Assignment 4

Rodrigo De Luna Lara

November 26, 2017

Introduction

The two problems used in this assignment are variations of the FrozenLake environment from OpenAI gym, described as follows:

> Winter is here. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. (...) However, the ice is slippery, so you won't always move in the direction you intend.

One problem has 16 states (4x4 grid), the second one has 64 (8x8 grid). Figure 1 shows the scenarios and Table 1 shows the nomenclature for the states and actions. This particular problem is interesting because of its stochastic nature. A decision to move in a direction can result in moving in a different one at random so this problem is non-deterministic, making it interesting to explore in the context of the different algorithms.

Table 1: MDP States and Actions

States		Actions	
S	Starting Position	0	Left
G	Goal	1	Down
F	Frozen Surface	2	Right
H	Hole in the Ice	3	Up

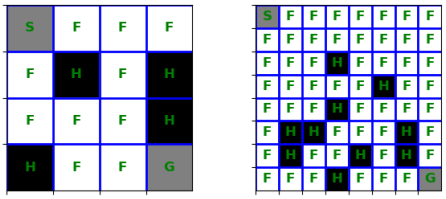

Figure 1: 4x4 (left) and 8x8 (right) Frozen Lake Scenarios.

The rewards for this environment are also very punishing, only having 2 possible values. If the agent reaches the goal the reward is 1, otherwise it is 0. There are no rewards for getting close to the solution or for reaching the goal sooner. The agent has access to binary rewards only, which is interesting in the context of learning, as it can only fail or succeed, there are no intermediate rewards. The horizon is infinite, it doesn't matter how long it takes as long as the agent succeeds.

For each of the problems (4x4 and 8x8) three algorithms are run:

- Value Iteration and Policy Iteration
- Q-Learning:
  - Epsilon greedy
  - Noisy action selection

Value Iteration

The implementation of this algorithm runs until a tolerance in the innovation of the values is reached (with a value of 1×10^{-10}) or the maximum number of iterations is reached. The algorithm updates the values of $Q^*(s,a)$ with $\sum p(s'|s,a)\left(R(s,a) + \gamma V^*(s')\right)$ and $V(s)$ with $\max(Q^*(s,a))$ to calculate $V^*(s)$ at each iteration. γ was fixed to 0.99. Other values of γ were tested, the only significant difference being how quickly the values propagated and converged (the higher the γ the faster the values converged).

The innovation is calculated as the sum of the absolute differences between the value matrices ($\sum |V^*(s) - V^*(s')|$). The mean $E[V^*(s)]$ is also calculated at each iteration, as well as the number of actions that were updated in it and the time it took. Also, besides updating the values, the policy is updated at each iteration. Figure 2 and Figure 3 show the evolution of values and policies at different iterations for both problems. In both cases it can be seen that they are propagating outwards from the goal state.

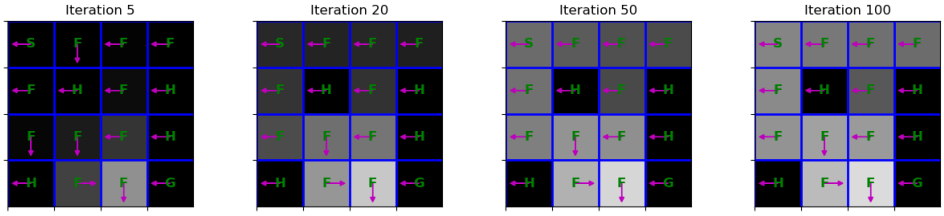

Figure 2: 4x4 Value Iteration Policy and Values.

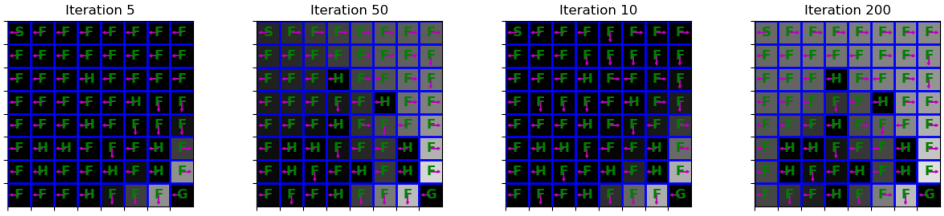

Figure 3: 8x8 Value Iteration Policy and Values.

Figure 4 shows the different metrics for the Value Iteration for both problems. The smaller 4x4 problem has a smooth convergence and reaches a higher mean value with the same tolerance as the 8x8 problem. The 8x8 problem has a bump in the innovations, it doesn't decay monotonically as the 4x4 problem does. Value iteration seems to propagate higher values towards the states in the grid that make up the more optimal route to the goal state. For example in the 8x8 problem, the top-right corner of the grid has higher values, and that region corresponds to the most optimal route, you just have to go straigh right and straight down from the start to reach the goal in this case. The 4x4 problem has less options available to reach the goal state, there are only 2 paths at most, and it is able to propagate higher values towards them. The mean of the values could then be used as a way of knowing how much of the space an agent needs to explore to reach the goal.

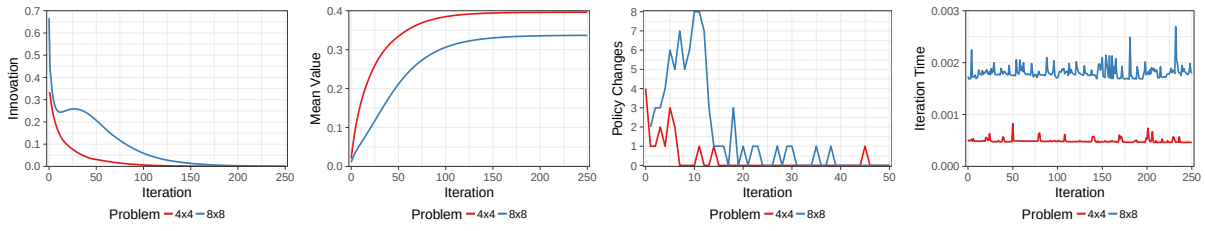

Figure 4: Metrics for Value Iteration.

Regarding the changes in the policy (action changes), the first few iterations have the most changes in policy. The policy converges a lot sooner than the values, after 50 iterations there are barely any changes in the policy. Surprisingly, the size of the problem doesn't really affect how quickly the actions stop changing, just how many changes there are in the policy (which is expected given the number of states). Finally, the time for each iteration for both problems is insignificant, even if the time for the 8x8 problem is approximately 4 times higher than for the 4x4 (the same factor of number of states), it is still in the order of milliseconds. Value iteration shows to be computationally cheap.

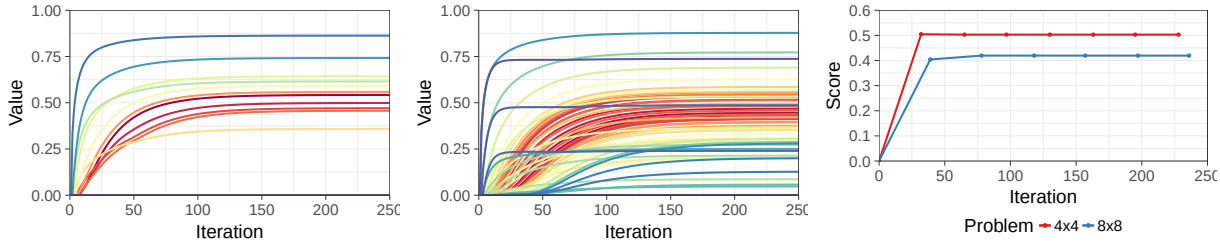

Figure 5: Change in Individual Values for 4x4 (left) and 8x8 (center). Mean Score Over 100 Episodes (right).

Figure 5 shows the individual values for each state in the problem, as the policy change plot shows most of them converge around 50 iterations. There are some states that converge quickier than others, most likely the neighboring states to the goal state, from which the values propagate as seen in Figure 2 and 3. The last plot in Figure 5 shows the mean reward for 100 episode runs for the first ~200 iterations. After ~50 iterations there is no significant change in the mean reward for both problems; again it converges sooner than the values as the policy stops changing.

Animations for the whole Value Iteration process for each problem were generated and can be found in the accompanying files (4x4 and 4x4)

Policy Iteration

Policy iteration starts by initializing a random policy, an iteratively computes the values for the policy $V^{\pi}(s)$ by solving the corresponding set of linear equations in the state value function: $\sum P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$, the results are then fed to the state-action value function $Q^{\pi}(s, a) = \sum P(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')]$ to calculate the new optimal policy at each iteration. The same metrics as Value Iteration are calculated at each iteration to compare performance between both.

Figure 6 and Figure 7 shows the values and policies at different iterations for both problems. It can be seen that at a fraction of the iterations the values and policies are very similar to Value Iteration when Policy Iteration converges (with the same tolerance).

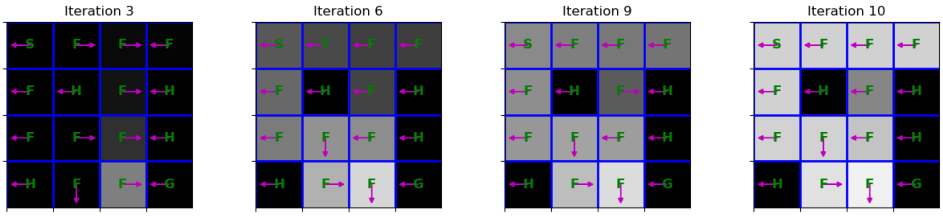

Figure 6: 4x4 Policy Iteration Policy and Values.

The values also propagate from the goal state, although at a much faster rate. The solutions are extremely similar however, so they both converge to essentially the same solution. The propagation of the policies and how they affect the values from the goal state also show that these algorithms (both Value and Policy Iteration) seek out the path of least resistance (the optimal path) in a very logical way.

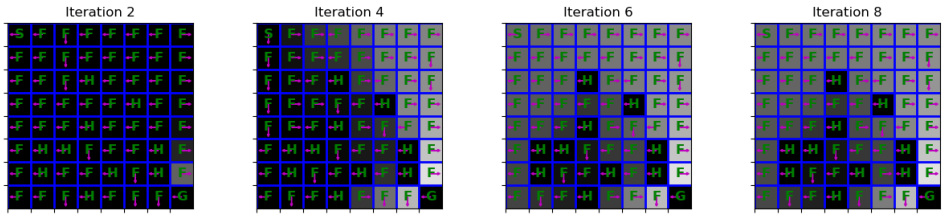

Figure 7: 8x8 Policy Iteration Policy and Values.

Figure 8 shows the evaluation of the metrics for Policy Iteration. There is in general a less smooth behavior than Value Iteration. Policy iteration converges more quickly albeit less smoothly, this is evidenced by the large number of policy changes per iteration and the magnitude of the innovations. The differences between both problems (4x4 and 8x8) are similar to Value Iteration, except for the iteration time. In Value Iteration both problems had negligible times and in similar orders of magnitude. However, Policy Iteration shows to be more computationally expensive and sensitive to the number of states. The 4x4 problem is within the same magnitude in time, but the 8x8 problem is 10-60 times larger and is not constant as with Value Iteration.

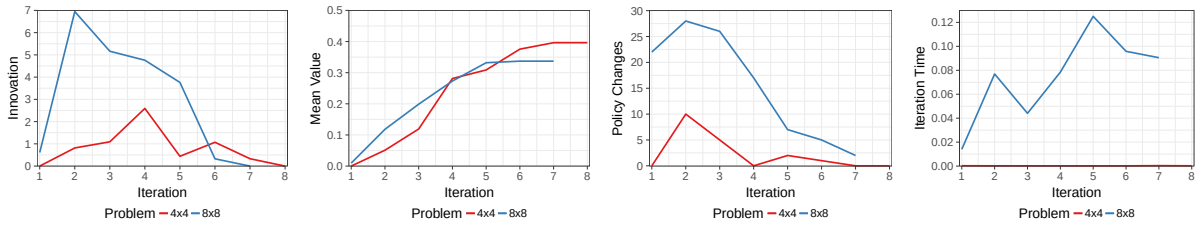

Figure 8: Metrics for Policy Iteration.

The time per iteration in Value Iteration remained nearly constant, while in this case it seems to increase with each iteration. Policy iteration converges in significantly fewer iterations but with a higher associated computational cost, several orders of magnitude higher (at least for the 8x8 problem). Value iteration is less computationally expensive and less computationally efficient while Policy iteration is more computationally efficient and more computationally expensive.

Figure 9 shows the evolution of values for individual states for Policy Iteration, as well as the mean reward for 100 episodes for each iteration for both problems. In less than 10 iterations both problems converge, although in a more jagged way with respect to Value Iteration. Both algorithms converge to a good mean score in a relatively low time, but they require prior knowledge on the structure of the MDP. It's like a student being given all the formulas required for a simple math test and he just needs to find the answers. The agent has at its disposal the state transition and reward models, so it can quickly determine what the best policy is.

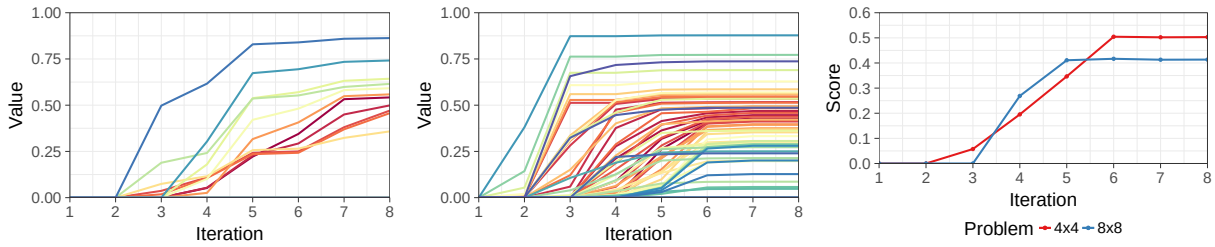

Figure 9: Change in Individual Values for 4x4 (left) and 8x8 (center). Mean Score Over 100 Episodes (right).

Both algorithms apply offline planning, they are model-based learners, they are not agnostic to the MDP as other model-free algorithms that must learn from experience. Value and Policy iteration are more akin to supervised learning, while model-free learners are more akin knowledge-based AI.

Animations for Policy Iteration were also created (4x4 and 8x8). In contrast with the animations for Value Iteration they converge much faster with higher number of policy changes per iteration, as discussed before.

Q-Learning

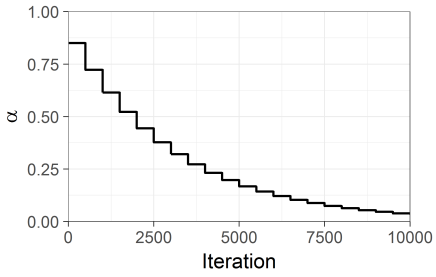

Figure 10: Learning Rate versus Iterations

For Q-Learning 2 different methods of action selection are implemented, to test different strategies of explotaition versus exploration in the context of both of the problems and their varying difficulty and size:

1. ε-greedy selection: the agent will either select a random action to explore or with probability $1-\varepsilon$ will select an action based on the current Q-values. In this implementation the value of ε is fixed, but several values are tested, including $\varepsilon \in [0.05, 0.075, 0.1, 0.125, 0.15, 0.25, 0.375, 0.5, 0.75, 0.90]$
2. Noisy selection: an action is selected with $\text{argmax}\left(Q(s) + \frac{\text{X}\sim\text{N}(\mu,\sigma^2)}{i+1}\right)$ where i is the current iteration or episode.

The discount rate is also tested at several values, $\gamma \in [0.95, 0.96, 0.97, 0.98, 0.99]$. The learning rate is set to decay on each iteration, following $\max\left(\alpha_{\min}, \alpha_{\max} * 0.85^{\text{floor}(i/500)}\right)$, with $\alpha_{\max} = 0.85$ and $\alpha_{\min} = 0.05$ which follows the sequence of values seen in Figure 10

The array of Q-values is initialized to 0 at each experiment (for each γ and ε). The actions are determined by the previous approaches and then the Q-values are updated on each step of the episode with the formula $Q(s,a) \mathrel{+}= \alpha(R(s,a) + \gamma \max(Q(s')) - Q(s,a))$. For each γ and ϵ the Q-learner runs for 10000 episodes with a max number of steps of 200 (as per the properties of the environment in OpenAI gym). The rewards are then smoothed using generalized additive models for better visualization and easier performance comparison.

Figure 11 shows the results for the ε-greedy Q-learner for the 4x4 problem. Across all plots it can be seen that high values of ε result in lower rewards for the agent. With high ε the probability $1-\varepsilon$ is very low, and so the agent tends to explore more, most likely falling in holes more often because of this.

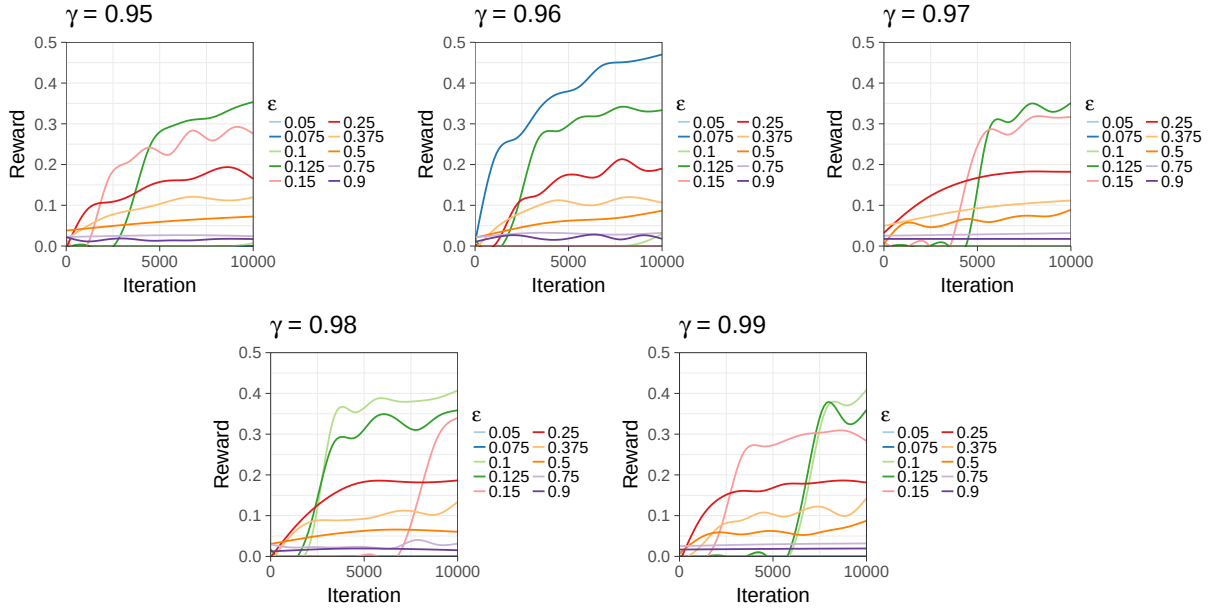

Figure 11: Reward for 4x4 Problem with Varying Discount Rates

Given that the grid is small, there is not much room to maneuver and the slippery nature of the "ice" in the problem (the stochasticity), an agent that explores more falls more often in the holes than one that exploits more based on the Q-values. So in this small lake, a more conservative agent that doesn't explore much is more successful, given that there are not a lot of available paths to the goal state. 25% of the states are end states, and they are placed in such a way that the path to the goal is not very direct, it is more difficult for the agent to explore because of this. In most positions of the grid, there is a failure state in the immediate vecinity, hence why too much exploration is negative.

The best agent is the one with $\gamma = 0.96$ and $\varepsilon = 0.075$. The effect of the discount in the agent is that it tends to take more iterations for the agent to start learning the higher it is. So in this problem a moderate look-ahead is preferred with low exploration. Which makes sense given the configuration of the grid. The number of states is unhelpful in this environment, as it requires a very precise movement without much room for error; therefore, the agent is much more conservative.

Figure 12 shows the results for ε-greedy Q-Learner for the 8x8 problem for varying γ and ϵ. The plots show that the agent is barely able to learn something in the same number of episodes than the 4x4 problem, and interestingly only with very high values of ε, in complete contrast with the smaller problem.

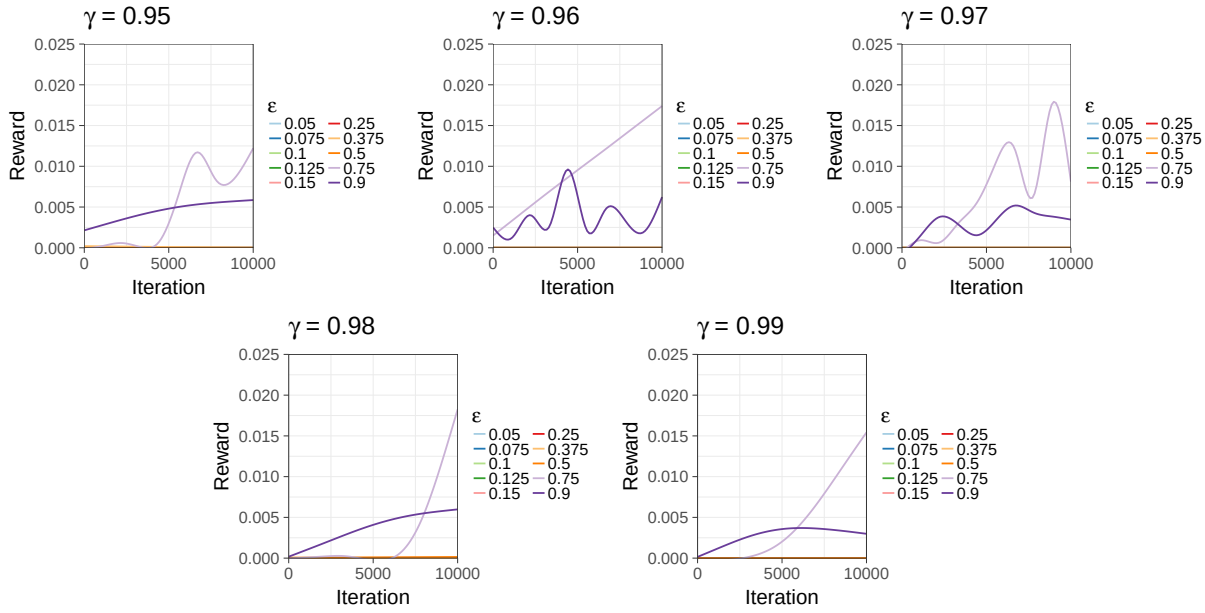

Figure 12: Reward for 8x8 Problem with Varying Discount Rates

This shows that for this larger problem, a very conservative agent that doesn't explore much never reaches the goal state. In this problem, only ~15% of the states are failure states, and there is a very direct route all the way right and all the way down that doesn't have many obstacles in the way. In contrast with the tighter 4x4 grid which required the agent to move carefully without exploring much, in this case it requires a lot of exploration or it is unable to ever find the goal state.

It is also evident that the scores are significantly lower than with the 4x4 problem. Given that both problems run the same number of episodes, it can be inferred that for problems with more states the agent requires more episodes as well to be able to learn more. We could make an analogy with a student, if he only has to study the content of a single lecture, he requires less time than if he were to study the content of 8 lectures. On the larger problem (with more lectures), he would also need to focus more on the general ideas and less in the details, he would need to spend more time exploring the content instead of exploiting every detail in them, just like the agent in this problem.

The difference between the 4x4 and 8x8 problems show that smaller isn't necessarily easier, one could have an infinitely large grid without any obstacles and it would be easier for an agent to reach the goal state than a small grid world filled with obstacles and just a few available paths. However, even if a larger problem

could be easier, it still requires more computations for the agent to learn, given the math involved. This is evident in the results for Policy Iteration, the 8x8 grid required 4 more times to reach the optimal policy, even if it's an easier grid world as per the configuration of the states.

Figure 13 shows the results for both problems for the Q-Learner with noisy action selection, with varying discount rates. The results are significantly better than with the ε-greedy approach, but they seem to be overfitting quickly. The effect of the discount rate varies with the problem; for the 4x4 problem a lower discount rate is better, while in the 8x8 problem the center values yield the highest rewards. Both problems show that a higher discount rate results in lower scores and slower convergence rates, with values < 1 being preferrable.

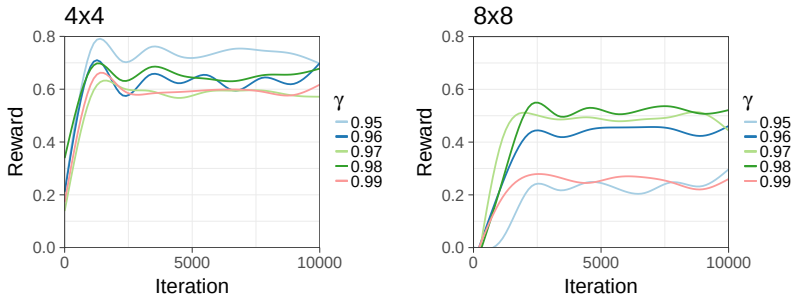

Figure 13: Reward for Varying Discount Rates (non-greedy)

The results also show that there is no one single value of γ that can be used for all problems, it has to be treated as a hyperparameter. The same applies to ε, it needs to be treated as hyperparameter as the structure of each problem will favor more exploration or exploitation, as shown with both problems here.

Even if the Q-Learner with noisy action selection has better overall accuracy, the ε-greedy approach seems to be better with basis in good practices in Machine Learning. The noisy approach seems to quickly overfit, while the opposite shows a clear trend of improvement over the episode span, even if there are some oscillations in the smoothed function. Over an infinite number of episodes the ε-greedy approach would (at least in theory) have better results than the noisy approach. These two approaches have different tradeoffs, the noisy approach gives a quick solution that is also quick to stop improving, while the ε-greeedy approach is slower but has steady improvement.

Figure 14 shows the distribution of actions for both approaches for both problems. Both problems seem to have a high preference to go left, which is particularly interesting as the goal state is always to the right of the starting state. This could be explained by two things: either the stochasticity of the problem means that most of the times you want to go left you actually go right, or the selection of the actions is ill-defined and biased to choose this particular action.

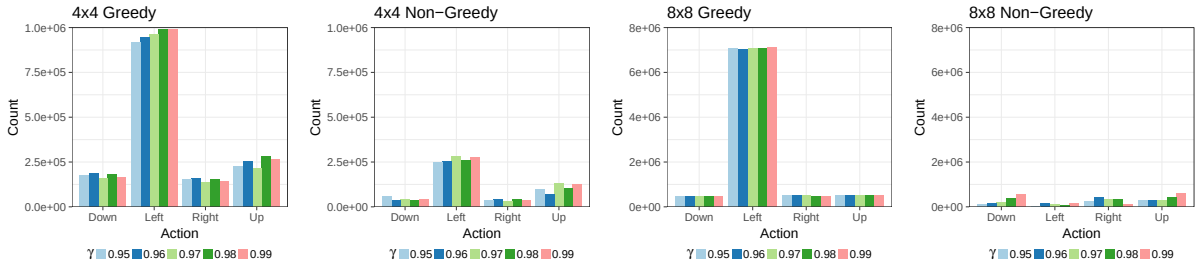

Figure 14: Distribution of Actions

The value of γ doesn't seem to have any significant effect in the choice of actions, they are pretty uniform across the different values of γ for each action. It can also be seen that the noisy Q-Learner tends to have more uniform distribution among the different actions, in particular of the case of the 8x8 problem. It also has significantly less actions in both problems, which are indicative that the agent is reaching an ending state (either falling into a hole or reaching the goal) sooner.

In the context of overfitting that was seen before, the noisy Q-Learner may be discovering a sequence of actions that reaches the goal state most of the time, and trying to improve upon it on each episode. That may explain why it has less actions. The ε-greedy approach may also be taking much more time exploring before committing to an adequate policy.

Figure 15 and Figure 16 show the individual distributions of steps per episode for the different values of ε for each value of γ.

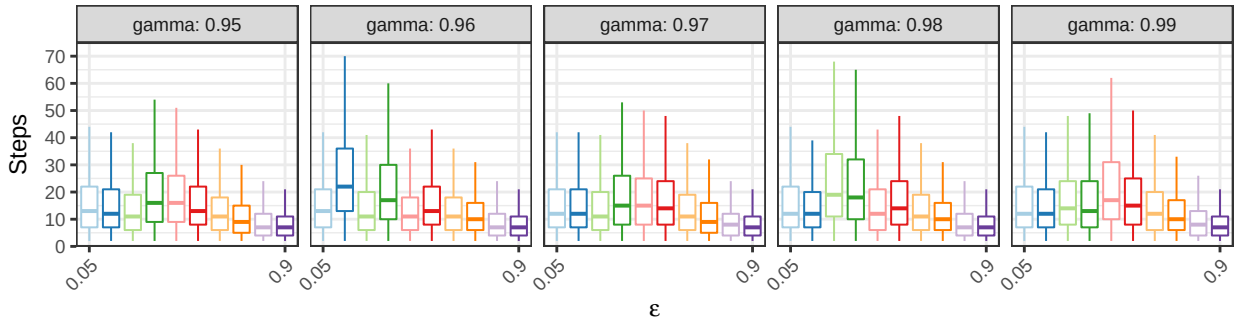

Figure 15: Distribution of Steps in Episodes for 4x4 Problem (Greedy)

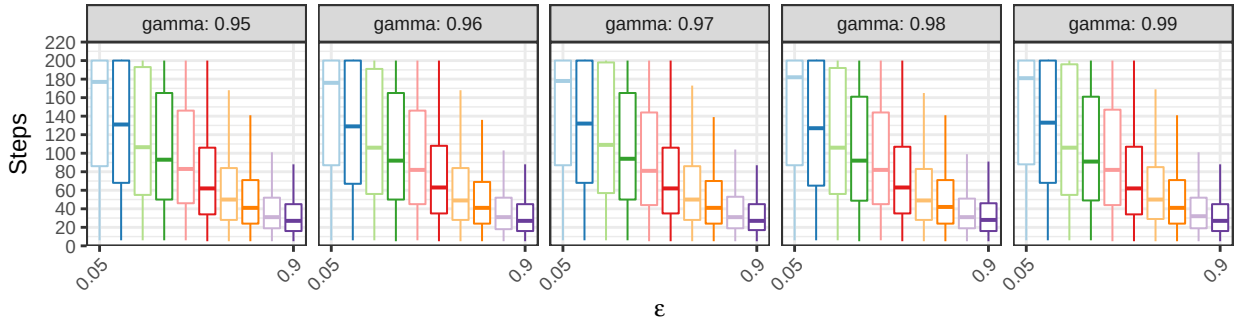

Figure 16: Distribution of Steps in Episodes for 8x8 Problem (Greedy)

For the 4x4 problem the boxplots show that when exploration is preferred (high ε) there are less steps in the episode, most likely because it falls more quickly in a hole due to the reasons mentioned before (small grid, stochasticity, etc). When exploitation is preferred (low ε) the agent takes more steps but does so in a more controlled way, using the knowledge it gathered so far in the Q-values to advance safely through the lake.

For the 8x8 problem, high values of ε result in fewer steps, and considering that the two last values (0.75 and 0.9) were the only ones with learning, and how as you move left in the axis for ε the number of steps get consistently higher and higher, it can be inferred that the agent spends too much time wandering around the lake instead of exploring more aggressively to reach the goal.

Both sets of plots present cases in which exploration is better and cases in which exploitation is better, and also why hyperparameter optimization is required when using model-free learning. Like the No Free Lunch Theorem implies, there is no magic algorithm which performs well in all problems. Even if both problems here are from the same universe and describe the same scenario they require different learning strategies for an AI agent to be successful.

Figure 17 shows the distribution of steps for each value of γ for the noisy Q-Learner. In this case there is not such a big difference between the median number of steps as the discount rate increases. In fact, in the case of the 4x4 problem the distributions are pretty similar, the median barely changes between discount rates. Coupled with the results in Figure 13, it seems that for this Q-Learning approach and problem, the discount factor has a low impact on the results.

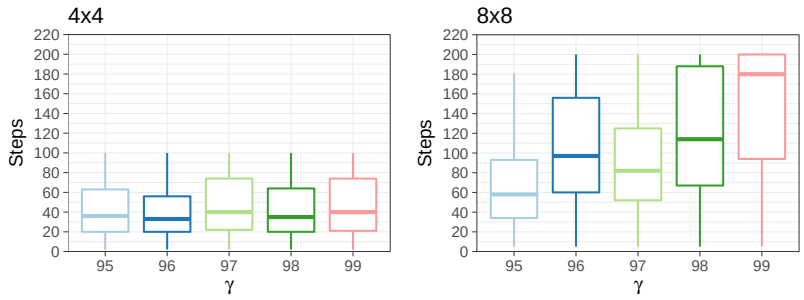

Figure 17: Distribution of Steps in Episodes (Non-Greedy)

However, in the 8x8 problem the discount factor seems to have a larger impact. The discount factor represents how important future events are considered by the learner. A discount factor of 0 would mean that the learner doesn't care about the future at all, while a higher discount factor would make the rewards propagate further through time. The size of the problem (number of states) is significant in this sense in that the horizon of rewards is much larger because there are more possible states. Then in a smaller problem the discount factor may not be as impactful as in a very large problem.

Figure 18 and Figure 19 show selected Q-Tables for the greedy approach (rows are states, columns are actions). In the 4x4 problem the tables only have large values for certain actions in some states only, indicating the tight path to reach the goal, also when $\varepsilon \in [0.05, 0.1, 0.15]$ the agent doesn't seem to be learning at all. In the 8x8 problem there is no learning at all except for $\varepsilon \in [0.75, 0.90]$. When $\varepsilon = 0.9$ the agent is learning well, and has decided actions for most states, with difference to the 4x4 problem. This also reflects what has been said about how the 4x4 problem is harder (requiring a very specific sequence of actions), and how the 8x8 problem is easier although it's larger (more possible sequence of actions).

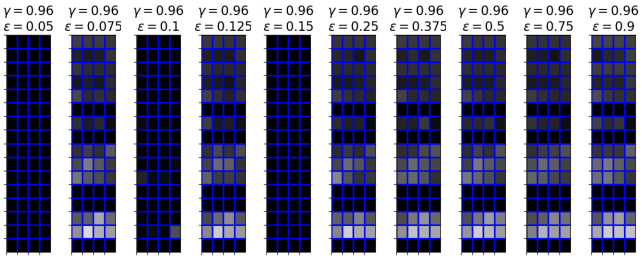

Figure 18: Q-Tables for 4x4 Problem (greedy)

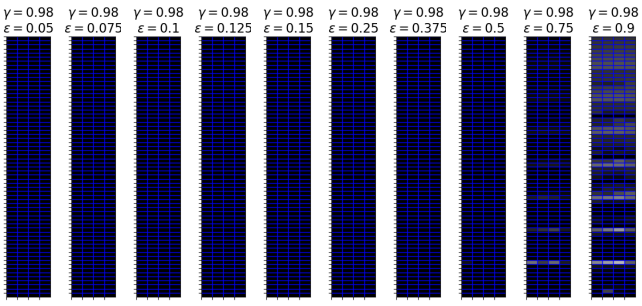

Figure 19: Q-Tables for 8X8 Problem (greedy)

Figure 20 and Figure 21 show the Q-Tables for each γ for the noisy Q-Learner. In all cases we see a much varied response and more certainty on which actions to take. In the greedy tables a lot of states had ambiguous

actions, split almost even equally between each possibility. In this case the agent learns to take a specific action with higher probability, committing each time to the same actions regardless of the discount rate.

This could explain why the noisy Q-Learning works better for these problems, there is less uncertainty on the actions and the agent is better able to reach an adequate solution, although quickly overfitting as discussed before. The mapping of actions in the ε-greedy approach is more ambiguous, and although the agent keeps on learning each episode, it does so more slowly than the noisy Q-Learning. It is also interesting to note that in all cases that there is learning (non-zero Q-tables), the tables tend to look very similar, the discount rate doesn't seem to be affecting the mapping of actions to each state.

Finally, the visualization of the Q-Tables also shows how the more states a problem has the more episodes it will take for it to start learning and developing knowledge. In the 4x4 problem, 10000 episodes are enough for the agent to learn reasonably. In the 8x8 problem, 10000 episodes are insufficient for the agent to learn.

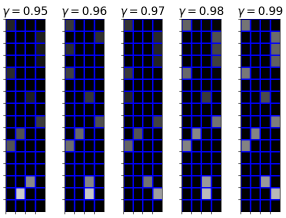

Figure 20: Q-Tables for 4x4 Problem (non-greedy)

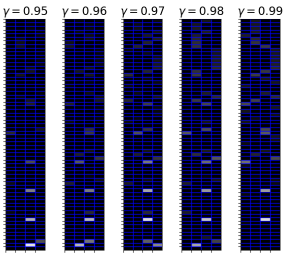

Figure 21: Q-Tables for 8x8 Problem (non-greedy)

Conclusions

Model-based and model-free algorithms were tested for the Frozen Lake problem with a 16 state version and a 64 state version. As expected, model based learning is quick to converge and provides great results, as the agent has prior knowledge on the MDP that it can directly use to determine the optimal policy. On the other hand, model-free algorithms, like the ε-greedy and noisy Q-learning implemented in this assignment, require for the agent to explore and exploit the world of the problem to iteratively determine an adequate policy.

Regarding the size of the problems, even if a larger problem is easier, the nature of the model-free learners will require more iterations to reach a solution just from the size of the problem alone. The presented ε-greedy Q-learning algorithm shows learning but in comparison with Value and Policy Iteration doesn't reach *good* scores. The Q-learning algorithm could be upgraded by using Neural Networks to boost its performance. The stochasticity in the problem makes it difficult, and Q-learning is among the basic algorithms in Reinforcement Learning, so more advanced algorithms should be expected to give better performance.

Reinforcement learning is interesting in the context of MDP's, which has a more tightly knit relationship with AI, supervised and unsupervised learning can also be used for AI, but it is more natural to think of an AI agent as a model-free learner that has to interact with its environment to learn how to reach a goal, which ties in nicely with the very definition of intelligence itself, understanding it as the ability to acquire and apply knowledge.