CS747 Programming Assignment 3

Drumil Trivedi 170020016

Windy Gridworld

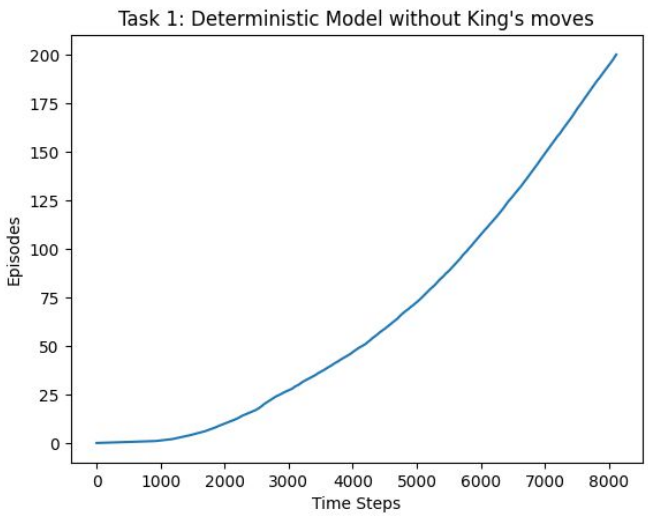
Implementation -

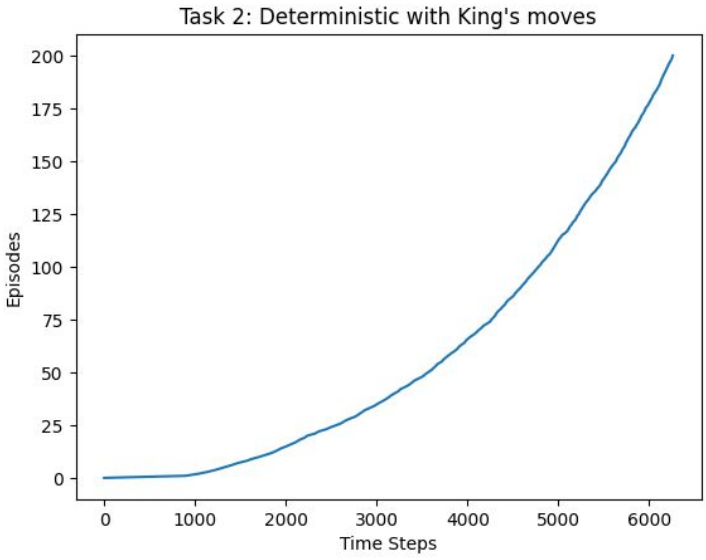
To take a step corresponding to an action, first I am calculating the “effect” of the action on the agent. For example, if the action is to move towards North-West (or Upper-Left), then the effect is (1,-1). Then, I add the effect of the wind. For example, if the wind in the current column is 1 in the North (or Upward) direction, then the effect of the wind is (1,0), thereby making the combined effect = (2,-1). I then add this to the current position. For example, if the current position is (r,c), then the resulting position will become (r+2,c-1). If the new position overflows the grid, I clamp the values between 0 and n-1. where n is the number of rows or columns. For example, if the updated position is (n+1,-1), the position will be clamped to (n-1,0). This is the final updated position, or the next state.

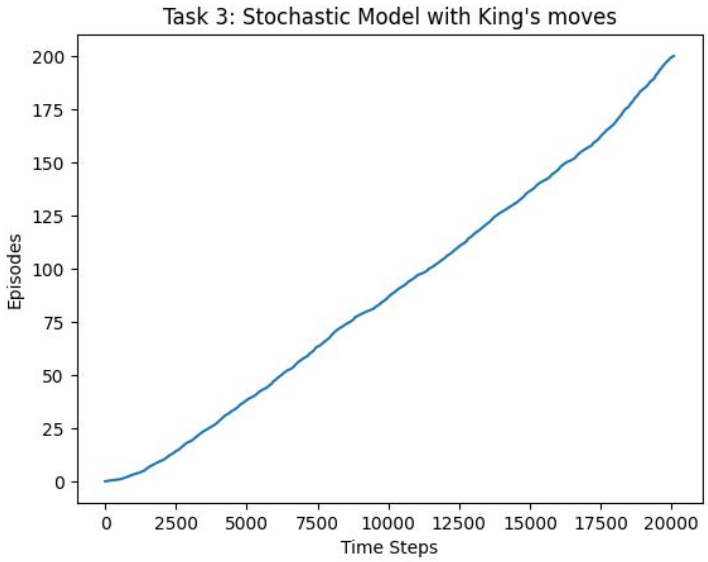
Tasks Considered -
Task 1 -(Base Case) Deterministic 4 move agent
Task 2 - Deterministic 8 move (Kings Move) agent
Task 3- Stochastic 8 move (Kings Move) agent
Task 4- Deterministic 9 moves (Kings Move plus stay at the same place only affected by wind)
Task 5- Stochastic 4 move agent

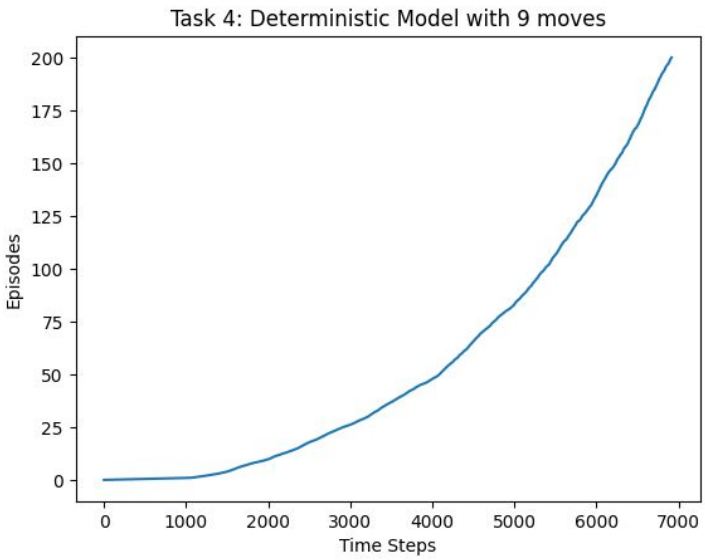
(I wasn't sure if Task 4,5 were asked but since they were mentioned in the text I have incorporated in my experiments as well)

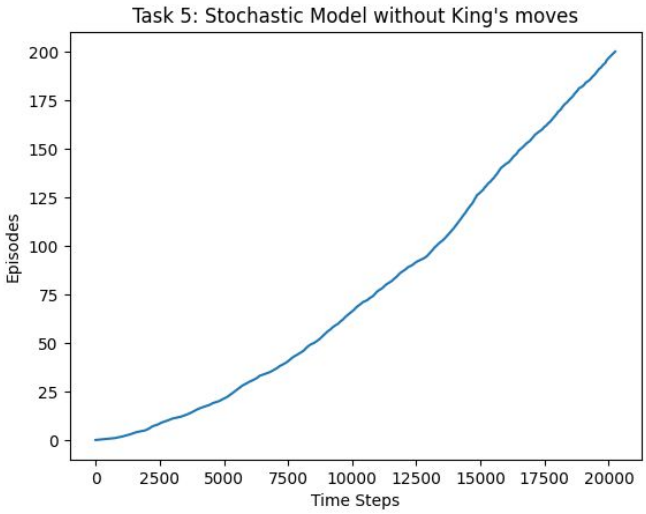
Graphs for Individual Tasks making use of Sarsa(0):











Observations wrt to Tasks :

We see that for all the tasks, the plots get increasingly steeper, i.e. the slope keeps on increasing. This shows that the number of timesteps per episode keeps on increasing as we are training. This is expected since as the agent learns, the estimated Q moves closer to the actual values, therefore, the agent starts taking optimal actions to minimize the path length.

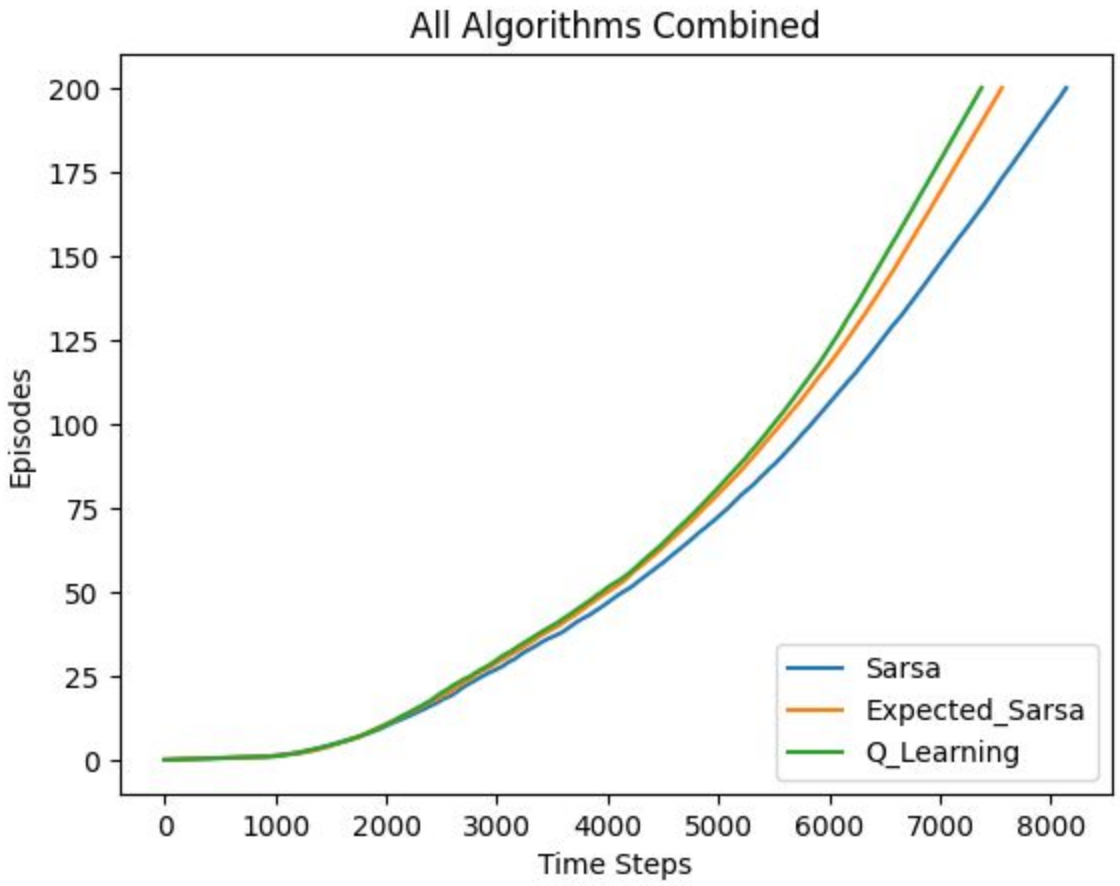
Furthermore, we can observe from the plot for Task 3 and 5, that the number of steps per episode is the highest. This is because even though the agent learns the optimal action successfully, it is not guaranteed to be able to reach the desired state because of stochasticity. This makes it take more steps per episode than the optimal case.

Finally, if we compare Tasks 1 and 2, we see that the number of steps per episode is higher for Task 1 as compared to Task 2. This is because in Task 2, the agent has a higher number of

actions. For example, if an agent has to take 1 step in both left and upward directions, it would have taken him 2 time steps, whereas it would take only 1 time step in Task 2 (assuming no wind in that column). This logic can similarly be extended for the case when wind isn't 0.

However, it might be slightly surprising to see that despite having another action (to stay at the same place), task 4 has a higher number of time steps per episode. This is possibly because the new action doesn't provide the agent with more power. Still, in a separate environment, having the possibility of not taking an action can in fact decrease the number of timesteps. Consider the case the start state is exactly 1 grid below the goal state and the wind in this column is 1. If the agent has the option of not taking an action, then it will reach the state in 1 time step. However, if that action is not available, it will have to take a longer route.

Base Case over different algorithms:



We see that lower episodes the graphs are close and then diverge to an ordering where, Q Learning has the least average time steps taken followed by Expected Sarsa and then Sarsa(0).
Making the ordering for rate of learning as Q Learning >Expected Sarsa > Sarsa(0).

I took several seeds and the graph looks the same (slight shifts occur but the graph ordering of algorithms) is the same. Secondly the values of $\alpha = 0.5$ and $\varepsilon = 0.1$ are used , if we vary these parameters the combined graphs look different, since they directly affect the values we plot.

They reason why Sarsa underperforms is that it depends on the epsilon greedy policy to evaluate the value function which introduces noise in this deterministic task , whereas as Q learning updates it Q function using the best action from the next state. It's hard to reason why this exactly happens but the setting of the MDP to be learnt is such that the ordering occurs.

Another interesting thing I observed was that if we increase the reward we provide on reaching the final state the Sarsa graph takes more time steps on average and expected sarsa becomes the best performer(maybe due to random seed because Q learning again becomes better at higher values or r)

E.g. If we set reward to 100 we get-