Exploration of Markov Decision Processes

CS7641 Fall 2023
Tyler Burki (tburki3@gatech.edu)

1 INTRODUCTION

Markov Decision Processes (MDPs) are an architecture for organizing and solving decision making problems in which there exists both an acting agent and a stochastic method that affects results (Wikipedia Contributors, 2019). MDPs use states, transitions between states, actions available to the agent, rewards for achieving states and a representation of the operating environment to describe the problem.

To better understand the concepts of MDPs and reinforcement learning; I will perform three experiments on two MDPs and analyze the results. One MDP will be considered "small" with fewer states, transitions and actions than the other, "large", problem. My hope is to leverage experience gained from solving the small problem in order to solve the large problem. Each MDP will be evaluated using model-based methods value iteration (VI) and policy iteration (PI) as well as the model-free method Q-learning. These methods will be evaluated on an individual basis as well as compared to determine the circumstances under which each might be beneficial.

2 PROBLEM DOMAINS

2.1 Small problem: Blackjack

The Blackjack problem from the Python *gym* library will be used ("Blackjack - Gym Documentation," n.d.). The problem differentiates 290 states, where each state is a 3-tuple of the sum of the player's cards, the card shown by the dealer and whether or not the player has a usable ace. Since an ace can count as the values 1 or 11, a usable ace is one where the player will not bust if the ace is counted as 11. In visualizations the status of the ace will be shown as "H<Sum of hand>" when any ace in hand is not usable (hard state) and "S<Sum of hand>" when any ace in hand is usable (soft state). In this version of blackjack, there is also a special state for a blackjack (natural) hand that occurs when the first two cards dealt to the player sum to 21 ("Blackjack - Gym Documentation," n.d.). This special state has an increased 50% reward over a standard win. To handle the affected state space, I needed to include a fourth value in the state tuple: whether or not a sum of 21 was a natural blackjack. Because these states are a subset of the "S21" states, we can extract them without affecting the formulation of the problem described above.

The problem can be considered small even though it has a decent number of states due to its small number of actions at each step—stay ("S") and hit ("H")—and well-defined transitions that lead to absorbing states quickly. Blackjack can be an interesting problem to learn from because of its heavy reliance on stochastic processes that can significantly change the strategy. The introduction of an adversary in the form of the dealer adds further intrigue as even properly maximizing the player's hand may not lead to a successful outcome.

2.2 Large problem: Frozenlake (20 x 20)

The FrozenLake problem from the *gym* library is used ("Frozen Lake - Gym Documentation," n.d.). The problem has been modified to be a 20 x 20 gridworld problem. The goal of the problem is to guide an elf from one end of a frozen lake to the other without falling in an ice hole (an absorbing state). The agent can traverse any of the 400 states by deciding to move left, down, right or up. The problem has the option to make the ice "slippery", which by default applies a stochastic process to decision making by giving the agent only a 33% chance to transition into their desired state with a 33% chance each to move in one of the two orthogonal directions. This feature has been enabled for my environment, but I have tuned the success rate of a desired transition to be 80% with a 10% chance to move in each orthogonal direction. Additionally, the default rate of hole generation has been lowered from 20% to 5%. Rewards were also changed with holes generating a -500 value (opposed to 0) and the goal giving +500 (opposed to 1). Step rewards were also changed from 0 to be $-1 + (1 - 1 / Manhattan\ distance(state, goal))$. To accomplish this, I was able to exploit the definition of the problem which always sets up the start position at (0,0) and (in this case) the goal at (19,19). This system incentivizes exploration towards the goal since the penalties will decrease as the goal is approached. These changes were performed to help the algorithms converge in a reasonable number of iterations so that meaningful results could be procured.

This is a large problem not only due to its number of states, but the agent has twice as many options to transition through the world as in blackjack. Frozenlake is different from blackjack in an interesting way in that it is possible to avoid most bad absorbing states by

simply not choosing to move in the direction of, or orthogonally to, such a state. What effect will this have on the optimal policy? Will the holes be avoided at all costs, or will some risk be deemed acceptable?

3 VALUE ITERATION

VI is a model-based learning algorithm that leverages the knowledge of all aspects of the MDP to generate a "value" of being in a particular state. This value is a combination of the reward for that state with the discounted value of the best potential future states. The amount future values are discounted is dependent on parameter gamma, which is a real number between 0 and 1. Lower gammas bias towards immediate rewards while larger gammas bias towards long-term rewards. The algorithm iterates through all states indefinitely, updating values based on the results of the last iteration and the values of reachable states, until the values are updated less than some parameter theta (Bettosi, 2023).

3.1 Motivation

VI is guaranteed to converge to the optimal policy (Bettosi, 2023). This makes sense when we consider that because the algorithm is model-based, we have access to all states, rewards and transitions and we can monotonically improve the value of each state (or at least maintain it) in the direction of the truth by looking ahead at future states. It follows that with incremental improvement at each iteration we will eventually converge to a solution that satisfies theta in a finite amount of time. The simplicity of the algorithm and guarantee of convergence makes it a worthwhile algorithm to explore for solving MDPs.

3.2 Method

Both problems were solved using value iteration found in the *bettermdptools* Python library (Mansfield, 2023). Both problems were solved with the default hyperparameters of *gamma* = 1, *n_iters*=1000 and *theta*=1x10^{-10}. The library drops the reward for all states to zero when the algorithm converges, and this was used to determine the point of convergence.

3.3 Results

VI over blackjack converged to an average score of .07979 after only 12 iterations in 0.02 seconds. Convergence on Frozenlake occurred at iteration 291 after 0.78 seconds with an average score of 474.3878. The average scores by iteration can be seen in *Figure 1*. Since VI converges to the optimal policy, we can ascertain that the heatmaps of *Figure2* are a good approximation of the optimal values and policy of each state.

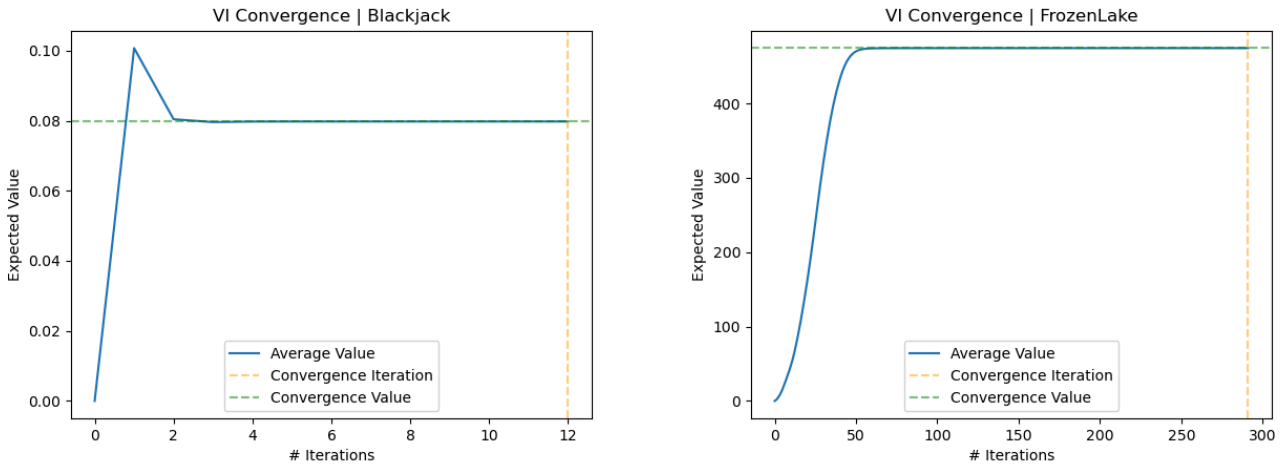

Figure 1

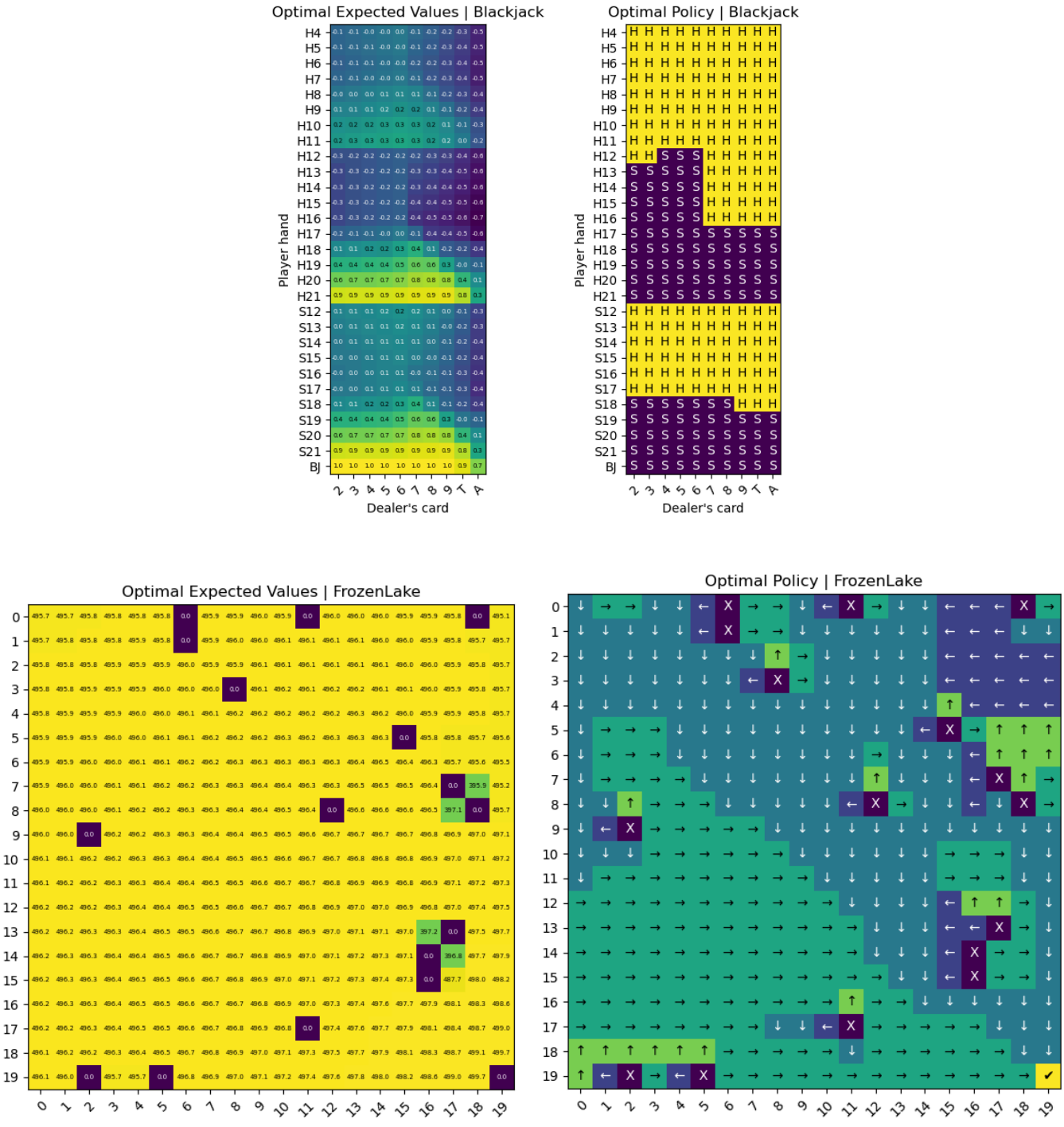

Figure 2

3.4 Analysis

Convergence for the blackjack problem is achieved in 25 times fewer iterations and 39 times less time, lending me to believe that my selection of small and large problems was correct. There are simply fewer states, actions and transitions than Frozenlake which means

fewer states to iterate over and fewer actions to calculate the result of. Frozenlake's convergence function is not particularly remarkable, as the mean value begins low and quickly increases to its convergence value after more of the states are updated. Since step rewards are negative and the only positive reward comes from reaching the goal, it follows that many states can reach the goal without causing the agent to wander forever. Comparatively, it's interesting that blackjack's mean value peaks at iteration one. I had expected a monotonically increasing curve instead of a monotonically increasing one since value iteration was "improving" the values on each step. When I examine the problem, however, I see that there are a number of intermediate states that are further than average from a terminal state and therefore may overestimate their worth. A hard 4 hand is an example of this since hitting will never cause the player to bust but staying will still win if the dealer eventually busts. After a single iteration, it may appear that H4 is a valuable hand to have for these reasons, but at convergence we see that it is one of the worst hands because the combination of cards necessary to beat the dealer are less likely to appear than a different hand that displays similar characteristics after one iteration but has many more winning opportunities like H11.

Blackjack's expected value heatmap makes sense in that player hands like a natural blackjack have expected values of 1 for all dealer cards excluding a ten and an ace. This is because only a natural blackjack from the dealer can prevent the player from gaining a reward as the result is a tie. Other strong hands like S21 and H21 are highly-valued amongst their usable ace grouping. The policy for blackjack enables these values by avoiding hitting and busting unless it is likely the dealer has a better hand. For example, since the dealer must stay at 17, policy dictates that the player should stay at 18 if the dealer's card is 8 or less. This strategy gives the player the highest likelihood to at least tie the dealer, or force the dealer to traverse a number of suboptimal states in order to achieve a winning hand.

I found it interesting that the values in the heatmap for Frozenlake were so high for every state. This tells me that the agent can *almost always* navigate to the goal without falling in a hole. This can be seen in the policy, where every desired action is pointed away from the holes in a manner that the orthogonal slippage will not be able to pull the agent into the hole. The state at (0,19) (the top-right corner) advises the agent to proceed to the right—into the wall—instead of risking heading downwards and slipping into the hole. Instead, the agent will remain in that spot until it slips to its right, downward. It's no surprise then that the only states with sub-400 value scores have a hole on two sides, making it impossible to avoid a hole with certainty. Further research could be performed on the effect of the negative reward system on this policy. Presumably, reducing the penalty for falling in a hole would make the policy more likely to take chances around them in order to get to the goal more quickly.

3.5 Conclusion

Fast and simple, VI appears to be a great tool for obtaining the optimal policy for an MDP given the entirety of the information necessary to perform it. That is likely it's greatest downfall: needing to be in possession of all of the information for a system. The number of iterations taken to converge feels a little high, given that we have all of this information.

4 POLICY ITERATION

PI is a technique that uses two steps, policy evaluation and policy improvement, to find the optimal policy for an MDP. In the first step, policy evaluation, an existing (or randomly generated for the first step) policy is used to determine the value of all states. This process is then repeated until the values converge. The policy improvement step then uses these values to determine the maximal policy for each state. The process then begins over with evaluation and continues until the policies cease changing. The algorithm is guaranteed to converge to the optimal policy in a finite number of iterations ("Policy Iteration — Introduction to Reinforcement Learning," n.d.).

4.1 Motivation

By finding the optimal values for each policy before determining changes to the policy, we can hopefully converge in fewer iterations than VI. PI is also guaranteed to converge in a finite number of iterations without the help of theta since our upper bound must be the set of all policies, which is finite. In contrast, VI could theoretically improve the values infinitely ("Policy Iteration — Introduction to Reinforcement Learning," n.d.) as the values are updated towards the limit but never touching it.

4.2 Method

Both problems were solved using policy iteration found in the *bettermdptools* Python library (Mansfield, 2023). Both problems were solved with the default hyperparameters of *gamma* = 1, *n_iters*=1000 and *theta*=1x10^{-10}. The library drops the reward for all states to zero when the algorithm converges, and this was used to determine the point of convergence.

4.3 Results

PI was able to converge on both MDPs in fewer iterations than VI. Blackjack converges after 4 iterations and .04 seconds to the same value as VI, 0.07979. Similarly, Frozenlake converges to the same value as VI, 474.3878, after 14 iterations and 5.32 seconds. The mean values and policies are identical to the values and policies found for both problems during VI and shown in *Figure 2*.

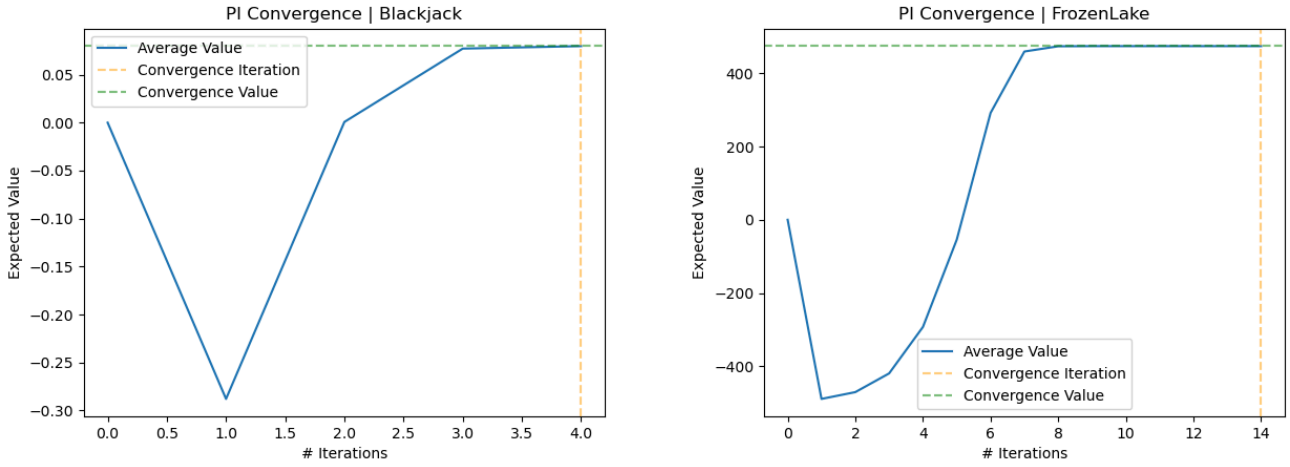

Figure 3

4.4 Analysis

Both problems see their time to convergence increase significantly using PI over VI. This is insignificant for blackjack due to its negligible run time, but Frozenlake sees a substantial increase of 682% in time taken to reach convergence. However, the number of iterations needed for convergence is significantly decreased from the levels seen in VI. It would seem on the surface that PI is less computationally intensive as it only needs to calculate values for each state/action pairing once per iteration (like VI) but can converge in fewer iterations. However, it is the policy evaluation step that must have its values converge that is likely responsible for the increased time taken. Because Frozenlake has a large number of states, getting the values to converge for a particular policy is likely an expensive procedure that, in this case, dominates the ability to perform fewer calculations for each state and action. Further investigation on this point should prove insightful. Does VI begin to struggle for even larger problems? At what point does the scale tip in favor of performing fewer calculations based on the actions?

Both curves in *Figure 3* have a strongly negative expected value, which coincides with the selection of a random policy to begin with. In Frozenlake, for example, the starting policy may insist that the agent walk directly into a hole, garnering the maximum penalty. These inefficiencies are quickly rooted out through policy evaluation and improvement by leveraging the maximum values found during the evaluation phase.

The convergence of both problems to the same values and policy as VI is unsurprising given that both methods have been stated as converging to the optimal policy, of which there is only one. Intuition tells me that values may differ slightly in practice for larger problems, as we are only guaranteed to converge to the optimal policy.

4.5 Conclusion

The guaranteed policy convergence in a finite and bounded number of iterations is an appealing benefit that could prove more useful than VI in a situation where VI is failing to converge to a reasonable answer quickly. For smaller problems or where "good" policy approximations are acceptable, the speed of VI places it above PI.

5 Q-LEARNING

Q-learning is a model-less method for determining the optimal policy of an MDP. Because it is model-less, unlike VI and PI it does not have access to transitions and rewards ahead of time and must experience them in a simulated environment to begin to draw conclusions on what the structure of the problem might be. The algorithm looks to maximize the value return from each state by finding the best action to take in each state and doing so with a probability 1 - theta. It follows that for a certain percentage of actions, defined by theta, the algorithm must take a suboptimal action in the hope of better future rewards. This process helps the algorithm explore the state space and avoid getting caught in a pattern of selecting the "least bad" action. This idea has many similarities to the simulated annealing strategy in that we want the algorithm to explore *a bit* but still make sure it can stay on a good path.

5.1 Motivation

In many real-life scenarios, the information available to VI and PI is not present and they are therefore not options for determining policy. Q-learning gives us an opportunity to derive a good approximation of the optimal policy with little knowledge of the problem structure or environment.

5.2 Method

Both problems were solved using q-learning found in the *bettermdptools* Python library (Mansfield, 2023). A grid search was performed to determine the best hyperparameters for each problem. However, time became an issue as some parameters cause significant performance degradation with the algorithms. As a result, the grid search was performed for smaller numbers of iterations than are seen in the final product. *Table 1* contains the parameters used for each problem. Convergence was defined for the blackjack problem as having 100 consecutive iterations with the mean value generated being less than .00000001 away from the value generated by the first iteration in the sequence. Convergence was similarly defined for Frozenlake, with 100 consecutive iterations at .00001.

Problem	gamma	init_alpha	min_alpha	alpha_decay_ratio	init_epsilon	min_epsilon	epsilon_decay_ratio	n_episodes
Blackjack	.99	.8	.01	.8	1	.01	.9	500,000
Frozenlake	.99	.9	.1	.99	1	.7	.9	250,000

Table 1

5.3 Results

Blackjack converged to 0.078 in 53.10 seconds at iteration 337,046. Frozenlake converged to 474.901 in 367.99 seconds at iteration 206,423. Mean values and policies differ from the optimal ones found in VI/PI for both problems.

5.4 Analysis

The first, most obvious change between the convergence graphs of VI/PI and Q-learning is that the Q-learning curves fluctuate significantly and do not monotonically move towards the convergence value. This is because the agent has no way of telling if a particular action will be beneficial or not until it experiences it for itself. These fluctuations are particularly pronounced for blackjack since, for example, it must learn through trial-and-error that one should not hit on a hard twenty. Conversely, Frozenlake varies less, likely due to the sparseness of the holes as a result of the modification outlined in *2.2 Large problem: Frozenlake*. It is simply easier for the agent to find a "not bad" state by wandering around. The difference in convergence iterations bolsters this claim, as not only does Frozenlake converge in fewer iterations but the values generated are also closer to the optimal in relative terms. Despite these advantages, Frozenlake takes a significantly longer time to converge. The size of the state space is considerably larger, particularly

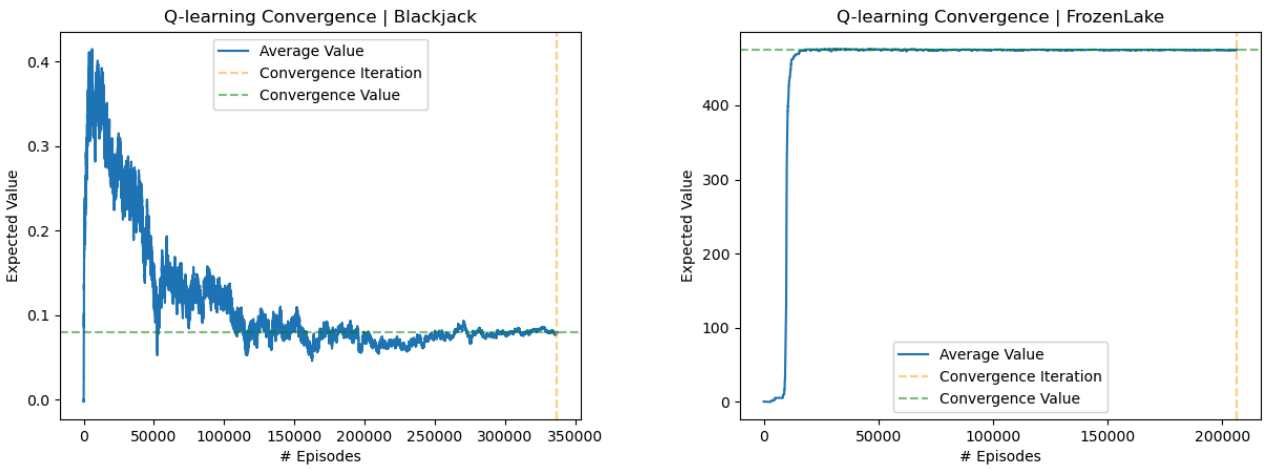

Figure 4

when we have little information on where we are headed, so it is important to coerce the algorithm to converge in as few iterations as possible. Both problems see massive increases in time over VI/PI, once again due to the scarcity of information available to Q-learning and its need to explore the states many times to arrive at an approximation.

Epsilon played a large role with regard to converging Frozenlake. The parameter that determines how often a random action is taken begins and remains high throughout the process in order to ensure that the algorithm explored states not directly on the path to the goal. The reward function dependent on a state's Manhattan distance to the goal likely played a large role in necessitating this. Without a high epsilon, I found that the algorithm would head directly for the goal and ignore less optimal paths altogether. This could be both a boon and a liability for determining the policy, as the policy down the optimal path mirrors that of the optimal policy, but due to the stochastic nature of the underlying environment, if the agent ever finds itself in a state outside that path it could end up in trouble. Blackjack did not need such a level of exploration, and instead benefitted from cutting the learning rate. In blackjack, if we find a good strategy for a particular hand, it is more likely that additional state transitions will lead to a better outcome. Therefore, the blackjack learning tends to adhere more closely to proven strategies it has already discovered. This bias can be seen in *Figure 5* at point (S18, 2) on the policy map where the policy recommends to hit on a soft 18 even when the dealer's card is not valuable.

Overall, the policies generated do not differ much from the optimal, and in cases like Frozenlake may perform as well as the optimal policy given good luck.

5.5 Conclusion

The speed and simplicity of VI and PI cannot be beaten, but in a world without perfect information Q-learning is a decent substitute and was able to approximate the optimal policies to an extent that they would be beneficial.

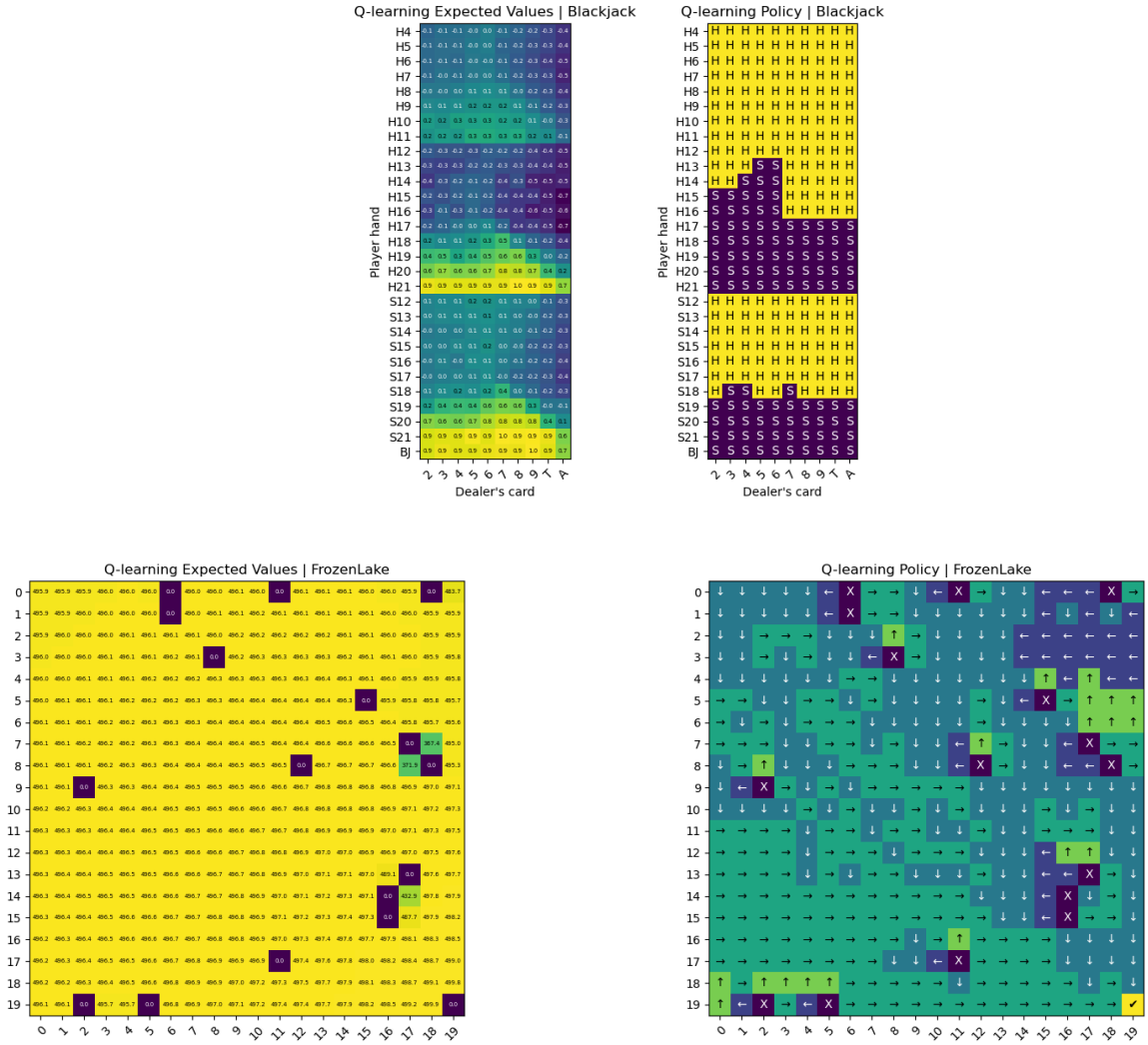

Figure 5

6 CONCLUSION

VI and PI are preferable for situations where the mechanics of the problem are well known. VI is fast and should be preferred to PI in situations where VI is able to converge in a short amount of time and/or iterations. If VI is having difficulty converging, PI is a good replacement as it guarantees a finite and bounded search space. Q-learning is useful for determining an approximate policy without the information used in VI/PI, but tuning the algorithm to achieve a good approximation in a reasonable amount of time can be difficult.

7 REFERENCES

1. Bettosi, C. (2023, September 15). Reinforcement Learning: an Easy Introduction to Value Iteration. Retrieved from Medium website: https://towardsdatascience.com/reinforcement-learning-an-easy-introduction-to-value-iteration-e4cfe0731fd5
2. Blackjack - Gym Documentation. (n.d.). Retrieved from www.gymlibrary.dev website: https://www.gymlibrary.dev/environments/toy_text/blackjack/
3. Frozen Lake - Gym Documentation. (n.d.). Retrieved from www.gymlibrary.dev website: https://www.gymlibrary.dev/environments/toy_text/frozen_lake/
4. Mansfield, J. (2023, March 31). bettermdptools. Retrieved from GitHub website: https://github.com/jlm429/bettermdptools
5. Policy iteration — Introduction to Reinforcement Learning. (n.d.). Retrieved November 27, 2023, from gibberblot.github.io website: https://gibberblot.github.io/rl-notes/single-agent/policy-iteration.html
6. Wikipedia Contributors. (2019, May 20). Markov decision process. Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Markov_decision_process