Project 4: Markov Decision Processes

Scott Merrill
smerrill7@gatech.edu

I. Introduction

Unlike other Machine Learning paradigms concerned with prediction or classification, Reinforcement Learning (RL) problems seek to train agents to act optimally by rewarding desired behaviors and punishing unwanted actions. RL typically requires casting the problem into a Markov Decision Process (MDP), which may not be intuitive or well-suited for all problems. However, properly framing a task as a RL problem often enables for efficient solutions. As such, RL techniques are used widely in domains ranging from supply chain management to robotic motion. This paper seeks to frame two complex problems as MDPs such that they can be solved efficiently in the RL framework.

Three RL algorithms derived from Richard Bellman's dynamic programming equations will be used to solve two MDPs with very different properties. In particular, policy iteration (PI), value iteration (VI) and Q-Learning (QL) will each be considered; the performance of each of these algorithms will be analyzed as to determine their properties and the types of problems in which each algorithm may be preferred.

II. MDP Environments

Framing a problem as an MDP requires defining the states of the environment, the actions an agent can take, the reward an agent receives for taking an action and a transition function which defines the probability distributions of resulting states given the agent took some action a in state s. To effectively evaluate the various properties of each RL algorithm, two problems were selected that differ with respect to each of these parameters that define an MDP.

A. Taxicab Problem

The Taxicab problem is a simplified vehicle routing problem defined by several customers with specific pickup and drop-off locations. The goal of the problem is to identify the most efficient route to get the customers to their respective destinations. The problem has become increasingly relevant in recent years as companies like Uber and Lyft plan to deploy on-demand self-driving taxis.

To frame the problem as an MDP, we considered a simplified 5 X 5 grid world containing 4 passengers R, G, B and Y as shown in Figure 1. The taxi must navigate around the walls to pick up a passenger and transport them to their destination (which may be any other passengers' location). With a 5 X 5 grid, there are 25 possible positions for the taxi. In addition, a passenger can be either at their initial location or in the taxi; thus, there are 5 possible locations for the passenger. Finally, there are 4 possible destination locations. In total there are $25\ X\ 5\ X\ 4 = 500$ states in the given problem. There are also 6 possible actions the taxi can make; move north, move south, move east, move west, pickup passenger and drop-off passenger. The reward function of the MDP contains both latent state and objective state rewards; that is, rewards are provided in both the objective state (when customers are dropped off) as well as the intermediate steps required to reach the objective state. In particular, the agent of the environment is provided a reward of +20 for successful transportation of a passenger, while a penalty of -10 is charged for illegal pickups and drop-offs; that is attempting to pick-up or drop-off a passenger at the wrong location. Finally, a reward of -1 is given at each timestep. Note that this reward representation incentivizes the agent to transport passengers to the correct location in the least amount of time. Finally, the transition function for the taxicab problem is purely deterministic; any attempt to move north will result in the taxi moving north with a probability of 1.

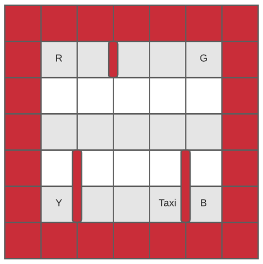

Figure 1

B. Blackjack Problem

Blackjack is a popular zero-sum card game played at casinos globally. Each card two thru ten is assigned a value corresponding to their respective numbers, face cards are given a value of 10 and Aces can take on a value of either one or eleven. A player is initially delt two cards and can choose to "Hit" and receive another card or "Stick" to stop receiving cards. The goal of the game is to get as close to a total value of 21 without exceeding this value and "Busting." The player plays directly against a dealer who is forced to hit when their score is below 17 and stick if their score is 17 or above.

To characterize the game as an MDP, we first make the simplifying assumption that cards are dealt from an infinite deck; this assumption prevents the agent from gaining an advantage by "counting cards" – a strategy frowned upon by casinos whereby a player maintains the current count of high cards and low cards as to gain a potential edge over the house. We will later relax this assumption to consider more complex MDP representations of blackjack allowing card counting strategies. The states of the game can be characterized by the players current sum, the dealers showing card, and whether or not a player has a "usable" ace; that is an ace which can be counted as 11 without a player going bust. The players current sum can take on only values from 12 to 21; sums less than 12 aren't interesting as it is always beneficial for the agent to hit since doing so will only improve their hand (they can't bust in these states). Moreover, there are 10 player sums that are "interesting" for the agent to consider. In addition, the dealers showing card can take on 10 possible values and a player may or may not have a usable Ace. With this representation, there are a total of 10 X 10 X 2 =200 different states. The actions the agent can take are to either hit or stick. The rewards the agent receives are +1 if the player beats the dealer, +1.5 if the player gets a blackjack, 0 if the agent ties the dealer and -1 if the player busts or loses to the dealer. Finally, note that the transition function is stochastic as both the next card dealt from the deck and the dealers hidden card are both unknown. This stochasticity makes the transition function inherently complex. Computing the probability of a player receiving a reward of +1 if he chooses to stick when his hand sums to 15 as a function of the dealers showing card is difficult to compute. Since solving the blackjack MDP using model based RL algorithms requires a transition and reward function, the value of each was approximated using Monte Carlo simulations. In each state and for each action, 10,000 simulations were made, and the transition and reward functions were estimated based on these simulations.

C. Why are these MDPs Interesting

The two MDPs are interesting not only because of their analogs to real world problems but also because they differ broadly in terms of number of states, number of actions, timing of rewards and stochasticity of transition functions. Both the state and action spaces of the taxicab problem are considerably larger than that of the blackjack problem. In addition, the taxicab agent is provided a reward with each action it takes whereas the blackjack playing agent only receives a reward when the game terminates. Finally, while the transition function in the taxicab problem is simple and deterministic, that for the blackjack problem is complex and stochastic. These vast differences between the two MDPs will enable for deeper analysis of the properties of each RL algorithm and help to determine the scenarios certain algorithms may be preferred.

III. Solving the TaxiCab Problem

In this section we introduce PI, VI and QL and use them to solve the Taxicab MDP.

A. Policy Iteration (PI)

PI is a model-based planning algorithm that repeats a policy evaluation step and a policy improvement step until the resulting strategy stops changing. The policy evaluation step approximates the value of each state by recursively applying the bellman operator until the value function converges. The update rule is determined according to (1) where $V^{\pi_k}(s)$ denotes the value of state s for following policy π_k.

$$V_{i+1}^{\pi_k}(s) = \sum_{s'} T(s,\pi_k(s),s')\left(R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')\right) \tag{1}$$

When the value function converges, PI proceeds to the policy improvement step. The policy improvement step greedily selects the action that maximizes the one-step-look-ahead of future rewards and is given in (2).

$$\pi_{k+1}(s) = argmax_a\left(\sum_{s'} T(s,a,s')\big(R(s,a,s') + \gamma V^{\pi_k}(s')\big)\right) \tag{2}$$

This simple two step procedure is guaranteed to converge in polynomial time to the optimal policy and value function for any infinite horizon discounted MDP. With finite horizon's however, PI may fail as the optimal policy may not be stationary. With undiscounted MDP's, the algorithm only converges when there exist zero-cost absorbing states; without these states the value function may diverge leading to sub-optimal or unstable policies.

PI was first applied to the taxicab problem with $\gamma = 0.99$ and the resulting average of the value function for each iteration is shown in Figure 2. The algorithm converges to the optimal policy after 16 iterations in a total of 0.16 seconds; a visualization of the optimal action in each state to pick up passenger R is shown in Figure 3. Note that the given problem has many optimal policies; the long-term reward received is the same whether the agent decides to move north or west when in grid position (2, 1). Furthermore, PI terminates immediately when the mean of the value function reaches its peak indicating the algorithm efficiently identifies one of the many optimal policies. Had the mean of the value function remained constant for many iterations would suggest the algorithm is oscillating between different optimal policies in each iteration. Moreover, PI appears robust to problems small MDPs with multiple optimal policies.

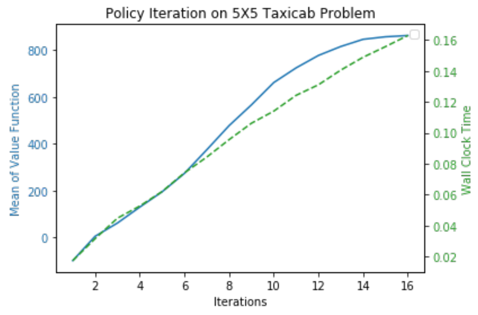

Figure 2

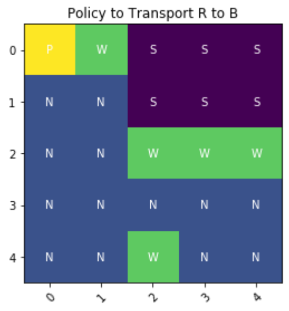

Figure 3

We next evaluate the impact of varying the discount factor (γ) has on the convergence speed of PI. The convergence speed of PI on the taxicab problem for 10 different values of gamma is shown in Figure 4. The time required for PI to converge appears to monotonically increase with increasing values of gamma. Intuitively, this makes sense as with larger values of gamma rewards more distant in the future have a greater impact and need to be considered when estimating the value of a given state. In contrast, smaller values of gamma devalue distant rewards resulting in more myopic behavior of the agent.

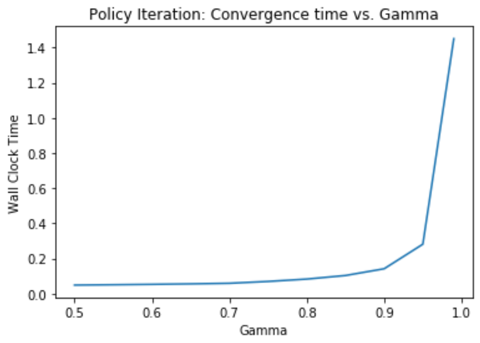

Figure 4

PI was next run on the taxicab problem with gamma set to 1. Figure 5 show the first 20 iterations and as can be seen the algorithm oscillates between two policies and fails to converge. The algorithm was run for 1,000 iterations in total and a consistent pattern was seen in iterations 21 through 1,000. Without discounting future rewards, the value of each state diverges to infinity and thus distinguishing between state values becomes impossible; this results in unstable behavior. Restricting PI to only consider "soft" policies—probabilistic polices where actions in each state have a non-zero chance of being selected—may reduce the inherent trust PI places on the accuracy of the state-value function and perhaps allow for convergence of PI on the undiscounted taxicab problem.

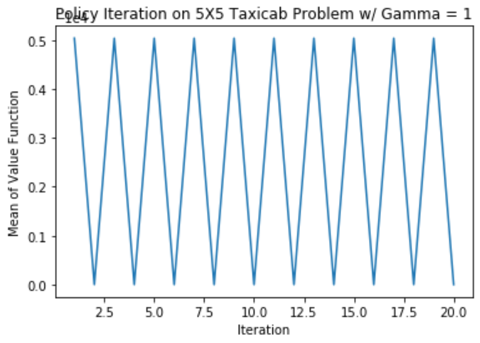

Figure 5

The taxicab problem is interesting as the routing problem is one self-driving taxis will need to solve to maximize efficiency. While PI appears to provide an efficient solution in a reasonable amount of time, the world we considered was a simplified 5 X 5 grid with only 4 passengers. In reality, to ensure reasonable walking distances of passengers to various pickup locations, large cities likely need to be discretized into more than a 5 X 5 grid. In addition, an efficient routing to more than 4 possible destinations is necessary. Moreover, adequate solutions to the taxicab problem should scale well with both the size of the grid and number of destinations. To evaluate the effectiveness of PI as a potential solution to the routing problem, a 10 X 10 grid world with 7 passengers and the same world dynamics is considered. This new world is over 10 times larger and now contains 5,600 states. Figure 6 shows a representation of this more complex world.

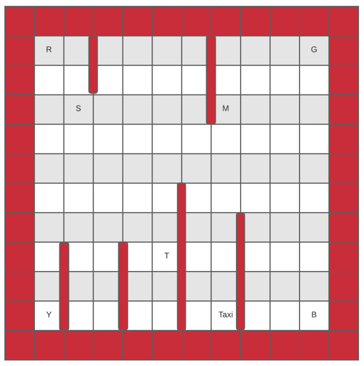

Figure 6

PI was applied the to the complex new world with $\gamma = 0.99$ and the mean value function and execution time is shown in Figure 7; the optimal strategy of the taxi to pick-up passenger R is presented in Figure 8. While the mean of the value function appears to flatline, the algorithm fails to converge to an optimal policy after 1,000 iterations thus suggesting the algorithm is oscillating between different optimal policies on each iteration thus preventing the algorithm from terminating. In the 5 X 5 taxicab problem it was found that PI was robust to multiple optimal policies.

However, in the taxicab problem, the number of optimal policies is an increasing function of the size of the problem and thus when the state space was increased many more optimal policies were introduced. And, with significantly more optimal policies, PI appears less appropriate. Modifying PI to consider early stopping when the value-function stops changing may be necessary in problems with many optimal strategies. In the 10 X 10 taxicab problem, doing so results in the algorithm identifying one of the optimal policies in 64.4 seconds with each iteration taking an average of 1.84 seconds. In the 5 X 5 grid word, PI completed each of its 16 iterations in 0.1138 seconds. Moreover, while the size of the state space increased by a factor of 10, the time for each iteration of PI increased by a factor of 258. Further, the number of iterations required to converge more than doubled. Given state spaces thousands of times larger would be required to model large cities such as NYC, PI may not be feasible solution. Other routing algorithms, or approximate solutions to the MDP may provide results that scale better in the number of states.

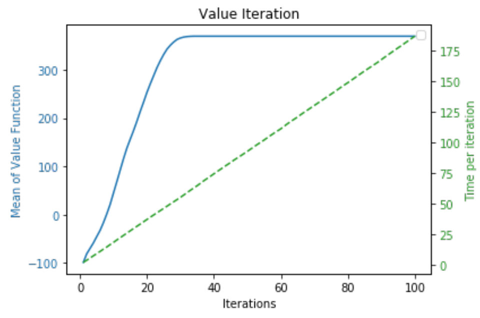

Figure 7

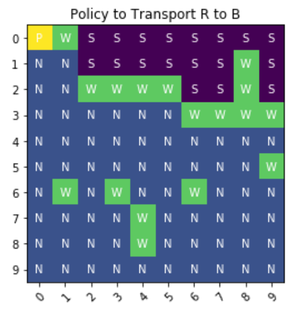

Figure 8

PI was lastly run with varying discount factors on the larger taxicab problem, a plot of the results is shown in Figure 9. The plot is quite interesting as convergence time appears to jump significantly as gamma is increased passed 0.8. This can be explained by many more required iterations of the policy improvement step when gamma is increased passed this point; when gamma is below 0.8, only 40 iterations of PI were required while 100 iterations were required for larger values of gamma. The positive correlation between wall clock time and gamma is consistent with the behavior of the smaller taxicab world; increasing gamma increases the complexity of the problem as the value of particular states must be calculated based on rewards further in the future. Moreover, altering the reward structure of the problem such that it can be modeled with smaller values of gamma will likely improve execution speed and may be appropriate if training time is constrained.

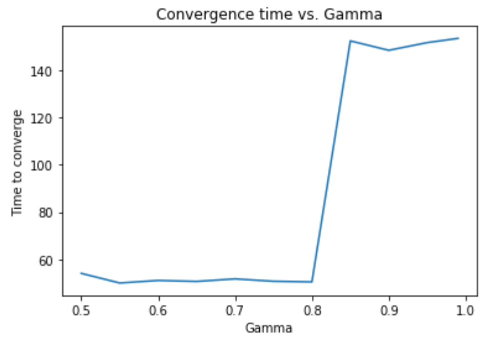

Figure 9

B. *Value Iteration (VI)*

VI provides an alternative to finding the exact solution for infinite horizon discounted MDP's. The algorithm repeatedly applies the bellman update to the state-value function shown in (3) until convergence. The optimal policy is then derived from the optimal state-value function with the greedy one-step-look-ahead shown in (4).

$$V_{i+1}(s) = \max_{a} \sum_{s'} T(s,a,s')(R(s,a,s') + V_i(s')) \tag{3}$$

$$\pi^*(s) = argmax_a \left(\sum_{s'} T(s,a,s')\big(R(s,a,s') + \gamma V^*(s')\big) \right) \tag{4}$$

While running VI until convergence provides an exact solution, it's often computationally expensive. Further, computing the exact state-value function is also often unnecessary as the policy from (4) often converges to the optimal policy prior to the convergence of the state-value function. In 1994, Puterman introduced a stopping criterion in (5). His proof shows that when the span of the state-value function on consecutive iterations is sufficiently low, the greedy policy with respect to this value function is epsilon-optimal. Moreover, in our experiments VI is run until this stopping criterion is met with a selected epsilon of 0.01.

$$span_{\infty}[v_{k_0+1} - v_{k_0}] \leq \frac{1-\gamma}{\gamma}\epsilon \tag{5}$$

VI was run with $\gamma = 0.99$ to solve the taxicab problem and the resulting reward for each iteration is shown in Figure 10. After 1,200 iterations and 0.80 seconds VI converged to the same optimal policy as PI. PI, however, completed in only 16 iterations. This fact alone, however, doesn't demonstrate superior performance of PI over VI. Recall that in each iteration of PI, a policy evaluation step is required to determine the value function for the current policy. The number of computations required for PI is thus a function of both the number of policy improvements and the number iterations in each policy evaluation. In the given problem, PI did achieve superior runtime performance as the algorithm executed almost 5 times faster than VI. The superior performance of PI over VI might be explained by the large action space of the problem. With each iteration of VI, the algorithm evaluates the estimated long-term reward for each possible action. In contrast, in the policy evaluation step, PI only considers the long-term reward of the current policy. PI thus considers 5 fewer actions at each iteration than VI which approximates the speedup observed in the algorithm.

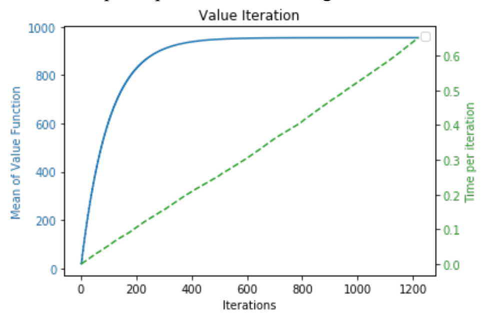

Figure 10

Figure 11 shows the effect various values of γ have on the number of iterations required for VI to converge. With larger values of gamma, rewards more distant in the future have greater impact on the value function; estimating these distant rewards is inherently more complex and requires more iterations to compute. The positive association between gamma and number of iterations required for VI to converge also follows from Puterman's stopping criterion in (5). With larger values of γ, a smaller span between consecutive VI runs is necessary to identify a solution epsilon-close to the optimal policy. Since the bellman operator is a contraction, smaller spans are a direct result of increased iterations.

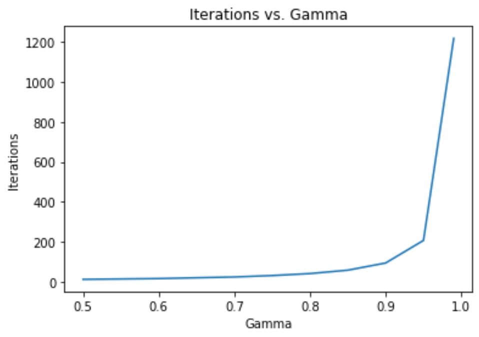

Figure 11

Figure 12 shows the impact setting $\gamma = 1$ has on the value function. The average value function increases with each iteration and fails to converge. Moreover, without discounting or an absorbing state, the value of each state in the taxicab problem becomes infinity and the policy created by a greedy one-step-look-ahead of the state-value function is unstable.

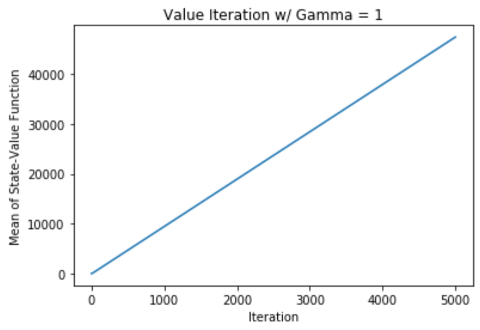

Figure 12

We finally apply VI on the 10 X 10 taxicab problem to evaluate how well algorithm scales with increasing problem sizes. Figure 13 shows the average of the value function and time for each iteration. The algorithm converges to the same policy as PI after 1,200 iterations and a total of 50.37 seconds. The fact that VI converged in the same number of iterations irrespective of problem size makes sense; given the deterministic nature of the problem; increasing size doesn't make learning the optimal policy more difficult but only more time consuming. The total execution time increased 77-fold. While such indicates VI also doesn't scale well with increasing state spaces, VI appears more equipped than PI which took over 500 times longer on the larger problem.

The fact that VI is more robust with increasing number of states than PI may, however, be problem specific rather than a universal rule. In the taxicab problem, increasing the number of states also has the effect of increasing the number of optimal policies. In problems with many optimal policies, solving for the value function and using it to derive a policy is more efficient than solving for the optimal policy directly using PI. In problems with many optimal policies, solving for the policy directly results in a very variable value function as in each iteration a different optimal policy may be followed. Solving for the value function directly using VI doesn't suffer from changing policies; rather, the value of the best action at each state is more consistent and thus allows for faster convergence.

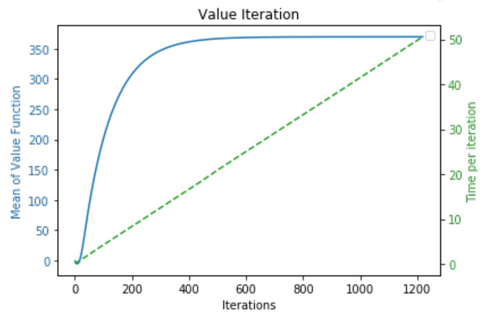

Figure 13

We finally evaluate the impact different discount factors have the convergence speed of VI in the more complex representation of the taxicab problem. The convergence speed of VI on the taxicab problem for 10 different values of gamma is shown in Figure 14 and demonstrates that convergence time appears to be exponential in gamma. Moreover, in problems with training time constraints, framing the problem such that they can be represented with smaller values of gamma may allow for quicker convergence and thus be more appropriate representations.

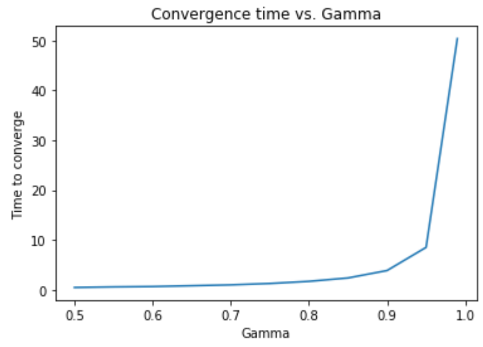

Figure 14

C. Q-Learning (QL)

QL is a model-free RL algorithm which doesn't require knowledge of the underlying transition probabilities and reward structure of the MDP. Instead, the algorithm estimates the action-value function, $Q(s,a)$, through experience and repeated sampling from the environment. The algorithm continually observes the current state s, takes an action a, and observes the resulting reward r, and next state s'. The algorithm then makes updates to $Q(s,a)$ based on the observed (s, a, r, s') tuple according to the QL update rule in (6).

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\left[r_t + \gamma \max_a Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \qquad (6)$$

In 1992, Chris Watkins demonstrated that with repeated application of (6), and infinite visitations to each state action pair, $Q(s_t,a_t)$ is guaranteed to converge to the optimal Q-function [2]. And, once $Q^*(s_t,a_t)$ is found, the optimal policy can be derived simply according to (7).

$$\pi^*(s) = argmax_a Q^*(s_t,a_t) \qquad (7)$$

To ensure adequate exploration, our implementation of QL uses an $\epsilon - greedy$ policy such that with probability ϵ, the algorithm makes greedy use of the current Q-table to select an action and with probability $1-\epsilon$ a random action is selected. ϵ is initially set to 1.0 and is slowly annealed by a decay factor of 0.99 to a minimum value of 0.01. The algorithm repeatedly samples the MDP until convergence. We consider QL to have converged when the 10, 50 and 100-episode average reward all producing similar results.

QL was run on the taxicab problem to identify the optimal policy without knowledge of the underlying MPD. The average episodic rewards are shown in Figure 15 and the policy identified is shown in Figure 16. The policy found using QL is different than that identified by both PI and VI, but still allows passenger R to be picked up in the minimum number of steps and is thus an alternative optimal policy.

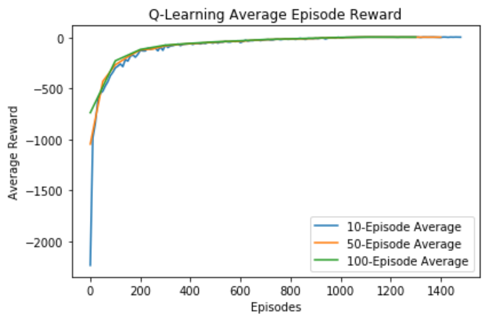

Figure 15

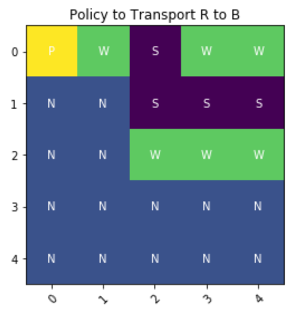

Figure 16

The number of episodes QL, takes to converge is highly sensitive to the exploration strategy. While we still maintain an $\epsilon - greedy$ exploration strategy, grid search was used to identify the initial values, decay factor and minimum values for ϵ that results in the quickest convergence. These values were found to be 0.6, 0.9 and 0.1 respectively and the learning curve for the optimized QL algorithm is shown in Figure 17. Convergence appears to occur slightly quicker in around 300 episodes and is likely a result of the larger minimum epsilon threshold which allows for quicker exploration of the state space.

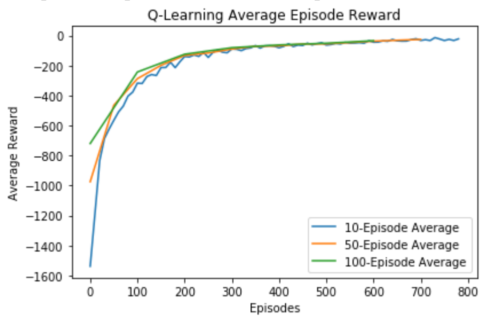

Figure 17

The convergence of QL is also highly dependent on the learning rate which determines the amount the Q-function is updated toward the Q-target on each iteration. We use grid search to find the initial value, decay factor and minimum value for alpha that results in the quickest convergence and the results are shown in Figure 18. It was found that decaying alpha slowed the convergence of QL and larger initial learning rates outperformed smaller initial values. This occurs due to the deterministic nature of the taxicab problem. The learning rate can be thought of as the weight placed on new observations; at its extremes, learning rate of 0 completely ignores new information, while a learning rate of 1 ignores previous Q-table values. It's often necessary to decay alpha in stochastic MDP's to stabilize learning and counteract the randomness. In purely deterministic MDP's like the taxicab problem, however, decaying alpha becomes unnecessary and the optimal value for the learning rate is 1.

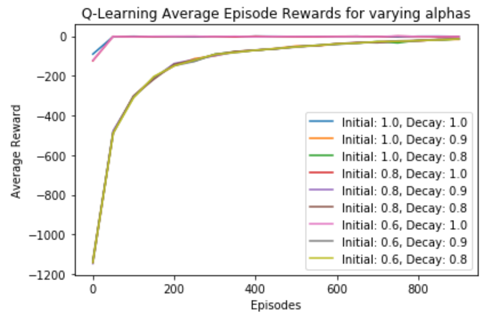

Figure 18

To evaluate the impact discounting has on QL, Figure 19 plots the episodic rewards received when setting $\gamma = 1$. While QL appears to converge and obtain consistent negative rewards in each episode, the policies identified are meaningless. Like PI and VI, without discounting future rewards, the true value of every element in the state-action value function becomes infinite. Since all values have the same true value, their approximate values during the learning process are also similar. This creates a myopic agent concerned with maximizing immediate rewards. The final policy found is effectively one where the agent makes random actions until it fortuitously winds up in a state where it can pick-up a passenger and achieve a positive reward.

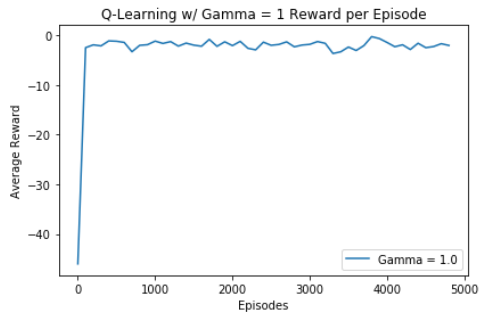

Figure 19

The optimized QL was next used to evaluate the more complex taxicab problem. The mean-value function and time to converge are shown in Figure 20. The algorithm takes 2,000,000 iterations and over an hour to converge. In contrast, QL took only a minute to converge on the 5 X 5 taxicab problem indicating that when the state space increased by a factor of 10, the execution time of QL increased by a factor of 60. This scaling factor makes sense as there are 6 possible actions the agent can make in each state. Thus, with smaller action spaces, QL will scale more efficiently with the number of states. Still QL appears to scale better with increasing state spaces than both PI—which took more than 500 times longer— and VI—which took 77 times longer.

The policy identified by QL is shown in Figure 21 and differs significantly from the optimal policy. In addition, there are some states that make blatant suboptimal decisions. In location (3, 0) for example, moving south will result in the taxi moving away from the pickup location of passenger R and incurring a negative reward of -1. Moving north in this state, while it will still result in a negative reward, will bring the taxi closer to the customers pickup location and should thus be preferred. Since QL is only guaranteed to converge to the optimal policy if each state action pair is visited infinitely often, the algorithm failing to identify the optimal policy is an indication of an inadequate exploration strategy. Furthermore, a slower decay speed of epsilon, or a larger minimum epsilon threshold would likely result for greater exploration and thus allow the algorithm to identify one of the many optimal policies.

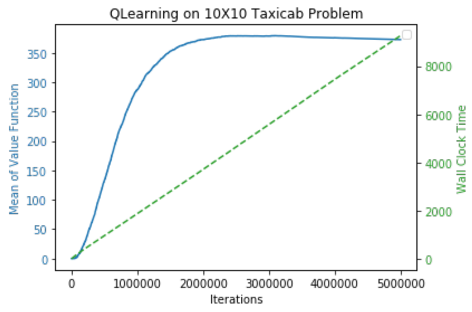

Figure 20

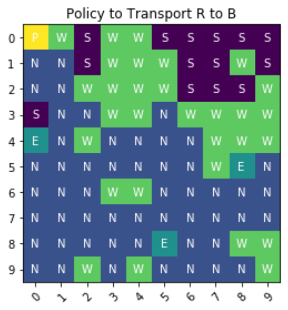

Figure 21

IV. Solving the Blackjack Problem

Casinos pay the same regardless of the number of actions made by the player; winning a game with 3 cards has the same payout as winning a game with 4 cards. Moreover, the appropriate value of gamma is 1 in the blackjack problem and any discounting of rewards would result in an agent that prefers to win with fewer cards.

In the taxicab problem, we saw setting $\gamma = 1$, was troublesome for the convergence of VI, PI and QL; in each case, the divergence of the value function resulted in the long-term rewards being indistinguishable resulting in inappropriate policies. To ensure a bounded value function when setting $\gamma = 1$, an additional absorbing state was created. Upon the conclusion of a blackjack game, the system enters this absorbing state, remaining there forever and receiving a reward of 0 in perpetuity. This state will enable the convergence of VI, PI and QL when the discount factor is set to 1.

A. Policy Iteration (PI)

Figure 22 plots the average value function on each iteration of PI on the blackjack MDP with $\gamma = 1$. The algorithm converges in only 2 iterations and takes a total of 0.72 seconds. The algorithm thus appears efficient in solving small stochastic MDPs with complicated reward functions.

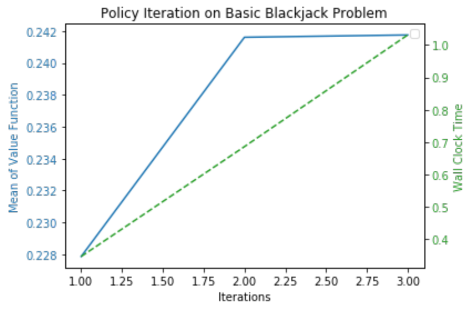

Figure 22

An interesting note from Figure 22 is the mean of the value function is positive, which is misleading as the value of a game of blackjack to the player is negative; given our modified rules, the "house" or casino has an edge of around 0.61% meaning if the player were to follow the optimal basic strategy (stationary strategies that don't depend on the current card count) the expected value of a \$1 bet is .9939. Taking the mean of the value-function is a biased way to evaluate the expected value of the game as it ignores the probabilities of observing each particular state. This fact can be most easily seen through analysis of the optimal value function presented in Figure 23. Such shows that when the player has a usable Ace (recall that this means it can be counted as 11 without causing the player to go bust), the value function is almost always positive, and in some states, significantly positive. However, the probability of a player having an Ace, let alone a usable one, is only $\frac{4}{52} = 7.69\%$. Furthermore, taking the mean of the value function produces a biased estimated value of the game as these unlikely states are weighted the same as more probable ones.

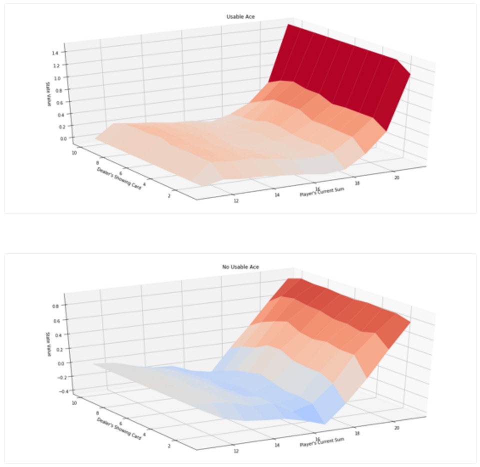

Figure 23

A visualization of the optimal policy from the algorithm can be seen in Figure 24. When the player doesn't have a usable Ace, the optimal policy is much more conservative. For example, the strategy suggests a player should always stick on a hard 18, but on a soft 18 the player should only stick when the dealer's show card is a 7. It makes sense to play more aggressively with a usable Ace as it's impossible for the player to bust with one additional card. In contrast, hitting is riskier if you don't have a usable Ace and thus more conservative strategies should be followed.

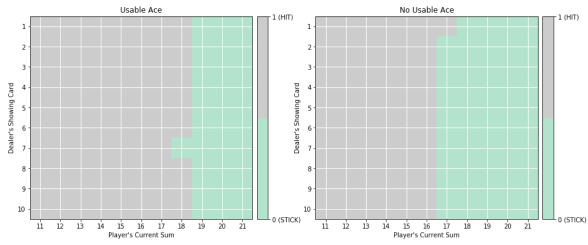

Figure 24

While setting $\gamma = 1$, makes sense in the context of the problem, we now consider setting $\gamma = 0.1$ as to evaluate the changes in behavior of the optimal policy. With $\gamma = 0.1$, PI converges in a single iteration taking 0.0017 seconds. Thus, with gamma reduced by a factor of 10, the effective horizon of the MDP is shortened and PI converges 400 times faster in half as many iterations. A visualization of the value function and optimal policy identified when setting $\gamma = 0.1$ is shown in Figures 25 and 26 respectively. The optimal policy appears more conservative than the undiscounted case indicating a preference of the player to win with fewer cards. This makes sense as with low gammas, rewards received significantly in the future are worth less to the agent. Additionally, we find the values of any particular state is lower than the undiscounted case and state values exhibit more extreme lows. The value to the player when their hand sums to 16 for example is much lower when we discount future rewards. In this state, the agent ill-advisedly sticks and receives a large negative reward when the dealer wins. Such illustrates how the reward structure of the problem incentivizes the agent not only to win, but to win with fewer cards resulting in significant departures from the optimal strategy and an increased house edge.

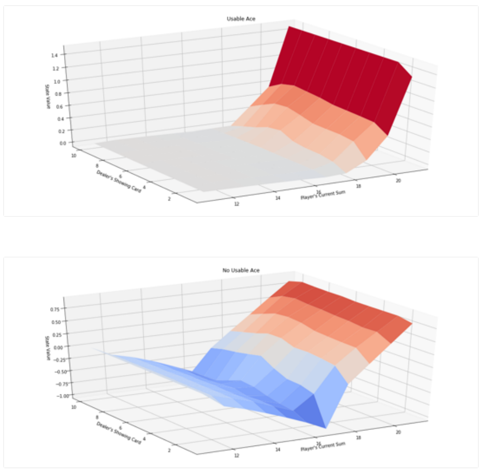

Figure 25

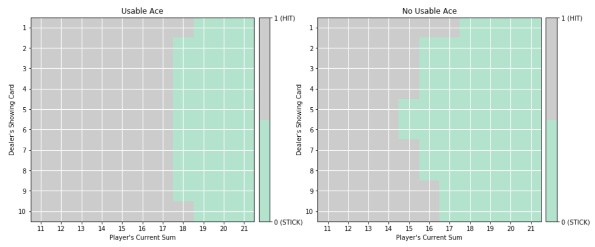

Figure 26

We next add complexity to the blackjack MDP by relaxing the assumption that cards are drawn from an infinite deck. Instead, we consider a situation in which a player joins a random blackjack table at a casino that uses a single deck. Since the player joins midgame, we also assume 26 cards have been drawn from this deck. The same rules of the game discussed previously apply, however now cards are drawn from this reduced deck without replacement.

These relaxed assumptions are interesting as they allow for more sophisticated probabilistic strategies such as card-counting. To enable an agent to learn a card-counting strategy the state of the problem is expanded to include the current count of the deck. The decks count is defined as the total number of 10s, face cards and Aces drawn less the total number of 2s, 3s, 4s, 5s and 6s drawn. A positive count thus indicates a deck that has more "low" cards remaining while a negative count indicates a deck with more "high" cards. The state of the game is now a function of the players current sum, the dealers showing card, whether the player has a usable Ace and the current count of the deck. To limit the state space in the problem we cap the count as +10 and -10; counts larger than +10 are thus treated equally as counts equal to +10. This representation of blackjack contains $10 * 10 * 2 * 21 = 4{,}200$ states.

With this new representation of the game, the previous transition and reward functions are irrelevant. To estimate the new transition and reward functions 10,000 Monte Carlo simulations were executed in each state and for each action; the resulting transition and reward functions were then approximated based on these simulations.

PI was run on this more complex representation of blackjack with $\gamma = 1$ and the mean value function at each iteration is shown in Figure 27. While PI takes only one additional iteration to solve the problem, the algorithm takes 31.36 seconds and is 43 times slower than the 0.72 second execution time on the basic blackjack problem. With 21 times more states, the policy evaluation step takes significantly longer to converge on each iteration thus deteriorating the performance of PI as a whole

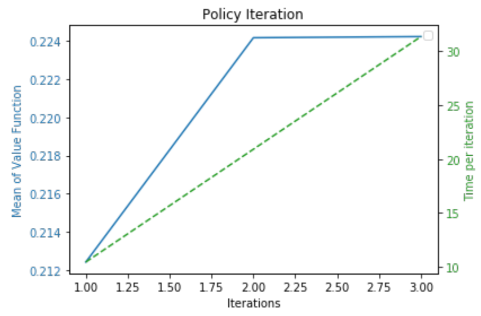

Figure 27

Another interesting note is that the mean of the optimal value function fell slightly when relaxing the assumption of an infinite deck. While the mean of the value function is still a biased estimate of the expected value to the player when following the optimal strategy, the impairment is still relevant as it suggests certain counts may be beneficial to the house. The antithesis to this is also true and the player can gain an advantage by keeping track of the current game count. To illustrate this fact, Figure 28 shows the optimal policy of the player when the current count is -5 and Figure 29 shows the optimal policy of the player when the count is +5. As can be seen, the policies are quite different. When the count is highly negative (that is -5) such suggests there are more high cards remaining in the deck and when the players hand exceeds 14 (and they don't have a usable Ace) it's almost always advised to stick. In contrast, a highly positive count indicates, the remaining deck is more heavily weighted towards lower cards. When this occurs, it's beneficial for the player to always hit when their hands sum falls below 17.

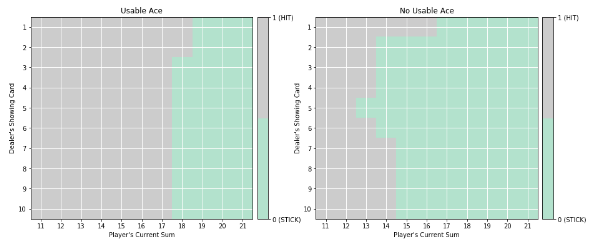

Figure 28

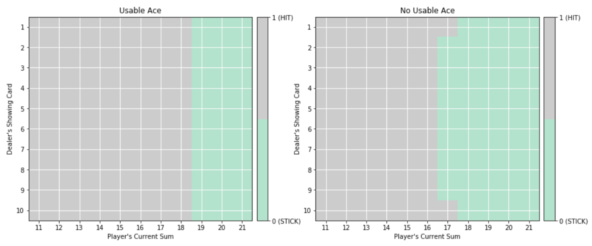

Figure 29

Figures 30 and 31 show the optimal value function observed when card count is -5 and +5 respectively. When the count is -5, the value to the player as a function of the dealer's show card is minimized when they lack a usable Ace and their hand sums to 16. In these states, due to the large number of high cards remaining in the deck, the best decision for the player is to stick even with a relatively low hand score; the probability of going bust is too high to justify hitting and the best chance of a positive reward is for the dealer to bust. When the count is high, these states still offer negative value to the player, however, they aren't as bad as with a low count. With a higher count, there are more low cards in the deck and hitting on a 16 is less likely to bust.

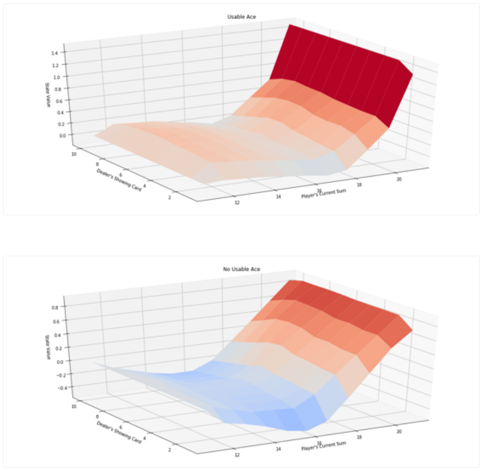

Figure 30

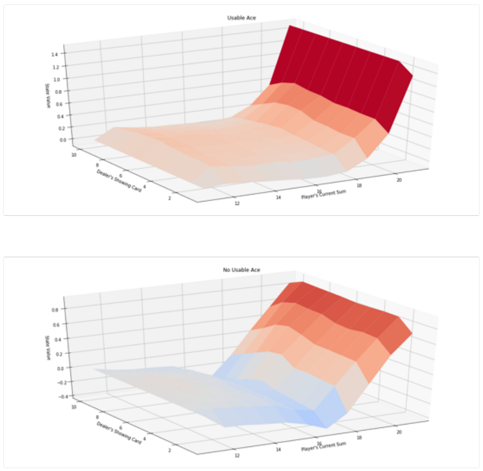

Figure 31

We next consider the effect setting $\gamma = 0.1$ has on the convergence of PI and the resulting optimal policies in the complex representation of blackjack. Figure 32 shows the mean of the value function plotted against wall clock time. PI converges in 2 iterations and taking a total of 0.35 seconds which is much faster than the 31 second runtime found when $\gamma = 1$. This is again consistent with our previous findings that the runtime of PI improves with lower values of gamma as rewards further in the future need not be considered.

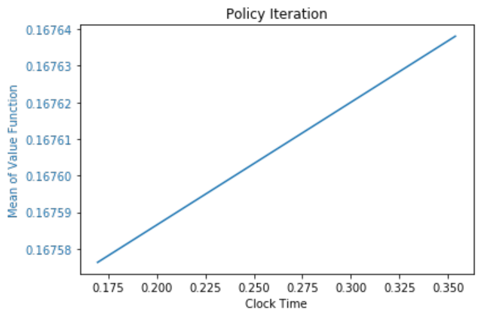

Figure 32

Figure 33 and 34 show the optimal policies when the count is -5 and +5 and $\gamma = 0.1$. In both cases we note that lower the value for gamma gives the player preference to win with fewer cards thus encouraging them to stick.

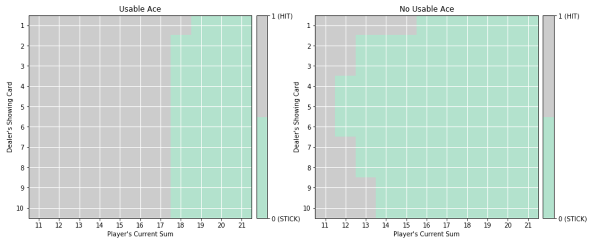

Figure 33

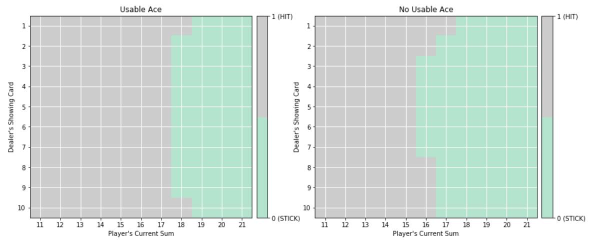

Figure 34

B. *Value Iteration (VI)*

The time and average of the value function for each iteration of VI with $\gamma = 1$ on the basic blackjack MDP is shown in Figure 35. 5 iterations of VI and 0.0013 seconds were required, and the value function converged to the same policy produced by PI. Given VI found the same policy in a fraction of the time, the algorithm appears preferred over PI on the basic blackjack problem. The outperformance of VI over PI on may be explained by the existence of only two actions in each state. Since value iteration works by greedily maximizing the one-step-look-ahead, each possible action needs to be considered at each state. While in the taxicab problem there are 6 possible actions at each state, there are only 2 in the blackjack problem; hit or stick. With only 2 possible actions at each state, querying the action space is cheap and results in the quick identification of the optimal policies.

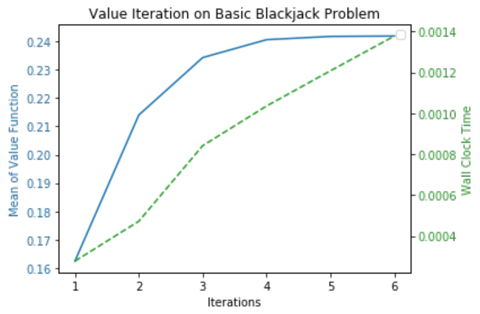

Figure 35

When VI was run with $\gamma = 0.1$ the resulting policy and value function were found to be identical to that found with PI and the algorithm converged in 2 iterations taking only 0.00036 seconds. The quicker convergence with smaller values of gamma is consistent with what we've observed previously and can again be explained by the fact that with larger values optimal actions need to consider more distant future rewards.

We next consider the performance of VI on the complex representation of blackjack with a discount factor of 1. The mean of the value function and execution time on each iteration is shown in Figure 36. VI converges in 5 iterations in 0.011 seconds. Moreover, while the state space of the problem increased by a factor of 21, VI converged in the same number of iterations and taking only 8 times longer. VI thus appears to scale quite well with increasing state spaces which may be counterintuitive. In a single iteration VI loops over each state querying all possible actions and resulting states; its worst-case complexity is thus $O(|S|^2|A|)$ suggesting with larger state spaces, the algorithm should perform significantly worse. VI's robustness to increasing state spaces on the blackjack problem is perhaps explained by the sparsity of the transition function. While there are 4,200 states in the advanced blackjack problem, only a small subset of different states can be reached from any given state. Thus, only a few next states need to be considered when the algorithm performs a greedy one-step-look-ahead leading to VI performing significantly better than its worst-case complexity.

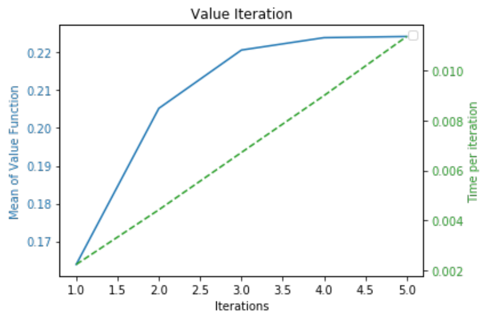

Figure 36

While both VI and PI converge to the same policy, VI identifies this policy in a fraction of the time; VI converges in 0.011 which is significantly faster than the 30 seconds required for PI. This is a consistent result with the simplified version of blackjack with only 200 states and likely occurs due to the small action space of the problem. With only 2 actions, querying the entire action space on each iteration is inexpensive. Further, the greedy action selection of VI results in more stable policies on each iteration leading to quicker convergence.

VI was lastly run on the complex representation of blackjack with a discount factor of 0.1. The algorithm completes in two iterations lasting 0.0045 seconds. The policies found by VI with this smaller discount factor are the same as the optimal policies found by PI. From this experiment, we once again observe that VI performs quicker than PI and reducing the discount factor increases the convergence speed of the algorithm.

C. *Q-Learning (QL)*

QL is first next run with $\gamma = 1$ to solve the basic blackjack problem. The algorithm is run with the default QL parameters defined in MDPtoolbox which specifies an initial learning rate of 0.1, a learning rate decay of 0.99, a minimum learning rate threshold of 0.001, an initial epsilon value of 1.0, an epsilon decay speed of 0.99 and a minimum value of epsilon of 0.1. The execution times and mean value function for the default QL after 200,000 iterations is shown in Figure 37. The algorithm fails to converge with these default parameters as the mean value function increases monotonically rather than flattening to a particular value. This is likely due to the low learning rate. The initial learning rate of 0.1 is quickly annealed to its minimum value after 459 iterations ($0.1\ X\ 0.99^{459} < 0.001$). This quick decay schedule makes for slow learning not suitable for solving MDP's with complex transition and reward functions. While the algorithm will still converge in the limit, after 200,000 iterations, the mean value function is a tenth of the optimal values found by PI and VI indicating millions of runs will be necessary for QL to converge with these parameters.

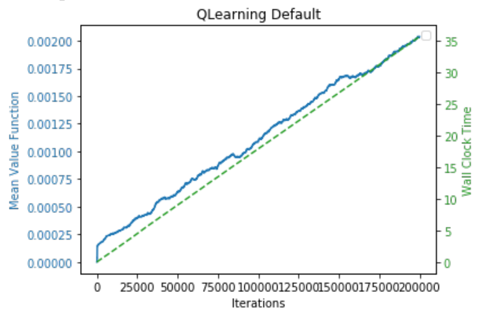

Figure 37

Though the default QL algorithm doesn't converge, we still plot the identified policy in Figure 38. While the strategy found is similar to the optimal policy when a player has a usable Ace, many more discrepancies are observed when such is not the case. This might indicate, that when a player has a usable Ace, the best action to take is more obvious and thus easier to learn. In contrast, when a player lacks a useable Ace, the best actions are less obvious; that is, the expected value achieved by hitting and sticking is similar. One example of the more complex expected value calculations can be observed in the state where the player lacks a usable Ace, has a hand that sums to 15 and the dealer is showing a 6. The optimal policy from PI for this state is to hit while the policy found by QL says to stick. In this state if the player hits, they will bust and lose if the flipped card is greater than a 6 which occurs $\frac{7}{13} = 53.85\%$ of the time. Alternatively, the player can choose to stick and hope the dealer busts. Given more than 30% of cards are worth ten the dealers hand most likely sums to 16. And since the dealer is forced to hit on a 16, they will bust, and the player will win if the flipped card is larger than a 5; such occurs $\frac{8}{13} = 61.53\%$ of the time. While this seems to indicate the optimal decision is to stick, it also assumes the dealer's hidden card is the most likely card in the deck. If instead the player assumes the dealers hidden card is equivalent to the expected value of all possible next cards, which is 6.54, then the dealer's hand is assumed to sum to 12.54 and the player will lose more often than not if they choose to stick.

Many of the states in which the QL policy differs from the optimal policy have similarly complex optimal actions. And these optimal actions may provide only a slight edge to the player if they are followed. Moreover, these states where the optimal policy provides only a slight advantage over an alternative policy are much more difficult to learn and many more iterations or better exploration and learning strategies are required to learn them.

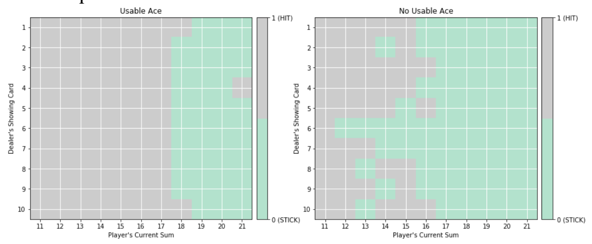

Figure 38

To support the convergence of QL on the blackjack problem, grid search was used to find the optimal decay speeds and minimum values for alpha and epsilon. These decays and minimum thresholds were found to be 0.9999 and 0.1 for epsilon and 0.9999 and 0.001 for alpha. Figure 39 shows the resulting convergence time and mean value function for QL run with these parameters. With the tuned parameters, convergence occurs and after only 75,000 iterations. In addition, the resulting mean of the value function found is around 0.09, which is much closer to the optimal average value function value of 0.224.

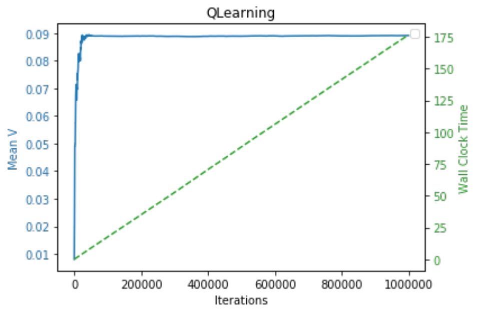

Figure 39

While the optimized QL algorithm appears to have converge, the found policy still differs from the optimal policy in several states as can be seen in Figure 40. One state of particular concern is when the player currently has a sum of 21, a usable Ace, and the dealer is showing a 4. In this state, the QL policy identifies the best action is to hit, which clearly isn't optimal; while hitting won't cause the player to go bust, sticking would result in the maximum possible score to the player. The unadvised strategy in this state is likely the result of both infrequent visits and fortuitous positive rewards observed when visiting this state and hitting. Additional visits to this state are likely necessary for QL to learn the appropriate policy.

The many discrepancies between the QL policy and the optimal policy may indicate an inadequate exploration strategy. QL is only guaranteed to converge to the optimal policy if each state action pair is visited infinitely often. Due to the low probability of the occurrence of some states, an $\epsilon - greedy$ policy with a minimum random action value of 0.1 may result in the algorithm never taking a particular action in a particular state even over the 1,000,000 iterations of the algorithm. Furthermore, without receiving sufficient (s, a, r, s') tupples in each state, the optimum policy can't be learnt efficiently.

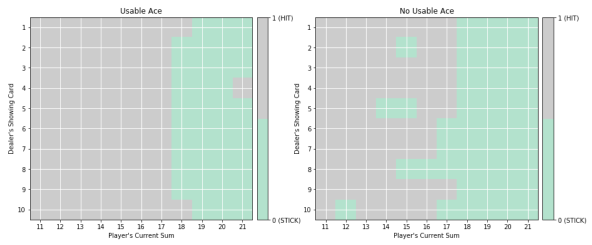

Figure 40

The tuned QL algorithm was next tested with $\gamma = 0.1$ and the resulting mean value function and clock times are plotted in Figure 41. 1,000,000 iterations of QL took 177.68 seconds, which is about the same as the execution time of the algorithm when $\gamma = 1$. This is expected as each algorithm was run for the same number of iterations. If early-stopping is considered, when the mean of the value function stops changing, the algorithm with $\gamma = 0.1$ would likely converge faster as rewards more distant in the future can be ignored thus simplifying the estimation of various states.

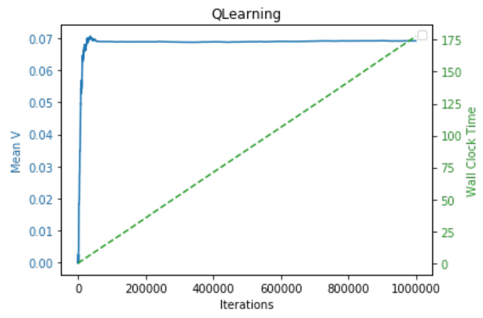

Figure 41

The policy produced by the algorithm is shown in Figure 42. The policy is very similar to the optimal policy found using PI and in total there are only 14 discrepancies between the policies. Considering there are 200 states, such indicates that in 7% of the states, the agent produces suboptimal actions. This is a significant improvement over the number of suboptimal actions the agent takes when $\gamma = 1$ and suggest the optimal policy with $\gamma = 0.1$ is much easier to learn. To further reduce this error rate, more conservative exploration strategies likely need to be considered to allow for each state action pair to be visited infinitely often.

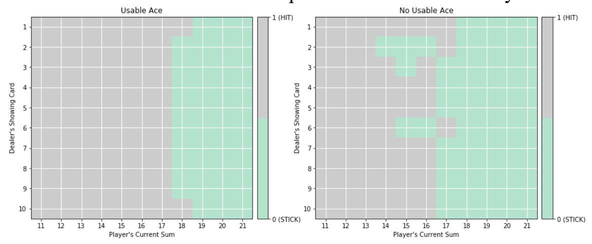

Figure 42

The optimized QL algorithm was next run on the complex version of blackjack and a plot showing the mean of the Q-function on each given iteration is shown in Figure 43. QL converges after 1,000,000 iterations as indicated by the flatlining average Q-table value. The algorithm, however, converges to a value of around 0.08, which is much lower than the optimal mean of the value function. This lower value in itself, however, doesn't indicate that the policies found by each algorithm are different; it is possible the policies derived from the Q-table are still optimal.

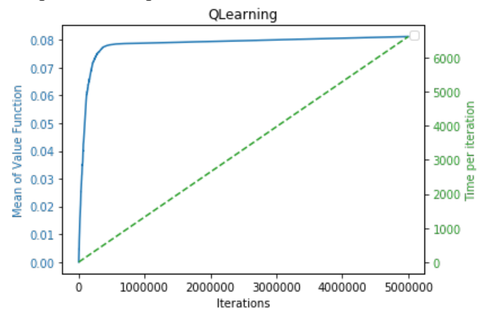

Figure 43

Figure 44 and 45 plot the policies found with QL when the current game count is -5 and +5 respectively. The policies found using QL differ significantly from the optimal policies and the strategies in many states lack logical sense. With 4,200 distinct states and the low probability of observing many game states, adequate visitation and execution of every action in each state is highly unlikely and thus the convergence of QL to the optimal policy can't be guaranteed. For example, the probability that the count is +5, the dealer is showing an Ace, and the player is delt pocket Aces is extremely rare. Furthermore, in the 5,000,000 iterations performed, it's unlikely that each action was attempted in this state more than a few times. Without exploring all possible actions enough times the optimal action in this state was impossible to learn. As a result, the policy for this state is to stick, which, based on basic intuition, is suboptimal.

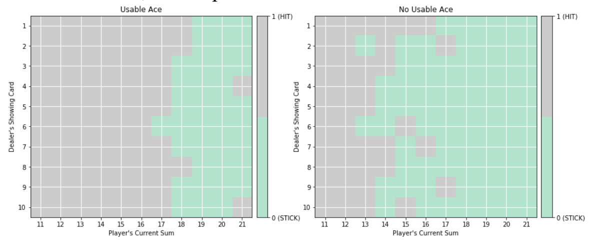

Figure 44

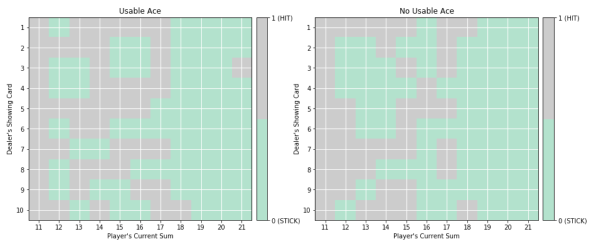

Figure 45

QL was finally tested on the complex blackjack problem with $\gamma = 0.1$. Figure 46 plots the mean value function and execution time for each iteration. The algorithm converges in roughly the same number of iterations as when $\gamma = 1$, however, the policy identified (shown in Figures 47 and 48) are much closer to the optimal policy found using PI. This is consistent with our previous findings on the simple blackjack game and again indicates that with smaller values of gamma, optimal policies are easier to learn. Moreover, when attempting to solve complex MDP's with QL, framing the problem such that the desired behavior can be learned with lower values of gamma will simplify the learning problem and help QL to identify better policies.

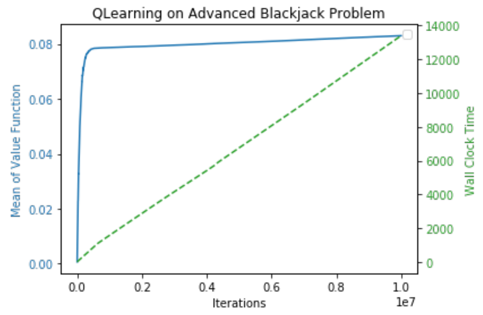

Figure 46

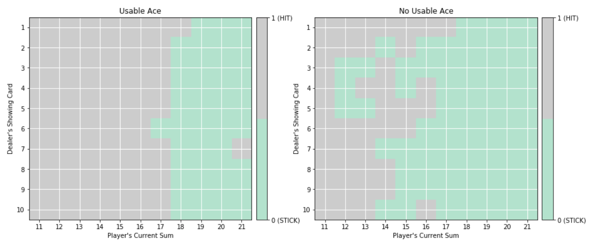

Figure 47

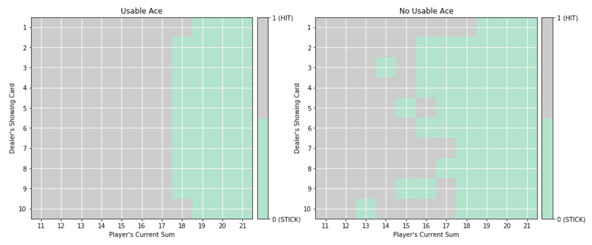

Figure 48

Overall, QL doesn't appear to scale well with increased problem sizes on the blackjack MDP. While the Q-table appears to converge, the policies produced lack logical sense and are great departures from the optimal strategy. The poor performance on the given problem is due to the extremely low probability of visiting particular states; the rarest states may have only been visited a handful of times in the algorithms 5,000,000 iterations. With very infrequent visitations to some states, an $\epsilon - greedy$ exploration strategy is likely insufficient as such further reduces the probability of exhausting all possible actions in extremely rare states. Moreover, in problems with rare states, alternative exploration strategies should be considered. In problems of this nature, using a random action selection model while learning off-policy may allow for more uniform action selections across the various states and thus lead to more appropriate optimal policies. Another alternative may be to act randomly in a particular state until that state has been visited a certain number of times. Once a state has been visited enough times the agent can proceed to explore that state using an $\epsilon - greedy$ policy. Such a strategy will also allow for more uniform actions to be taken in infrequent states and potentially lead to better strategies.

V. Conclusion

Framing a problem as an MDP is perhaps the most challenging aspect of RL and proper representation is critical to the performance of PI, VI and QL. Different MDP representations of the same problem can lead to drastically different solutions and can make learning easier or more challenging. The chosen value of gamma in particular is an important consideration when defining a RL problem. Smaller values of gamma reduce a problems complexity and result in quicker convergence for VI and PI. In addition, the policies learned using QL with smaller values of gamma were more similar to the optimal policy identified by PI. Large values of gamma serve to increase the complexity of the problem and make learning more difficult for each RL algorithm. In the extreme case when $\gamma = 1$, PI, VI and QL aren't guaranteed to converge. If a problem requires setting $\gamma = 1$, the use of absorbing states or $\epsilon - soft$ policies is necessary to enable convergence. If possible, however, a problems reward structure should be formulated such that the desired behavior is achieved with smaller values of gamma as such simplifies the learning process for all algorithms.

Some aspects of a problem, however, can't easily be manipulated and can significantly impact the performance of RL algorithms. In such situations, the choice of RL algorithm is particularly important. In general, PI may be preferred over VI in problems with large action spaces while VI tends to outperform in smaller action spaces or in problems with sparse transition functions. PI often provides efficient solutions in small MDP's and is robust to problems with a few optimal policies. With many optimal policies, however, PI may get stuck oscillating between different policies on each iteration and thus fail to converge. Early stopping, when the value function stops changing, however, can easily be applied to aid convergence. Alternatively, VI may be considered as the algorithm is naturally robust in problems with many optimal policies.

With large state spaces, both PI and VI can be painfully slow. Thus, the use of function approximators or unsupervised learning techniques such as clustering may be necessary to perform state-space aggregation and reduce the size of the problem. While combining states with similar characteristics may simplify the learning problem, it requires the assumption that states with similar characteristics should have similar optimal actions. And, if this assumption is violated, state aggregation can lead to suboptimal policies.

When the underlying transition and reward functions are unknown, model-free approaches must be considered. QL is one approach that can be applied without knowledge of the MDP. Proper parametrization of QL is necessary and can have significant impact on the algorithm's performance. The exploration strategy is a particularly important consideration. The traditional $\epsilon - greedy$, exploration strategy works well in most situations, but may fail in problems with very rare states. In these types of problems, learning off policy may be preferred. The learning rate is also an important parameter to consider. In deterministic environments, a constant learning rate of 1 is optimal, but in stochastic MDP's alternative annealing schedules are likely required.

References

[1] Bruno Scherrer. Performance Bounds for Lambda Policy Iteration and Application to the Game of Tetris. [Research Report] RR-6348, 2011, pp.54. ffinria-00185271v4f

[2] Chades I, Chapron G, Cros M-J, Garcia F & Sabbadin R (2014) 'MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems', Ecography, vol. 37, no. 9, pp. 916–920, doi 10.1111/ecog.00888.

[3] Mahajan, P. (2019, January 14). Playing Blackjack using Model-free Reinforcement Learning in Google Colab! Retrieved November 22, 2020, from https://towardsdatascience.com/playing-blackjack-using-model-free-reinforcement-learning-in-google-colab-aa2041a2c13d

[4] Watkins and Dayan, "Q-Learning." (1992)