CS747: Assignment 3

Shubham Anand Jain, 17D070010

November 12, 2020

1 Gridworld comparisons across bootstrapping methods

In this assignment, we implemented the Windy Gridworld from Exercise 6.5 of Sutton and Barto (2018) and compared the performance across various bootstrapping methods, which include Sarsa(0), Expected Sarsa and Q Learning. Importantly, all our code is modular and the parameters are orthogonal; thus, any combination of parameters can be given to the code (not just king moves/king moves & stochasticity etc as expected).

To run, our main file **windy_gridworld.py** takes the following parameters as input:

1. –**algorithm**: one of **q_learning, expected_sarsa, sarsa0**
2. –**stochastic**: A boolean variable, given as 0/1; where 1 denotes that the wind is stochastic in nature
3. –**kingmove**: A boolean variable, given as 0/1; where 1 denotes that king moves are allowed (i.e, allowed to move in 8 directions rather than 4)
4. –**episodes**: The number of episodes to run for
5. –**randomSeed**: The random seed to the algorithm, to ensure reproducibility
6. –**epsilon**: The epsilon value to be used, since we are following an epsilon-greedy policy
7. –**alpha**: The alpha value to be used in the bootstrapping update

Our wrapper script is given in **get_plots.py**. It plots all the three algorithms with the same parameters, averaged over a number of iterations (which is taken as input). To plot only a single algorithm, the "algorithms" array can be changed without any loss to the rest of the program.

Additionally, the code boasts great flexibility - For example, the **control_instance.py** file which acts as the environment takes in the wind, start point, end point, size of the grid, stochastic additions of wind etc as arrays. These can be easily changed and we can get a variety of results by playing around with this program. We will discuss specific plots in the relevant sections below.

1.1 Windy Gridworld parameters & edge cases

For all the experiments in this document, $\epsilon = 0.1, \alpha = 0.5$, number of episodes = 200, number of random seeds averaged over = 50. For edge cases, in general we follow this approach: we move to whatever final position comes to be, even if it is out of the grid; and then do $fin_x = max(0, min(fin_x, size_x))$, and the same with y; basically, we clip the final position to lie inside the grid. This is true across all our experiments, including king moves and stochastic wind.

An interesting thing to note is that, this idea of ours results in the fact that any cell can always reach the end cell with non zero probability. Thus, we never really have to terminate an episode after a large number of timesteps - we keep it going until it reaches the end-state, since it is possible to do so from every state. Thus, we avoided this edge case.

Our Gridworld instance gives a reward of -1 at every step, as mentioned in the book. In the code, we pass reward as $+1$ to indicate that we have reached the final state (and thus save sending an extra bit of information), but for the actual updation, on looking closely, we consider it as -1 only.

Figure 1: Relative performance of agents

1.2 Windy Gridworld

In the normal windy gridworld, we observe results very close to the ones given in Sutton and Barto for Sarsa, in Exercise 6.5.
We see that Q-learning performs the best, which is expected since it converges to the optimal policy value function, instead our policy's value function. Sarsa performs worse than expected Sarsa, which is also in line with our intuition, since expected Sarsa takes into account all the possibilities that can occur.

1.3 Windy Gridworld with king moves

In Windy Gridworld with king moves, we observe that the number of timesteps taken per episode falls significantly from Windy Gridworld with just 4 actions. Thus, the additional flexibility in terms of moves offers a strikingly visible improvement in performance of the agent.

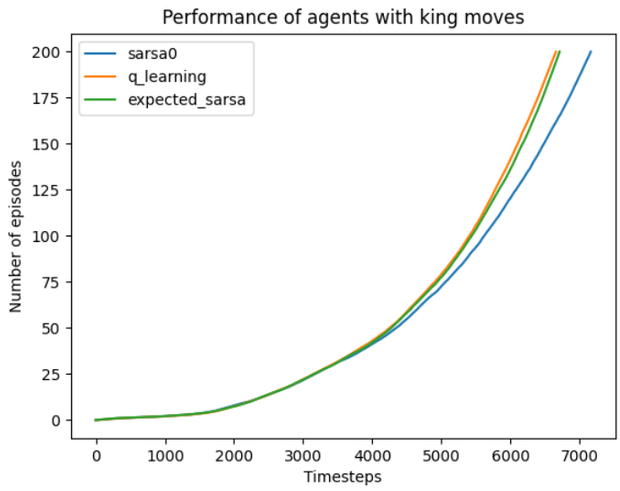

Figure 2: Relative performance of agents with king moves

1.4 Windy Gridworld with stochasticity

In this section, we consider both stochasticity in wind, and king moves. For the stochasticity in wind, with $\frac{1}{3}$ probability we add wind as $+1, 0, -1$ each. Considering our above policy of edge-cases, what this means is that, for example, if we are on an edge wrt the y axis, with 0 wind; then the probability of $+1, 0$ wind mean that we end up in the same cell, with probability $\frac{1}{3} + \frac{1}{3} = \frac{2}{3}$, instead of $\frac{1}{2}$ which is a possible alternate implementation.
Q-learning here continues to dominate over the other bootstrapping algorithms, due to reasons we already mentioned - it converges to the value function of the optimal policy, and not the

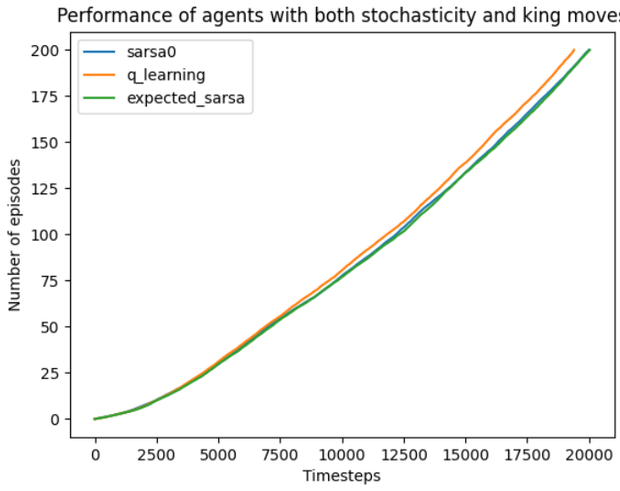

Figure 3: Relative performance of agents with king moves

policy that we are testing. The difference between Expected Sarsa and Sarsa is less clear; indeed, they seem to be very similar in terms of performance. This can be explained by understanding that Expected Sarsa tends to reduce the variance in the results, thus it performs well in most scenarios. However, adding stochasticity externally seems to showcase the difference in terms of where Sarsa and Q learning bootstrapping methods are expected to converge. (Basically, the added stochasticity removes the difference between Sarsa and Expected Sarsa methods).

2 Inferences and Observations

1. In general, we expect the performance to be as follows: Q-learning $\geq$ Expected Sarsa $\geq$ Sarsa, which is corroborated by the results (lower timesteps = better performance).

2. The amount of timesteps is as follows: King moves + stochasticity $\geq$ 4-moves $\geq$ King moves. Thus, stochasticity drastically increases the amount of timesteps taken, whereas king moves tend to reduce the amount of timesteps (as it offers more freedom). This is because stochasticity makes the end-time very uncertain; even if we can reach the end-state in one step, there is a $\frac{2}{3}$ probability that we might not reach it! Thus, we expect the number of timesteps to increase roughly by a factor of $\times 3$, by a back-of-the-envelope calculation. This turns out to match the experimental results very, very well!

3. The difference between Expected Sarsa and Sarsa is almost non-existent with stochastic wind. This is because here the stochastic wind increase the variance, and thus eliminates the advantage of Expected Sarsa in reducing variance. The value function of the policy they converge to tends to be much more important here.