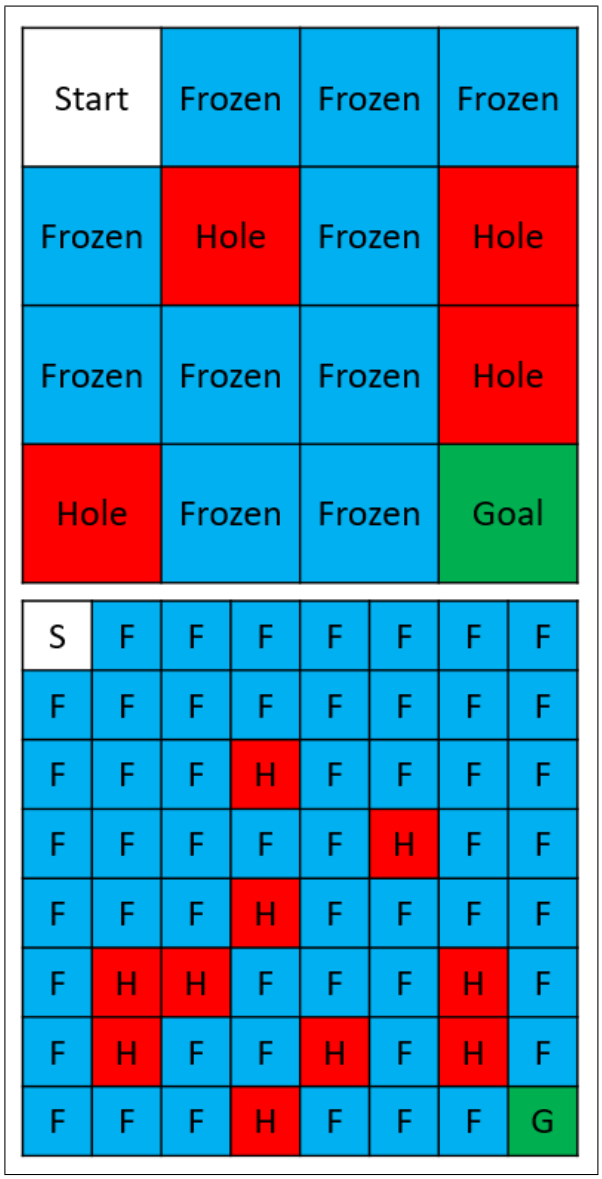

Fig. 1. 4x4 and 8x8 map of frozen lake problem

C. Policy Gradient Optimization

Since the value function has an expectation over all trajectories, it may be hard to evaluate in large action and control spaces. Therefore, instead of calculating value functions in advance, policy gradient method can be used to calculate a parametrized policy with estimated value functions. With the probability of action at a given state $\pi(x, u)$, trajectories are sampled and used for further optimization of the parameters. It relies on the idea of iteratively moving the policy parameters in the direction that increases the long term reward. This idea can be mathematically expressed as follows:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$$

where θ is the parameters defining the policy π.

Based on the policy gradient theorem with less variance:

$$\nabla J(\theta) = \mathbb{E}\big[\sum_{k=0}^{T-1} \nabla_\theta \log \pi(u_k|x_k, \theta) \cdot G_k^T(x_{k:T}, u_{k:T-1})\big]$$

where

$$G_k^T = \gamma^{T-k} r(x_T) + \sum_{k=t}^{T-1} \gamma^{k-t} r(x_k, u_k)$$

$$\pi(u|x, \theta) := \frac{e^{\theta(x,u)}}{\sum_a e^{\theta(x,a)}}$$

Since $\pi(u|x, \theta)$ is a scalar and ∇_θ is matrix gradient with $\theta \in \mathbb{R}^{|\mathcal{X}|x|\mathcal{U}|}$, I considered the derivative for each $\theta_{x,u}$ and combined them at the end. Also, θ is a measure of preference, but is not a probability measure (is not required to be greater than 0 or sum to 1), so the probability of action at a given state $\pi(x, u)$ is calculated using the softmax function as above.

$$\begin{aligned}
&\frac{\partial}{\partial \theta_{x,u'}} \log \frac{\exp(\theta_{x,u})}{\sum_a \exp(\theta_{x,a})} \\
&= \frac{\sum_a \exp(\theta_{x,a})}{\exp(\theta_{x,u})} \cdot \frac{0 - \exp(\theta_{x,u'}) \cdot \exp(\theta_{x,u})}{(\sum_a \exp(\theta_{x,a}))^2} \\
&= -\frac{\exp(\theta_{x,u'})}{\sum_a \exp(\theta_{x,a})}
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial}{\partial \theta_{x,u}} \log \frac{\exp(\theta_{x,u})}{\sum_a \exp(\theta_{x,a})} \\
&= \frac{\sum_a \exp(\theta_{x,a})}{\exp(\theta_{x,u})} \cdot \frac{\exp(\theta_{x,u})(\sum_a \exp(\theta_{x,a}) - \exp(\theta_{x,u}))}{(\sum_a \exp(\theta_{x,a}))^2} \\
&= \frac{\sum_a \exp(\theta_{x,a}) - \exp(\theta_{x,u})}{\sum_a \exp(\theta_{x,a})} \\
&= 1 - \frac{\exp(\theta_{x,u})}{\sum_a \exp(\theta_{x,a})}
\end{aligned}$$

$$\frac{\partial}{\partial \theta_{x',u}} \log \frac{\exp(\theta_{x,u})}{\sum_a \exp(\theta_{x,a})} = 0$$

Therefore, for a given time point, one row of ∇_θ was calculated for a given state, and were later combined with others to obtain the full gradient matrix for a given $\pi(x, u)$. The process described thus far is for a single trajectory. Since the estimation is done with multiple trajectories, the final gradient used for update is the average of gradients for all trajectories.

The process of sampling trajectories, calculating the gradient, and stepping in parameter space is repeated until the end of iteration numbers or a desired metric is met, such as perplexity or mean of episode reward.

IV. RESULTS

A. Value Iteration

For value iteration, the results can be explained with three figures (Fig. 2, 3, 4). The first is the frozen lake map labeled

with value and policy (Fig. 2). Only the final values are displayed in this report for better visual arrangements, but the progression of value and policy, listed in the appendix, can also be informative for understanding how the algorithm affects the result. Visually speaking, Fig. 2 intuitively makes sense in terms of value and policy - a grid has higher value when it is closer to the goal and farther away from holes, and the policy generally points towards grids with higher values. The progression looks like the grids touching the latest grid get updated on both policy and value until it reaches the start point, at which point the values keep getting updated with the same policies until the state-value function converges.

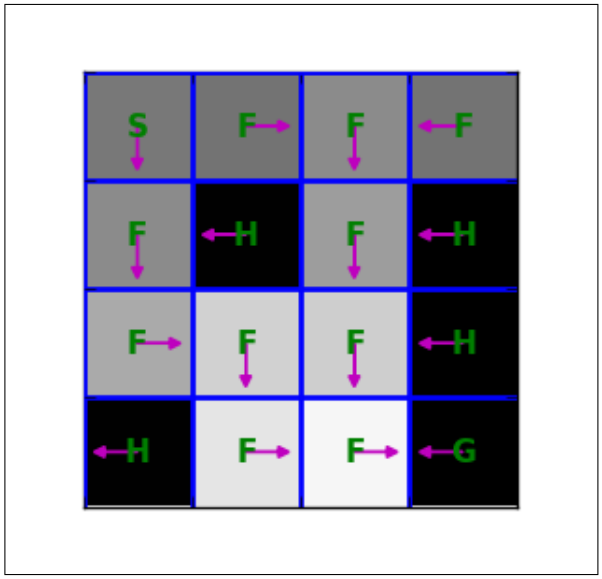

Fig. 2. Final state-value and policy described with shade of white and arrows, respectively (value iteration)

The convergence of state-value function and the progression of value iteration algorithm can be displayed in many different ways. The next two figures show how that convergence looks in graphs and numbers. Fig. 3 is the state values at different iteration steps. This graph matches the description earlier that some states are updated at later iteration steps, and also shows that the state values reach steady state before 20 iteration steps.

Fig. 4 shows a few different measures for observing the convergence of the algorithm. First is max—V-Vprev—, which indicates the amount of change in the state-value function for an iteration step. At low iteration numbers, this value will be high because state values are changing drastically from 0 to a value evaluated by the update equations. Conversely, this value should converge to 0 at high iteration numbers since values should reach a local optimum and not fluctuate with the given algorithm. Second is the number of changed actions for an iteration step, which should also converge to 0 with more iterations, since the policy should also be reaching one that gives a locally optimal state-value function. Third and last, the V[0] value indicates the value of state 0, or the starting point. This value is also a good indication of how close to convergence the algorithm is, since

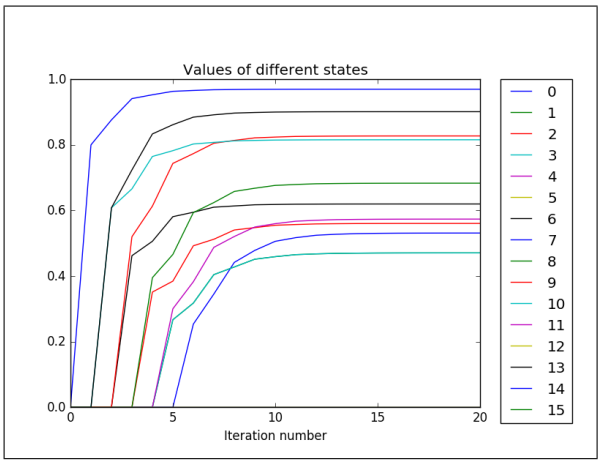

Fig. 3. State values for all states plotted over iteration number (value iteration)

all trajectories must end at the starting point, and thus no change in starting point's value most likely means no change in other state values as well.

Iteration	max|V-Vprev|	# chg actions	V[0]
1	0.80000	N/A	0.000
2	0.60800	1	0.000
3	0.51984	2	0.000
4	0.39508	2	0.000
5	0.30026	2	0.000
6	0.25355	2	0.254
7	0.10478	1	0.345
8	0.09657	0	0.442
9	0.03656	0	0.478
10	0.02772	0	0.506
11	0.01111	0	0.517
12	0.00735	0	0.524
13	0.00310	0	0.527
14	0.00190	0	0.529
15	0.00083	0	0.530
16	0.00049	0	0.531
17	0.00022	0	0.531
18	0.00013	0	0.531
19	0.00006	0	0.531
20	0.00003	0	0.531

Fig. 4. Other measures of convergence over iteration number (value iteration)

B. Policy Iteration

The results for policy iteration are formatted very similarly to those for value iteration. Fig. 5 is the equivalent of Fig. 2, except plotted with value functions and policies determined from policy iteration algorithms. They look identical and both intuitively make sense as described earlier, so it provides some evidence that the algorithm was performing accurately. However, based on the progression of the diagram (listed in the appendix), I could observe the difference between value and policy iteration methods. Unlike value iteration, policy iteration allowed for multiple states to change their values and policies, and the state values also converged much more quickly.

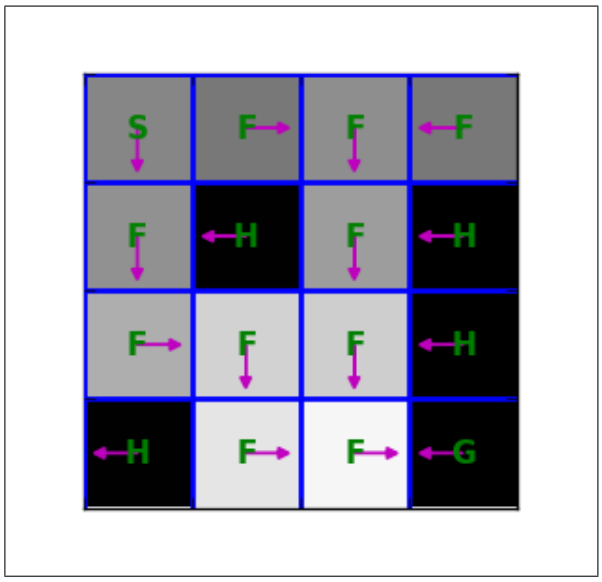

Fig. 5. Final state-value and policy described with shade of white and arrows, respectively (policy iteration)

The quick convergence is also explicitly visualized with Fig. 6. With higher iteration number, it makes sense that the state value increases for both value and policy iteration methods. With policy iteration, however, the convergence rate is consistently faster as shown in Fig. 6 and also other similar graphs in the appendix.

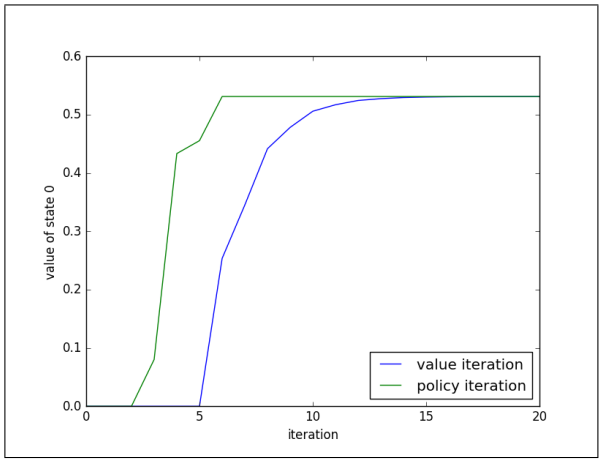

Fig. 6. Comparison of state value over iteration time between value and policy iteration methods

Fig. 7 is the state value for all states plotted over iteration time for policy iteration, analogous to Fig. 3 in value iteration. As already discussed with Fig. 5 and 6, Fig. 7 shows that the stead-state state-values are very similar between value and policy iteration methods, but the state values converge much more quickly for the policy iteration method.

Fig. 8 is the Fig. 4 equivalent of policy iteration method. This table confirms that the maximum number of changed actions per iteration is much higher for policy iteration, and thus allows for the value of state 0 (starting point) to converge to its optimum value at much lower iteration numbers. A new observation from this table is that the max change from previous to current state values is also higher for this algorithm, which makes sense because the policy evaluation method is completely solved using linear programming instead of being truncated after a single backup.

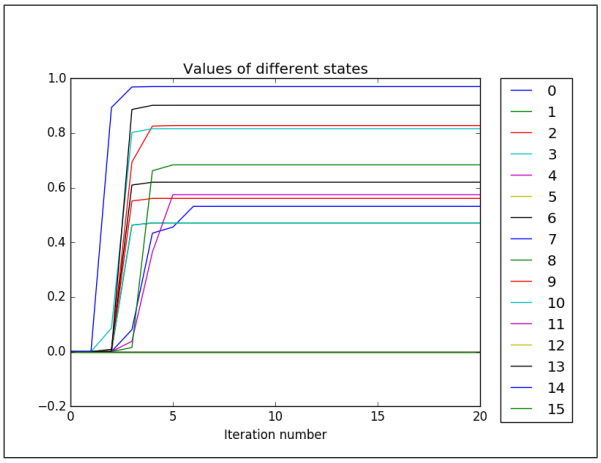

Fig. 7. State values for all states plotted over iteration number (policy iteration)

Iteration	max|V-Vprev|	# chg actions	V[0]
1	0.00000	N/A	-0.000
2	0.89296	1	0.000
3	0.88580	7	0.080
4	0.64660	4	0.433
5	0.20981	1	0.455
6	0.07573	1	0.531
7	0.00000	0	0.531
8	0.00000	0	0.531
9	0.00000	0	0.531
10	0.00000	0	0.531
11	0.00000	0	0.531
12	0.00000	0	0.531
13	0.00000	0	0.531
14	0.00000	0	0.531
15	0.00000	0	0.531
16	0.00000	0	0.531
17	0.00000	0	0.531
18	0.00000	0	0.531
19	0.00000	0	0.531
20	0.00000	0	0.531

Fig. 8. Other measures of convergence over iteration number (policy iteration)

C. Policy Gradient Optimization

The results for policy gradient optimization are presented with the Fig. 9-12. First is the episode reward (Fig. 9), which indicates the mean of rewards collected from sampled episodes with the policy calculated with PGO. The graph indicates that this value reaches steady state value pretty quickly (before 100 iteration steps), and also that it is around 0.8, which is pretty good considering the only reward for the entire map is 1 at the goal.

Fig. 10 is the mean episode length with trajectories sampled according to the policy at a given iteration time.

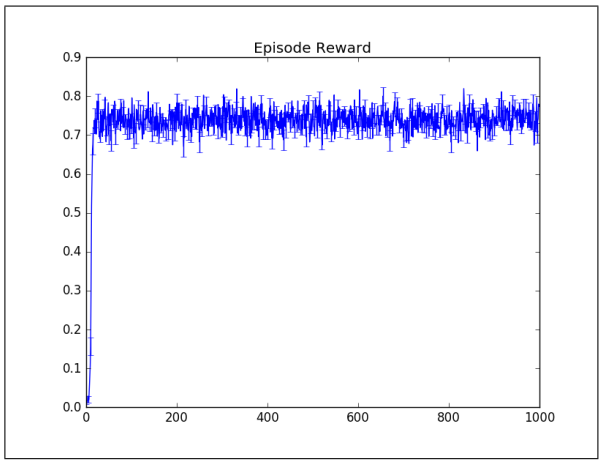

Fig. 9. Mean episode reward plotted over iteration time for 4x4 map

Similar to the mean episode reward, this value reaches steady state quickly, which is good indication that PGO is working properly. Also, the steady state value of around 6.5 makes sense with the map and noise conditions since the agent should be able to reach the goal with 6 moves with no noise.

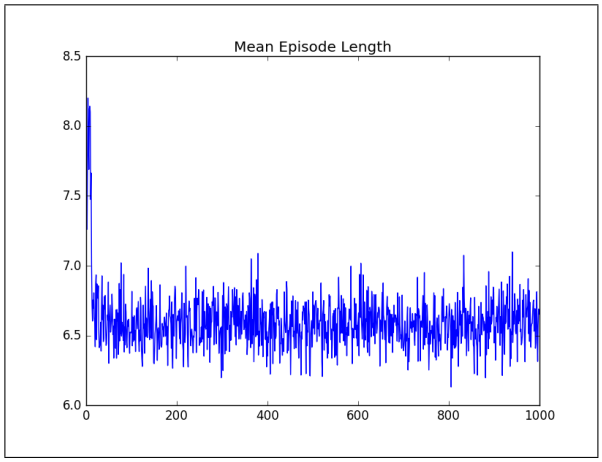

Fig. 10. Mean episode length plotted over iteration time for 4x4 map

Next is the mean KL divergence between old and new policies, indicating something similar to the ”number of actions changed” from value and policy iteration methods (Fig. 11). It measures the amount of change between old and new policies with KL divergence since the policies are now represented by probability distributions, and KL divergence can serve as a measure of similarity between two probability distributions. Concurring with Fig. 9 and 10, the KL divergence is greatest before 100 iterations, and seems like the KL divergence is maximum when the slopes of the episode reward and length are greatest. This makes sense because that is when there is the most amount of change in the policies, and thus more change in other measures as well.

Lastly, perplexity indicates the certainty of the policy at a given iteration time point. Since at iteration 0, it has no idea which policy is best, perplexity is set at 4 (presumably meaning that the policy can be any of the 4 actions at any given state). The fact that this value goes down to 1 indicates

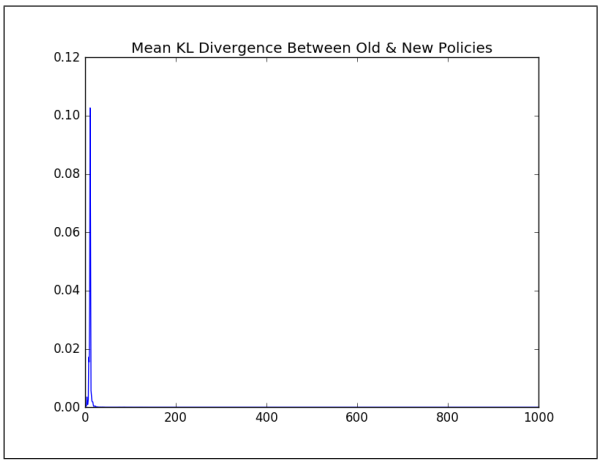

Fig. 11. Mean KL divergence between old and new policies plotted over iteration time for 4x4 map

that the algorithm is figuring out the distributions of actions for a given state that is basically only one action per state (very high probability measure for one action and very low for the other 3). This also indicates that PGO is working properly and the policies are being converged to a local optimum. It also has a high slope at similar points at episode reward and length, and reaches steady state before iteration 100.

These plots were created with 1000 iteration time steps so that they can be compared directly with the results for 8x8 map, which are listed in the appendix.

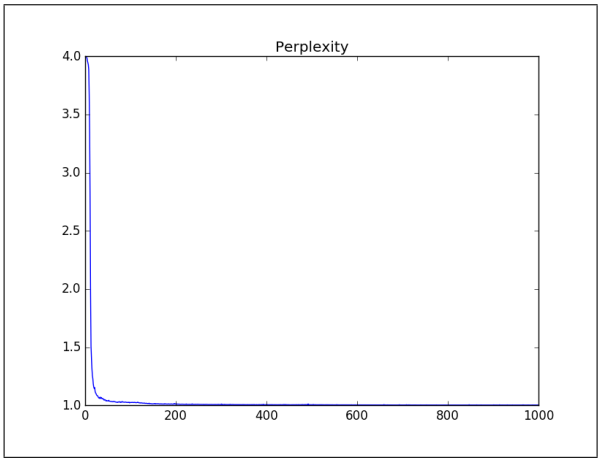

Fig. 12. Perplexity plotted over iteration time for 4x4 map

V. DISCUSSION

The results for value and policy iterations were discussed thoroughly in the Results section, with rationale for each plot related to the equations in the Technical Approach section. Since this project was a lot more structured than previous ones, there was not much room to explore in terms of changing parameters or method of approaching the problem. I simply followed the equations and directions, and was able to achieve what was expected of the algorithms.

Some things that I had change for policy gradient optimization for the 8x8 map were step size for gradient update (increased from the original value of 100 to 200), number of iterations for full convergence (from 100 to 1000), and the horizon value for episode simulation (from 10 to 50). These changes ensured that I had good convergence for the policy parameters, and also see the final result play out on the map for a single simulation of trajectory from start to finish. Also, the 4x4 and 8x8 maps had very similar characteristics for the four plots, such as the episode reward reaching a steady state value of around 0.8, the KL divergence being greatest at the steepest slopes of episode and length graphs, the perplexity moving from 4 to 1. The only differences were that the mean episode length were different (around 16 instead of 6.5, simply because there are more grids to be passed to reach the goal), and the convergence happened much later than for the 4x4 map. For direct comparison, the step size for 4x4 and 8x8 maps were identical (both at 200), and should not have been a factor in the faster convergence. Presumably, the slower convergence was entirely due to the increased number of failures due to a larger map and more places to explore/fall.

VI. REFERENCES

All information was acquired from Unviersity of Pennsylvania's ESE650 lectures, discussions on Piazza, official websites of Python and the respective libraries.