CS747 Programming Assignment 3 Report

Tezan Sahu — 170100035

November 10, 2020

The code for this assignment has been modularized into the following files:

- `envs.py` file implements the Windy GridWorld, along with its variants that include King's moves & stochastic winds
- `agents.py` file contains implementations of the Sarsa, Q-Learning & Expected Sarsa Learning agents
- `experiment.py` file implements the experimental setup for running a learning agent in an environment for a fixed number of episodes, and return relevant data (episode lengths, time steps & episode rewards) needed for generating plots
- `main.py` script, to run simulations, gather data & generate plots

1 Task 1

The Windy GridWorld has been implemented through the `WindyGridworldEnv` class in the `envs.py` file. Following are the key aspects of the implementation:

- The grid is implemented using `position` & `state`. Here, `position` refers to the coordinates of a block, starting from $(0, 0)$ (`state` $= 0$) at top-left corner & $(6, 9)$ at the bottom-right corner (`state` $= 69$).
- Wind strengths are implemented by specifying the values 0, 1 or 2 in the `winds` variable for the corresponding columns.
- The `_limit_coordinates(...)` internal function is used to restrict the final states during a transition within the bounds of the grid.
- The `_calculate_transition_prob(...)` is used to compute the final states after transition, following the wind strengths & being bounded within the grid limits. This returns a tuple containing (probability, new state, reward, flag to indicate end of episode).
- For every step take, a reward of -1 is assigned

- Since transitions are deterministic, the probability of a transition from one state using an action = 1 (to the fixed next state)
- Each episode starts at `position` = (3,0) & is said to end on reaching `state` = (3,7).

2 Task 2

The Sarsa(0) Agent has been implemented using the `SarsaAgent` class in `agents.py`. By default, it has 4 moves (UP, DOWN, RIGHT & LEFT).

Here, the agent follows an epsilon-greedy strategy, with $\epsilon = 0.01$, along with the SARSA update using $\alpha = 0.5$ (constant throughout the episode) & $\gamma = 1$. After averaging over 20 seed values (0-19), following are the plots for the same:

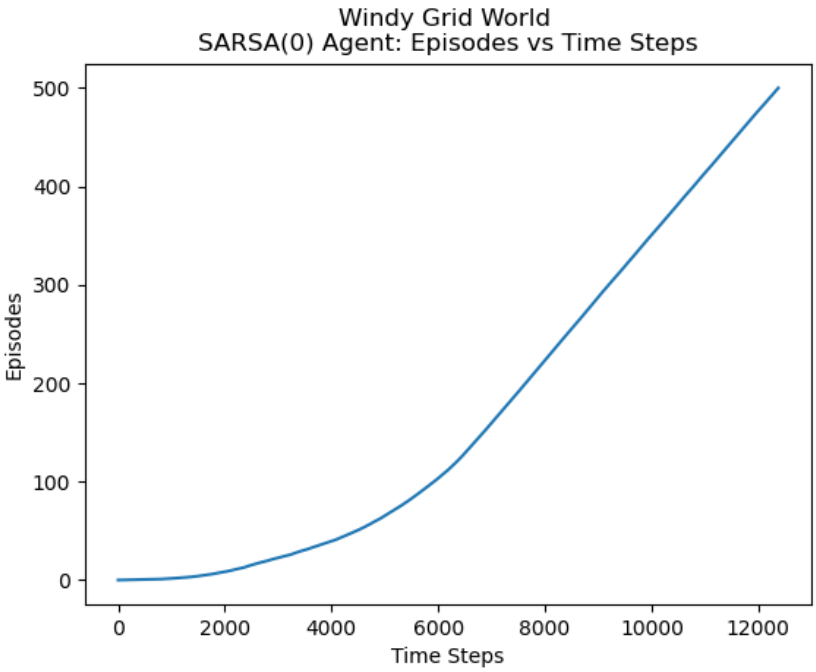
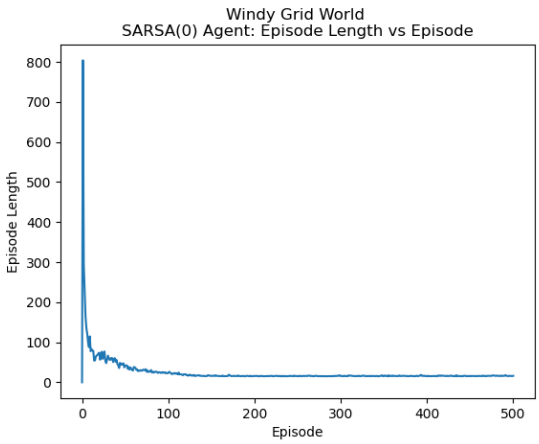


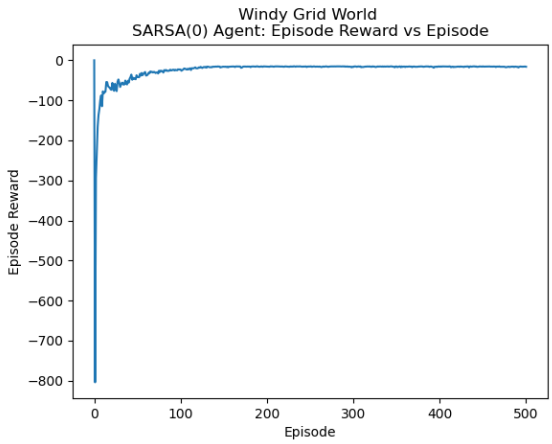
Figure 1: Baseline Episodes vs Time Steps plot for Sarsa(0) agent

Initially, when the agent does not have a good estimate of the optimal policy, it takes long time to reach the goal state. This justifies the large number of time steps taken to reach the goal states for the beginning few episodes. As the number of episodes increase the agent learns better policy and those states which occur on the optimal path are visited more frequently and we obtain better estimates for these states.

Since the reward for each step is -1, the Episode Lengths & Rewards plots are essentially flipped versions of each other. Initially, the Sarsa(0) agent took $\sim$ 800 steps to complete an episode, but gradually, converged to near-optimal number of steps. Continued ϵ-greedy exploration kept the average episode length at about 17 steps, two more than the minimum of 15.



(a) Episode Lengths for Sarsa(0) agent



(b) Episodic Rewards for Sarsa(0) agent

3 Task 3

For this task, we set `kings_move_allowed = True`, which now allows the Sarsa(0) agent to make 8 moves: (UP, RIGHT, DOWN, LEFT, UP-RIGHT, DOWN-RIGHT, DOWN-LEFT & UP-LEFT).

Here also, the agent follows an epsilon-greedy strategy, with $\epsilon = 0.01$, along with the SARSA update using $\alpha = 0.5$ (constant throughout the episode) & $\gamma = 1$. After averaging over 20 seed values (0-19), following are the plots for the same:

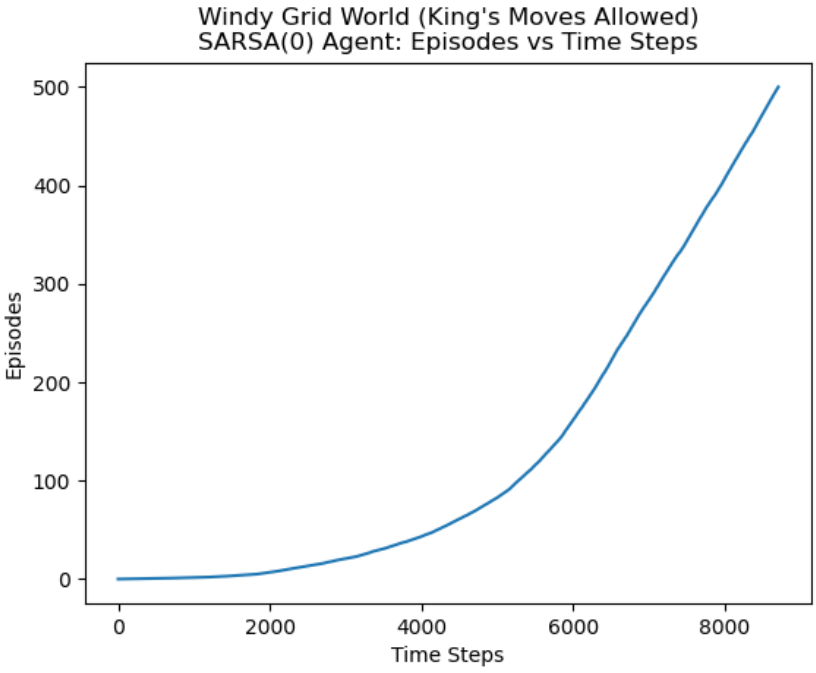
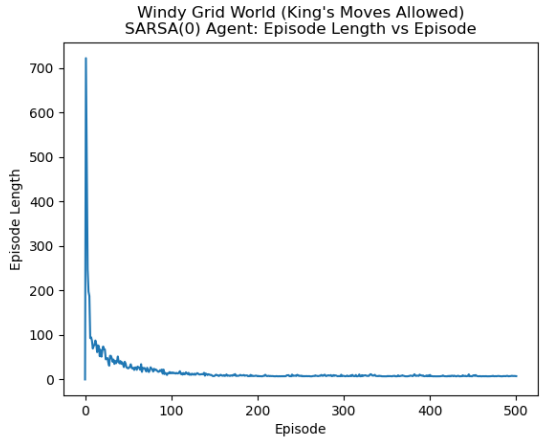


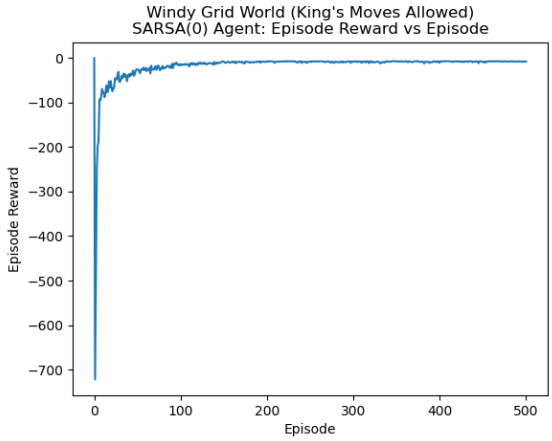
Figure 3: Baseline Episodes vs Time Steps plot for Sarsa(0) agent with King's Moves

Iterating over 500 episodes, we are able reach the goal state in minimum of 7 time steps

per episode. Continued ϵ-greedy exploration kept the average episode length to slightly greater than 7 steps. Here, we observe that the Sarsa(0) agent took ~ 700 steps to complete an episode, but gradually, converged to near-optimal number of steps.



(a) Episode Lengths for Sarsa(0) agent with King's Moves



(b) Episodic Rewards for Sarsa(0) agent with King's Moves

4 Task 4

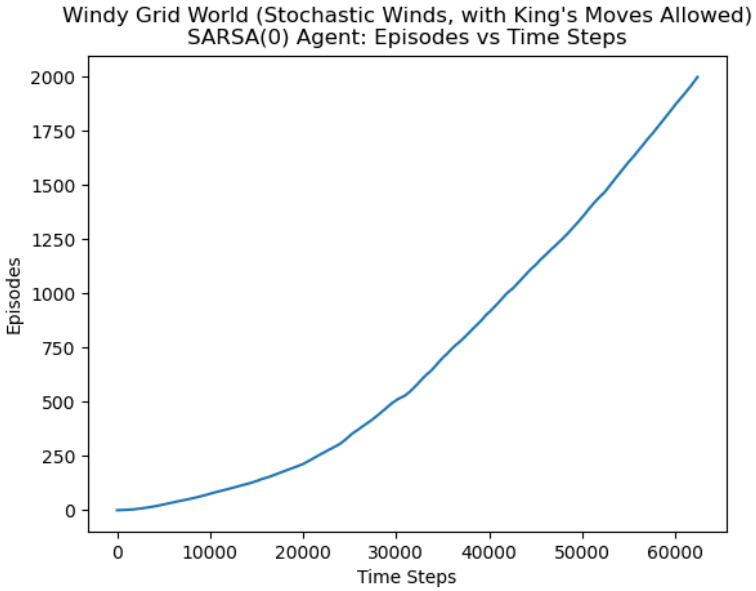


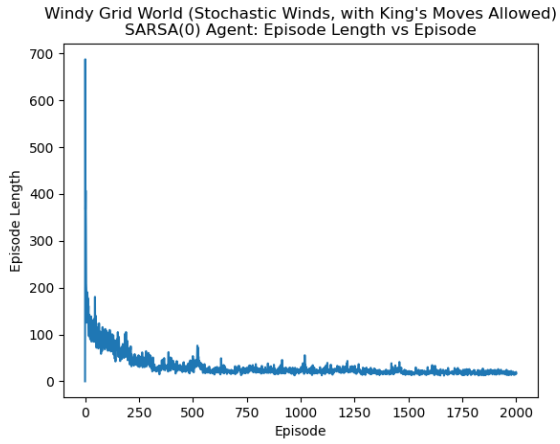
Figure 5: Baseline Episodes vs Time Steps plot for Sarsa(0) agent with King's Moves & stochastic effects of winds

For this task, we set `kings_move_allowed = True`, which now allows the Sarsa(0) agent to make 8 moves, & also set `stochastic_wind = True`. This enables the effect of the wind,

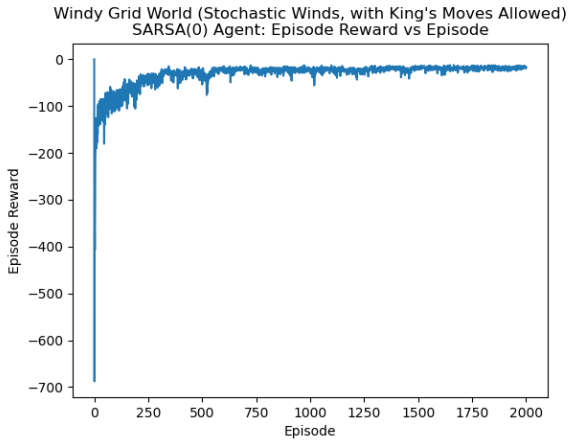
if there is any, to be stochastic (sometimes varying by 1 from the mean values given for each column).

Here also, the agent follows an ϵ-greedy strategy, with $\epsilon = 0.01$, along with the SARSA update using $\alpha = 0.5$ (constant throughout the episode) & $\gamma = 1$. The corner cases were dealt with same as the previous cases. After averaging over 20 seed values (0-19), following are the plots for the same

Due to the stochasticity, this task is more exploratory in nature than the task without it. The states which were seldom observed in previous tasks will now be observed more frequently. Due to this fact, this task requires even more data than the previous task.



(a) Episode Lengths for Sarsa(0) agent with King's Moves & stochastic effects of winds



(b) Episodic Rewards for Sarsa(0) agent with King's Moves & stochastic effects of winds

Even after running the task for 2000 episodes, the average number of steps per episodes is $\sim$ 18, which is much larger than the case without stochasticity (= 7). Moreover, we also observer the variance of episode lengths & rewards during convergence is much larger, primarily attributed due to the stochasticity of the winds.

5 Sarsa Agent in Different Environments

Following is a comparative study of the Sarsa(0) Agent, when acting under different environments (mentioned in the above tasks). One addition is the case for stochastic wind effects, with only 4 moves (no King's moves), just for the sake of completeness. All other parameters have been kept similar to previous cases.

It is evident that for the non-stochastic case, the agent discovers a policy to allow it reach the goal faster with King's moves enabled, simply due to the higher degrees of freedom available.

Contrary to the expectation, for the stochastic winds case, the agent without King's moves enabled seems to reach the goal faster on an average.

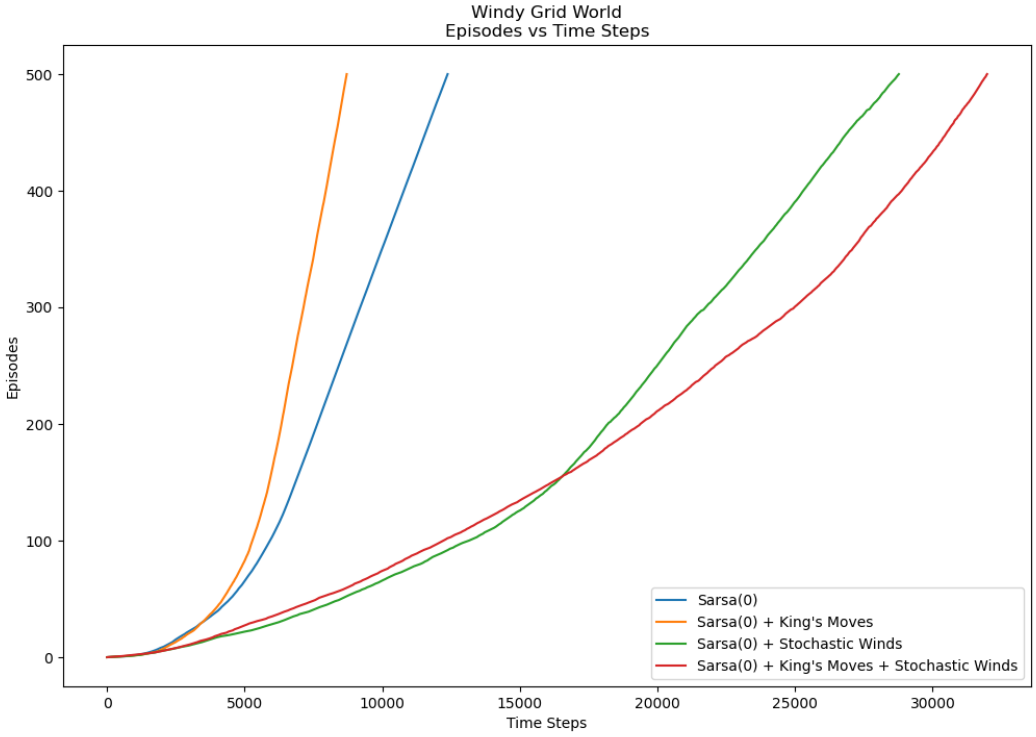
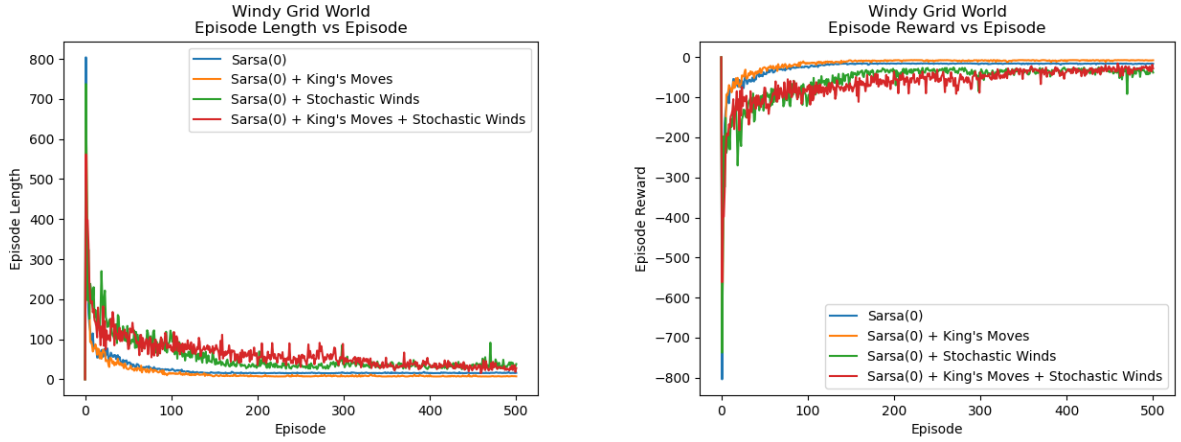


Figure 7: Comparison of Episodes vs Time Steps plot for Sarsa(0) Agent



(a) Comparison of Episode Lengths for Sarsa(0) agent

(b) Comparison of Episodic Rewards for Sarsa(0) agent

6 Task 5

The Q-Learning and Expected Sarsa Agents have been implemented using the `QLearningAgent` & `ExpectedSarsaAgent` classes respectively in `agents.py`.

Here also, the agents follow an epsilon-greedy strategy, with $\epsilon = 0.01$, along with the

respective learning updates using $\alpha = 0.5$ (constant throughout the episode) & $\gamma = 1$. After averaging over 20 seed values (0-19), following are the plots for the same:

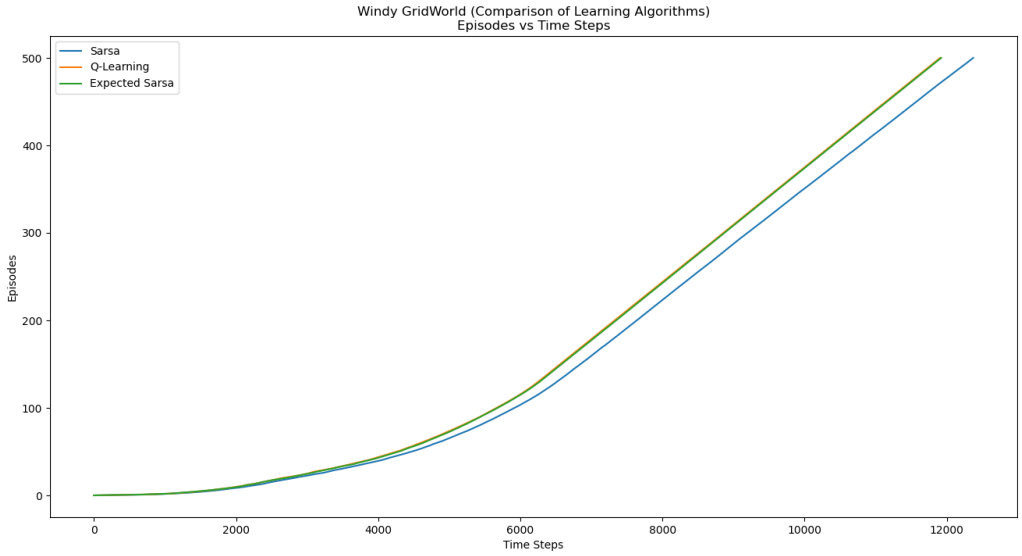


Figure 9: Comparison of Learning Algorithms on Baseline Windy GridWorld

We observe that both Q-Learning & Expected Sarsa manage to converge better to the minimum number of steps needed per episode (i.e. 15) compared to Sarsa(0).