Sahil Gandhi

Programming Assignment 4
CS 6140 – Christopher Amato

Q1: Domain

The domain consists of a 2-dimensional square grid. The start and the terminal states are fixed to the lower-left and upper-right respectively. A sample grid of size 4 looks like this:

```
0    1   2   3(T)
4    5   6   7
8    9   10  11
12(S) 13  14  15
```

Each number is what the state is identified with; 3 being the terminal state and 12 being the start state in this example. The reward for being in any non-terminal state is -0.1 and that for being in the terminal state is 5. 4 actions are allowed, and the domain has fixed probabilities for whether the chosen action will actually be the performed action or not. This adds a stochasticity in the world and helps the agent explore un-seen portions.

Q2: Code

The code comprises of one single class. The class includes implementation for both Q-learning and Sarsa (λ).

- The constructor initializes all the parameter values and also initializes the grid, Q-values and eligibility choices.
- The *learn* method runs either of the algorithms for 1 experiment (n_episodes) and prints the Q-values and the learned optimal policy.
- Methods *_evaluate_action_with_env_probabilities* and *_take_action* are just wrapper functions for the gridworld.py 's internal implementations.
- Method *_choose_action_from_policy* computes the ϵ-greedy policy for the given state
- Methods *_q_value_computation* for both algorithms do the q-value and e-value updates based on the formula
- The main algorithm is found in methods *_sarsa* and *_q_learning*
- Rest static methods and module methods are simply for plotting automation and running the algorithm for *n_experiments* number of times
- The arguments from the terminal are implemented using the python *argparse* module

Q3 & Q4: Q-Learning and Sarsa(λ)

Problems with code:

Majority of my code works correctly, however, one part isn't working as expected for which I've added a manual hack to ensure I get the right answer. While computing the reward of from the terminal state; as a consequence of the original implementation, I would get really large Q-values. I suspected that the final reward of +5 was counted twice at each iteration leading to the disproportionate Q-values. I changed the code to return a +4.9 in the terminal state and disregarded the subsequent rewards if the agent changed state from the terminal state to the terminal state. This fixed my error and I now have Q-values proportional to the maximum positive reward of +5.

Algorithm specific analysis.
In this assignment, we implement 2 algorithms; namely Q-learning and Sarsa(λ). Q-learning is an off-policy TD control algorithm and Sarsa(λ) is an on-policy method using eligibility traces.

Q-learning algorithm:

$$Q(s_t, a_t) = (1 - \alpha)Q_t(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \right]$$

In this case the learned action-value function, Q, directly approximates, Q^*, the optimal action-value function independent of the policy followed. The policy still has an effect in that it determines which state-action pairs are visited and updated.

The variables (hyper-parameters) that are affect the efficiency of the algorithm are α, ϵ & γ.

- α represents the step-size or the learning-rate of the algorithm. It determines the amount with which the future reward and the discounted one-step expectimax of the next Q-state propagates back to the current Q-state.
- ϵ is the parameter for the default-policy; the ϵ-greedy policy. This policy helps balance the exploration v/s exploitation strategy. We randomly pick a state with ϵ probability and pick the best state (one-level expectimax) with a $1 - \epsilon$ probability.
- γ is the discount factor. It decides the level with which rewards in future states affect the Q-values of the current state. If γ is low, then the less reward will be propogated back to the start state and vice-versa.

Analysis of Q-learning parameters:
I ran 500 experiments of 500 episodes each for varying values of parameters. The analysis of each is shown below.

Analysis for α:

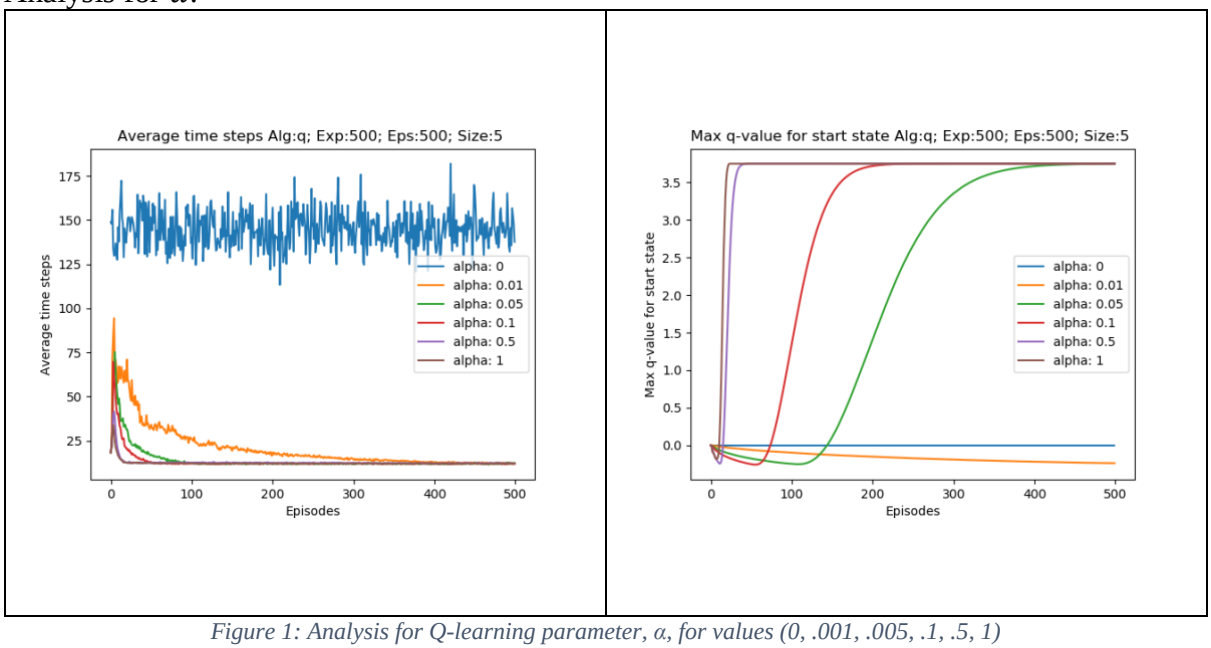


Figure 1: Analysis for Q-learning parameter, α, for values (0, .001, .005, .1, .5, 1)

From the above graph, we can conclude the following points:

- α = 1
    - Average step count per episode is very high for all episodes. Large alpha means the learning-rate is 1. What this means that at each expectimax calculation we are essentially replacing the current Q-state with $\alpha * (r_{t+1} + \gamma Q(s_{t+1}, a))$ as $Q(s,a) - \alpha Q(s,a) = 0$.
    - Because of this, the Q-values never converge to the optimal policy and the agent only randomly explores the domain till it reaches the terminal state.
    - Because of this, the reward of the terminal state does not propogate back to the Q-values of the 1st state as the agent takes a lot of time to reach the terminal state. This results in the per-state accrued penalty to be much higher than the final terminal state reward.
- For the rest of the values:
    - As we decrease alpha, the agent keeps learning at each iteration of the episode in contrast to the previous random exploration. The optimal policy is updated and reaches a convergence eventually.
    - Different values result in different convergence times and looking at the graph we can approximate that 0.1is the best value for alpha. It serves as a decent trade-off between a large and a small step size.

Analysis for ϵ:

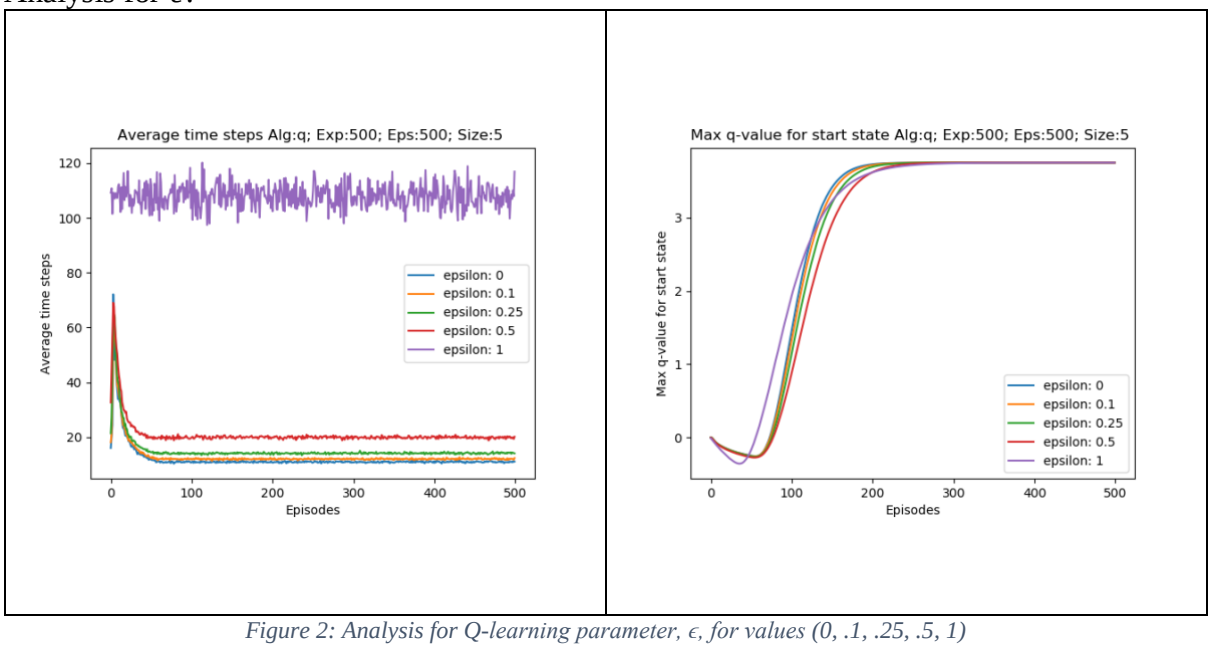


Figure 2: Analysis for Q-learning parameter, ϵ, for values (0, .1, .25, .5, 1)

From the above graph we can observe:

- ϵ represents the greediness of the baseline policy. The higher the value, the more explorative the policy is. As we can see, for epsilon=1, we get a complete random strategy and hence the average step count per episode is always high.
- Interestingly, even though the baseline policy is random, the max Q-value of the start state is comparable with that of the other epsilon values. This shows us that even though the baseline strategy was to only explore randomly, we will still converge and that too with a good enough reward.

- Nonetheless, the additional cost of steps per episode seems an over-kill for any learning algorithm.
- Looking at the graph, epsilon value of 0.5 seems like a good fit. What we're saying is that we should explore and exploit with an equally likely probability.
- This fits really well because we don't know the optimal policy a-priori.

Based on the above analysis, we choose the following parameters as the best fit:

$$\alpha = 0.1;\ \epsilon = 0.5 \text{ (rest remain default)}$$

Using these chosen parameters, we run the algorithm on a 10x10 grid. The results are shown below:

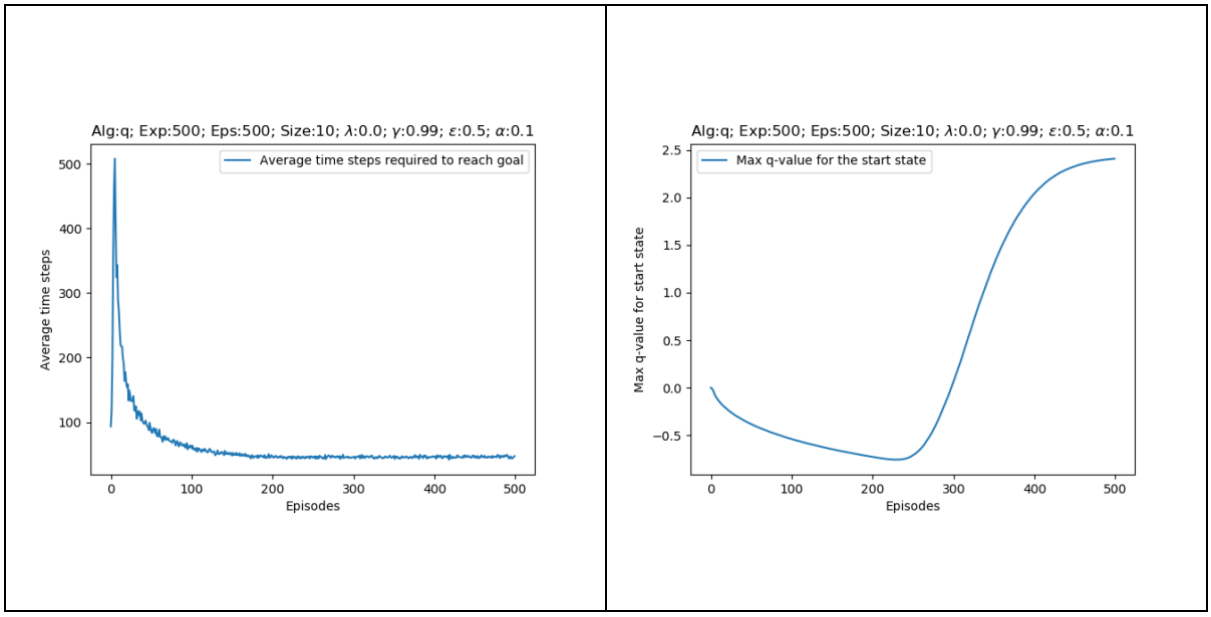


Figure 3: Analysis for Q-learning on a 10x10 grid based on chosen parameters

Looking at the graph above:

- We can conclude that the choice of our parameter was more-or-less correct.
- Alpha seem to be exactly correct as any increase in it would reduce the propagation of the Q-value to the start state
- Epsilon could have been reduced slightly to get a micro improvement in step count.
- The reason for this is that, our original analysis was done on a 5x5 grid. The state-action space was exponentially smaller than that present in the 10x10 space.
- Due to this, a large value of epsilon seems to explore even with a high probability even when the optimal policy has been converged to.
- This results in added penalties, and the eventual terminal reward's value is reduced drastically
- This is the reason why the Q-value are negative for the first 200 episodes and also why the first few episodes take so long to converge to the terminal state
- After the initial exploration, the agent converges to the terminal state at around 150th episode, but the optimal policy is reached much later at around the 320th episode.

Sarsa (λ) algorithm:

Sarsa (λ) is an extension of the normal Sarsa algorithm but using eligibility traces. Eligibility traces are one of the basic mechanisms of reinforcement learning. λ refers to the use of an eligibility trace. The algorithm, combined with an eligibility trace obtains a more general method that may learn more efficiently. There are 2 ways to view eligibility traces, the forward view and the backward view.
The idea in Sarsa (λ) is to apply the temporal difference of λ to every state-action pair than to states. Hence, now we need a trace for every state-action pair.
Let $E(s,a)$ denote the trace for state s and action a; then

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t E_t(s,a) \;\; \forall\, s,a$$

$$\text{where, } \delta_t = r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$$

$$\text{and } E_t(s,a) = \begin{cases} \gamma\,\lambda\,E_{t-1}(s,a) + 1 & if\ s = s_t, a = a_t \\ \gamma\,\lambda\,E_{t-1}(s,a) & otherwise \end{cases} \quad \forall\ s,a$$

Sarsa (λ) is an on-policy algorithm, meaning it will approximate to $Q^{\pi}(s,a)$ for the current policy π. It will improve the policy gradually based on the approximate values of the current policy. For policy improvement, we will use the ϵ-greedy method to form a balance between exploration and exploitation.

Analysis of Sarsa (λ) parameters:
I ran 500 experiments of 500 episodes each for varying values of parameters. The analysis of each is shown below.

Analysis for α:

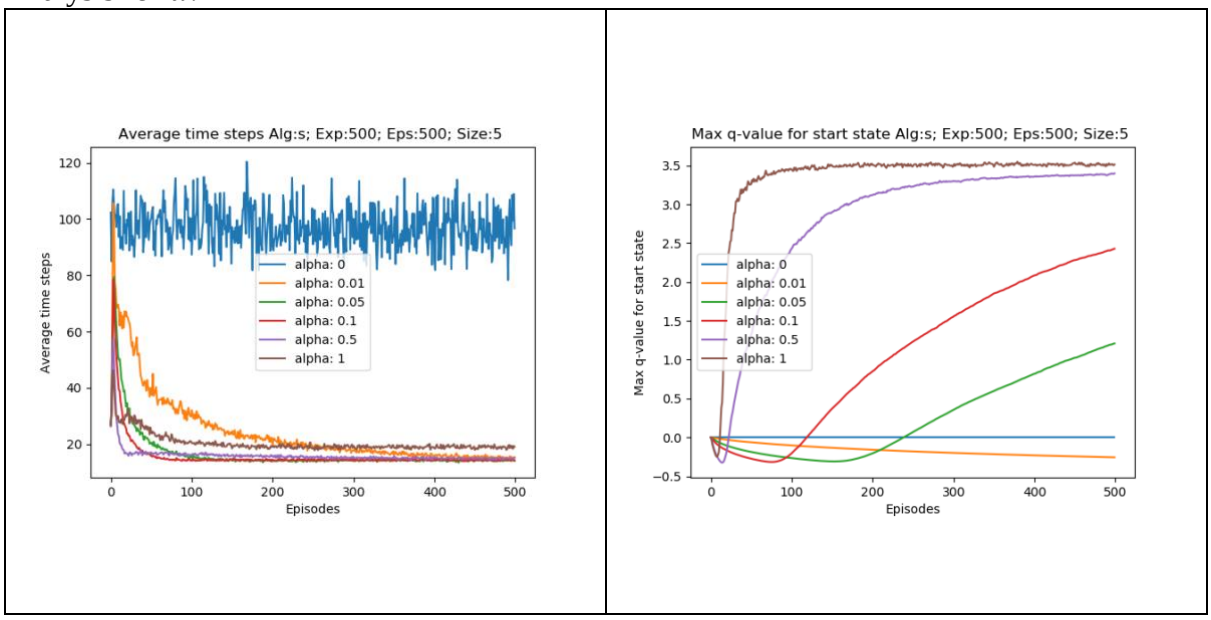


Figure 4: Analysis for Sarsa (λ) parameter, α, for values (0, .001, .005, .1, .5, 1)

From the above graph we can observe:

- Just like for Q-learning, alpha=1 is very random and does not provide a good balance
- We can see that the best value seems to be 0.5. It serves as a good balance by keeping the step count low as well as propagate maximum reward back to the start state as soon as possible.

Analysis for ϵ:

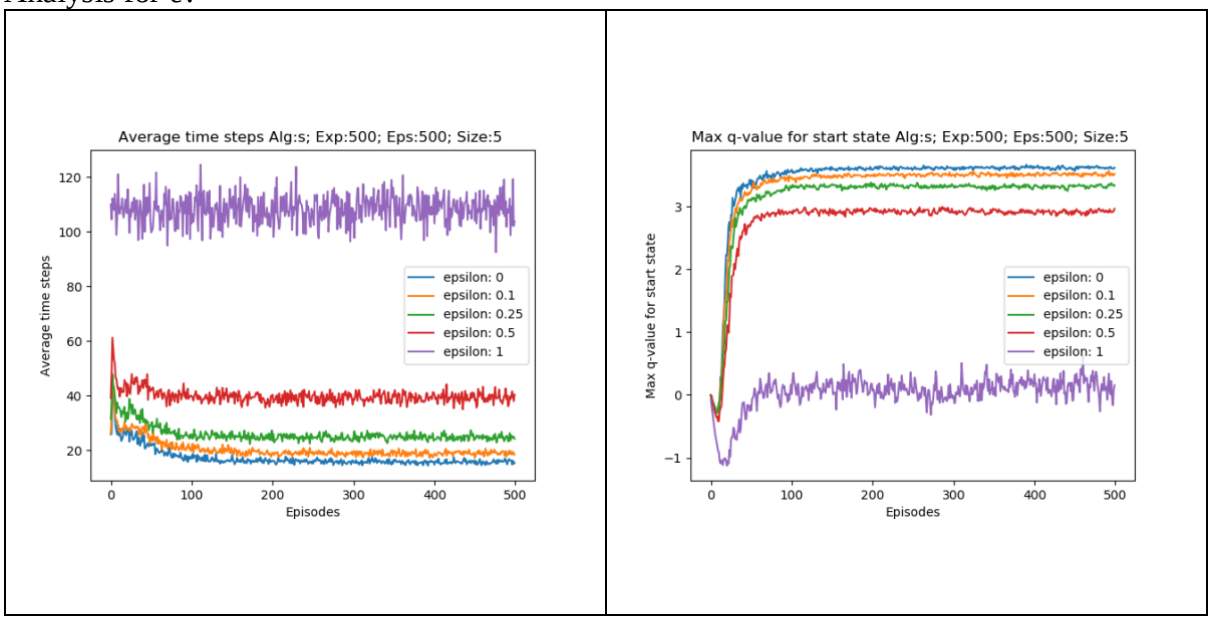
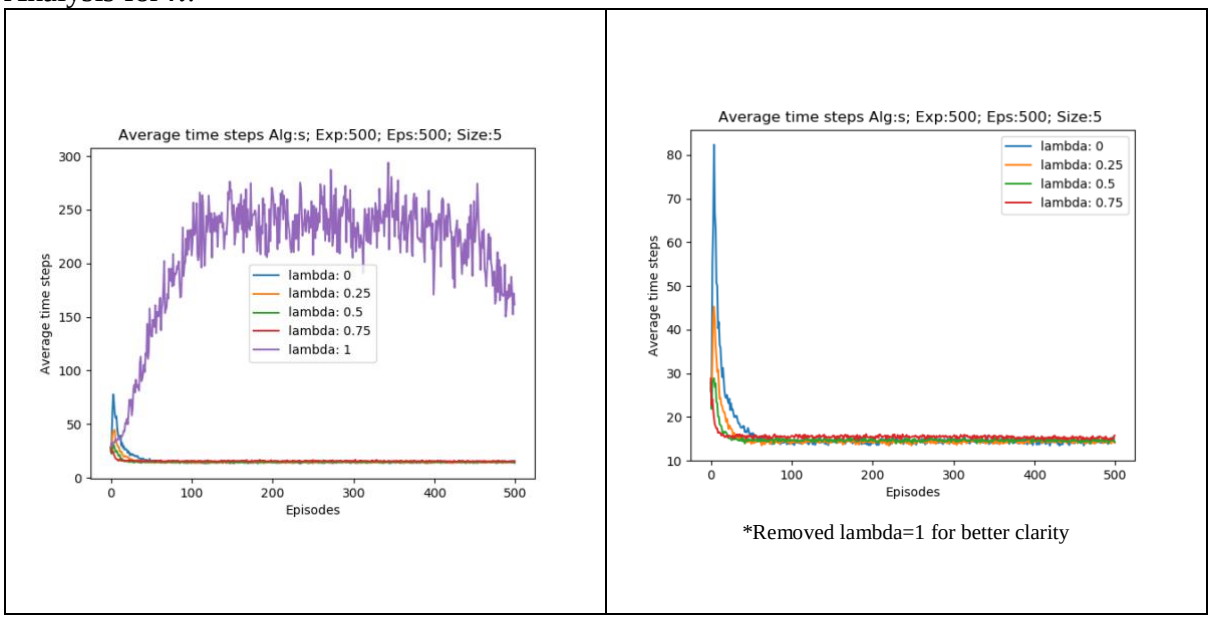


Figure 5: Analysis for Sarsa (λ) parameter, ϵ, for values (0, .1, .25, .5, 1)

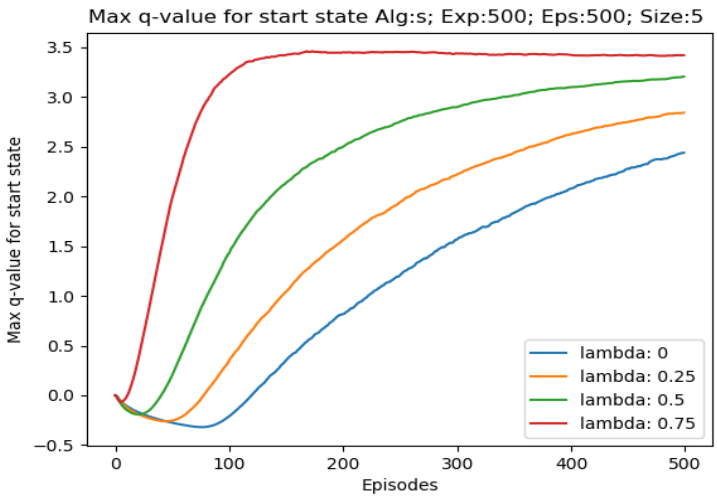
From the above graph we can observe:

- We use an epsilon-greedy policy as our base policy to serve as a balance between exploration and exploitation.
- As we can see that epsilon values for 0 and 0.1 serve very good results, but keeping epsilon 0 means, we are instructing the algorithm to only exploit and not explore at all.
- To keep room for stochasticity and allow the algorithm to explore with some probability, the best value lies between 0.1 and 0.25.

Analysis for λ:



*Removed lambda=1 for better clarity



*Removed lambda=1 for better clarity

Figure 6: Analysis for Sarsa (λ) parameter, λ for values (0, .1, .25, .75, .5, 1)

Looking at the above graphs:

- We can clearly conclude that a higher lambda is a good estimate for the Sarsa algorithm
- This is also fits well with the theory of Sarsa as a large value of Sarsa allows for a longer time difference chain
- However, like all other parameters, there is a cut-off beyond which the increasing the value prohibits the algorithm from working efficiently
- For our example, the optimal value for lambda seems to be around 0.75

Based on the above analysis, we choose the following parameters as the best fit:

$$\alpha = 0.5;\ \epsilon = 0.1; \lambda = 0.75 \text{ (rest remain default)}$$

Using these chosen parameters, we run the algorithm on a 10x10 grid. The results are shown below:

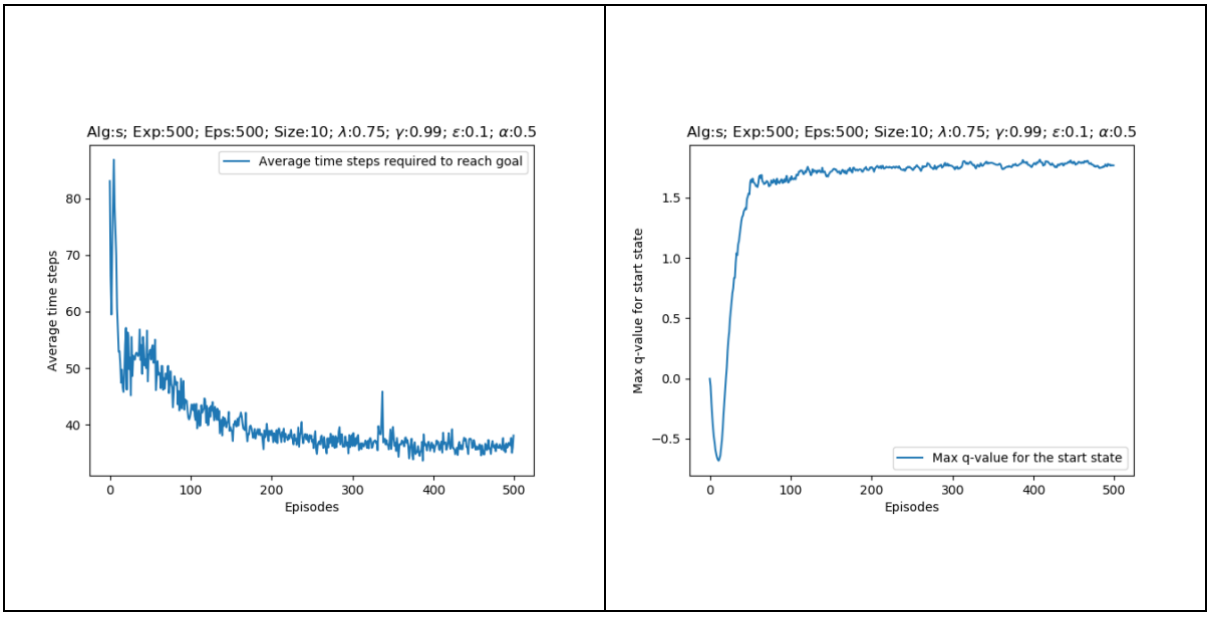
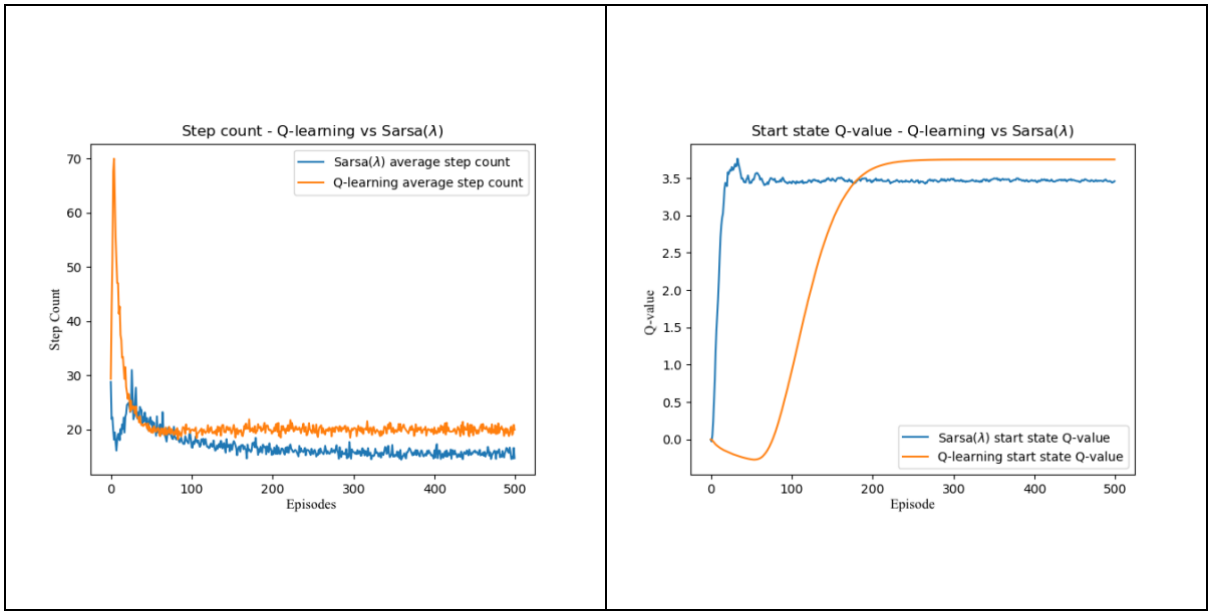


Figure 7: Analysis for Sarsa(λ) on a 10x10 grid based on chosen parameters

Looking at the above graph, we can conclude the following about Sarsa (λ) algorithm:

- The estimate for our parameters ($\alpha = 0.5;\ \epsilon = 0.1; \lambda = 0.75$) were on-point.
- Sarsa(λ) algorithm does not require much of exploration as it tries and converge to the optimal policy based on its eligibility choice (E_t), hence, the epsilon value of 0.1 works very well, and despite using a strategy that has low exploration probability, the policy converges very quickly to the optimal policy.
- Having large values for alpha and lambda also work in the favor of the algorithm.
- The algorithm takes an average of 200 episodes to converge to the optimum policy

Comparing Q-learning and Sarsa (λ) together, based on selected parameters



The above graph shows the comparison between Q-learning and Sarsa (λ) run on their respective chosen parameters. This plot aligns the theory in the sense, that

- Q-learning does more exploration, due to which its initial episodes are very sporadic and takes a longer time to converge
- Sarsa (λ) does little exploration and is its step counts per episode are lower from the start
- Because Q-learning does enough exploration, it is capable of converging to a much better solution (as shown in the graph on the right); even if it takes a little longer than Sarsa (λ)