# Assignment 3: Report

*CS747*

Siddharth Saha
170100025

November 13, 2020

## 1 Windy Gridworld

For ease of manipulation, the origin (0,0) grid square is defined at the top left and the world is represented as a 7x10 array. The x-coordinate increases rightwards, while the y-coordinate increases downwards. Thus the wind effect is subtracted, rather than added. The function inside `windy_gridworld.py` returns reward and next state based on the following input parameters:

- Current state
- Action
- Actions List (Valid inputs: Four moves or King's moves)
- Stochasticity (Default value: False)
- Wind Strength (Default value: [0, 0, 0, 1, 1, 1, 2, 2, 1, 0])
- Maximum x-coordinate (Default value: 9; 0-based indexing)
- Maximum y-coordinate (Default value: 6; 0-based indexing)
- Goal state (Default value: [3, 7])

We update x and y coordinates with the full effect of the action and the cross wind (at the initial x-coordinate). Finally, the bounds of the gridworld are imposed on the `[y_new, x_new]` state obtained.
This same script generalizes to account for stochasticity of wind by adding a random integer in the range [-1,1] if the wind strength at the initial x-coordinate was non-zero.

## 2 TD Control Algorithms

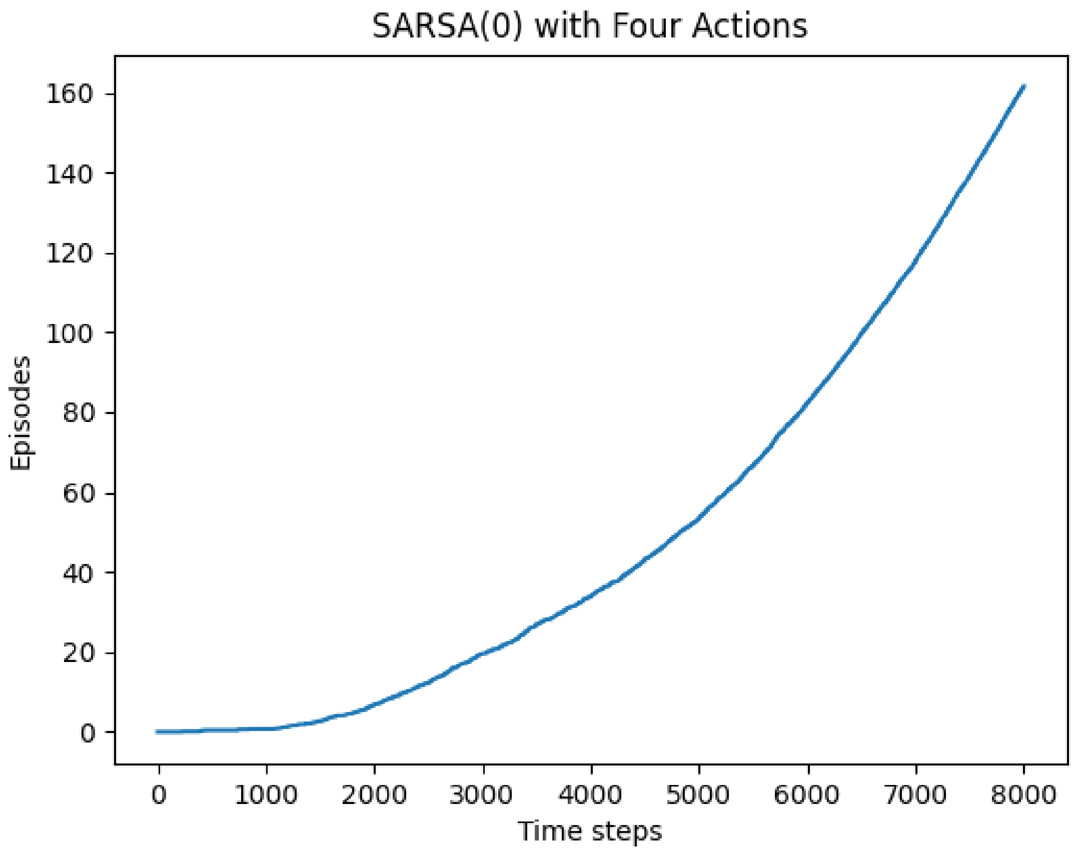
All 3 TD control algorithms are parameterized by the maximum number of time steps. They break out of the while loop as soon as the maximum time step has been reached and return the number of episodes completed at each intermediate time step. The policy is $\epsilon$-greedy in the action values. When choosing greedily in the presence of multiple maximum values, the `np.argmax()` function picks the lowest of maximum value indices. The effect of randomly selecting (`np.random.choice()`) from these maximum value indices was examined. It was observed to produce no discernible effect on the plot.

## 3 Agent's Performance

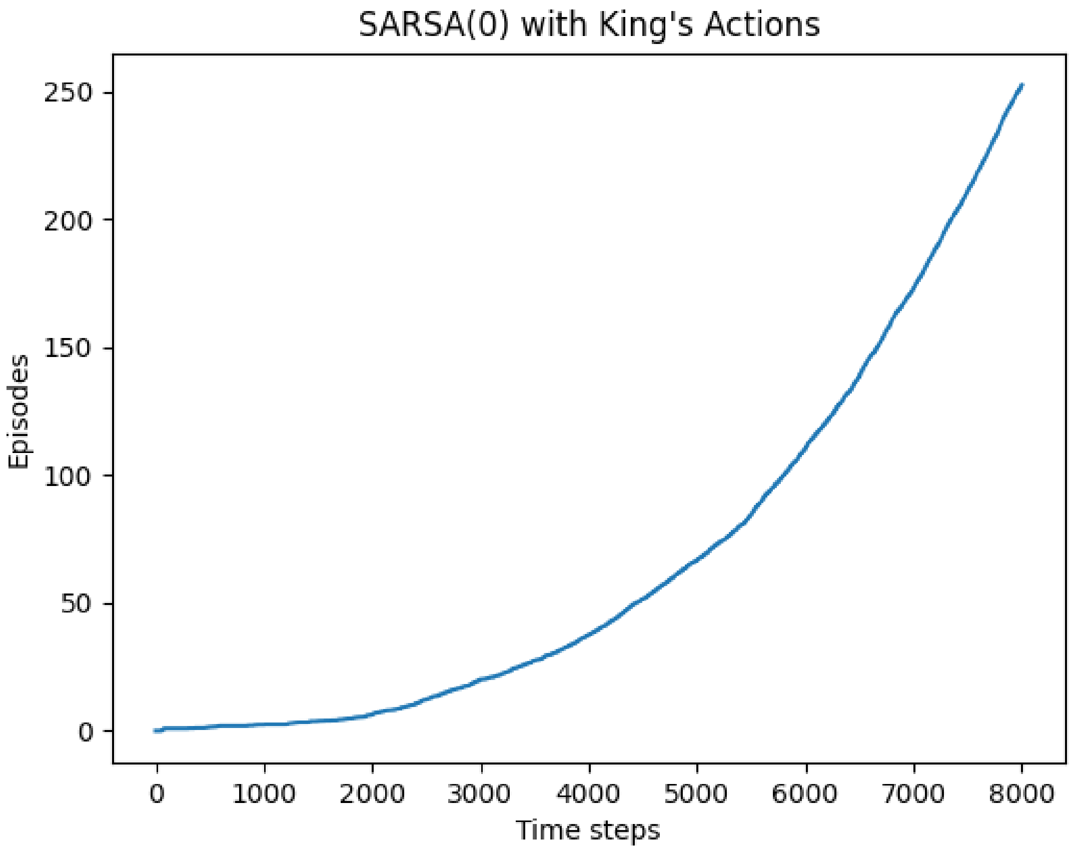
In each of the experiments, identical parameter values were taken as given in the example, that is:

- alpha = 0.5
- epsilon = 0.1
- gamma = 1 (undiscounted episodic task)

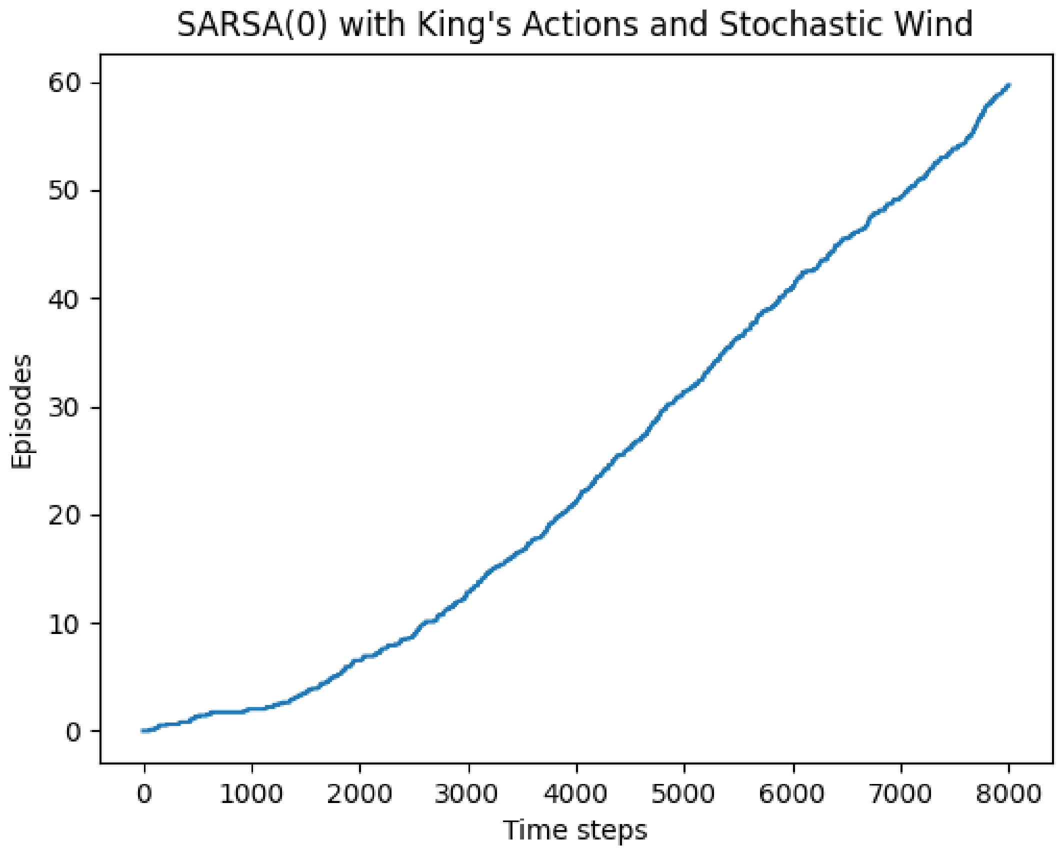
The plots have been averaged over `10` runs.



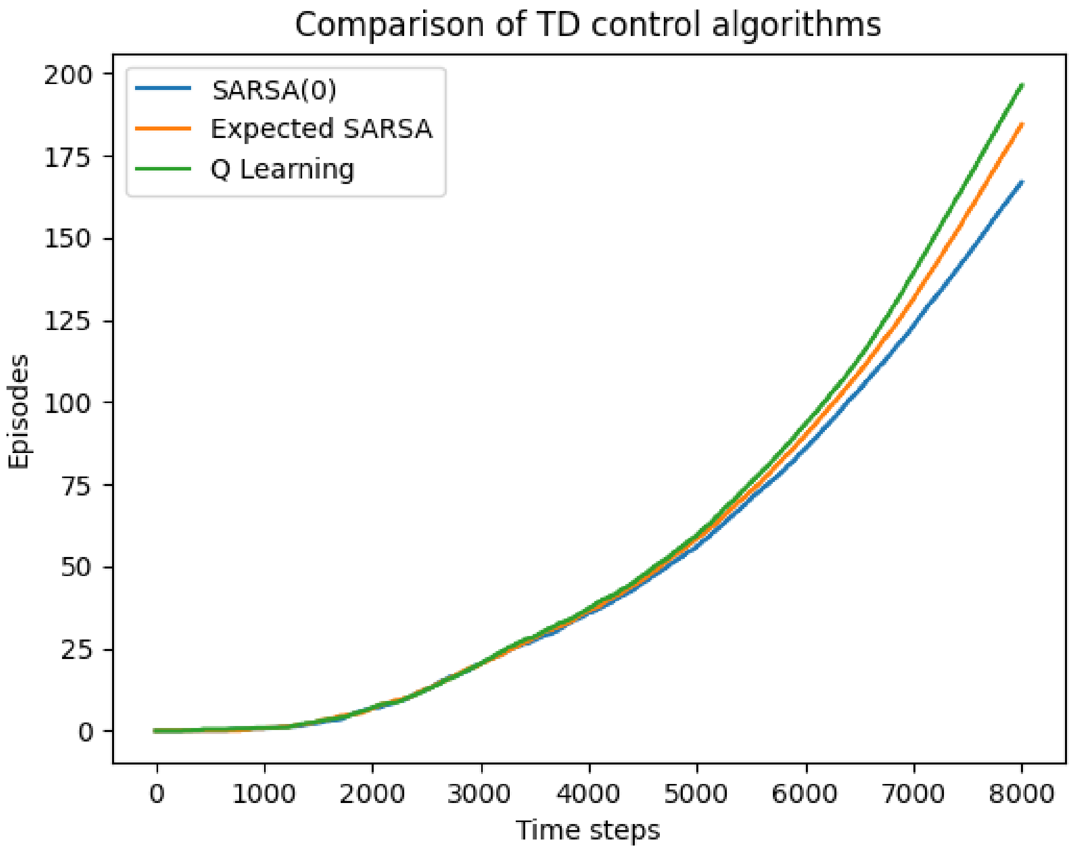
This result nearly matches the baseline plot presented in Sutton and Barto. The first few episodes required time steps of the order of 400. Thus we see near zero slope at the start.



Intuitively, we would expect more number of successful episodes in the same time with increase in available actions. The trade-off is that the search space is increased, but we observe from the plot that the benefit due to shorter optimal path length easily wins.

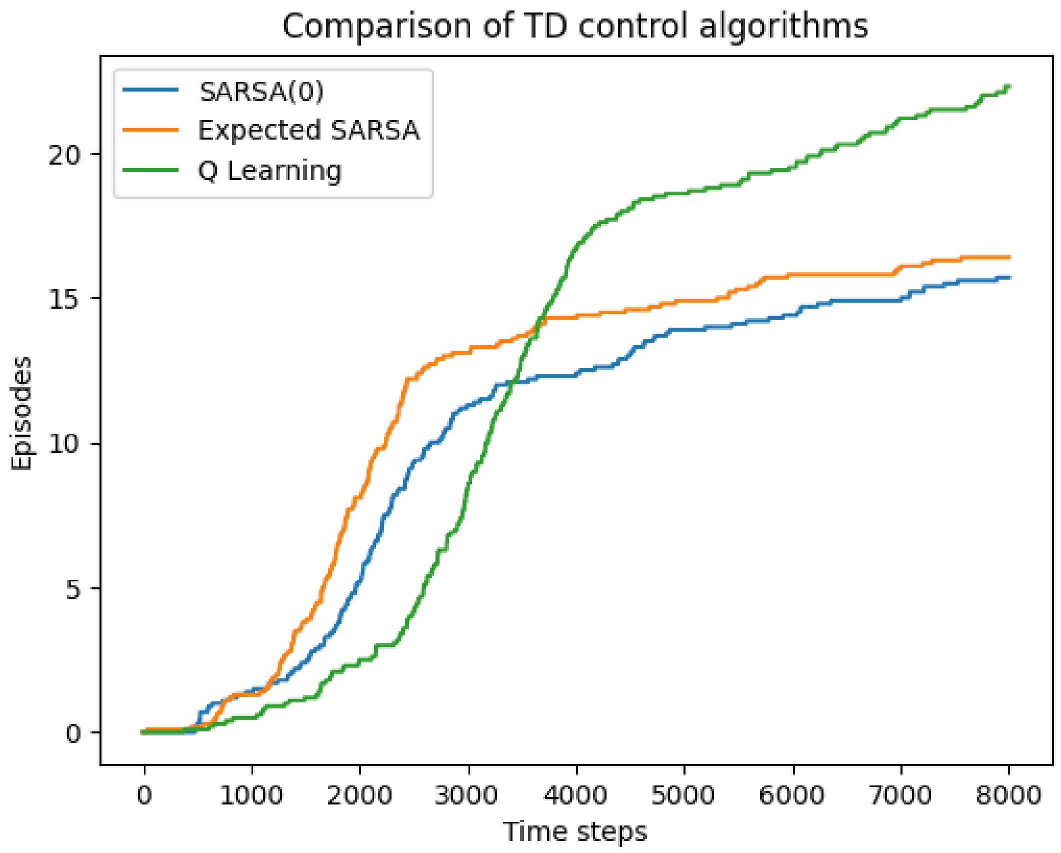


Here the first observation is that there cannot be any definite optimal path as the behavior of the cross wind itself is stochastic. The agent is still able to learn decisions that help us reach the goal position in the presence of stochasticity. The number of completed episodes in the same time are intuitively much lower.



Here we distinctly infer from the plot that Q-learning performed better than Expected SARSA, which in turn out-performed SARSA(0) (For the given values of epsilon, alpha and gamma).

Taking inspiration from the Cliff Walking problem (Example 6.6), we run another experiment where we impose penalty of -100 on trying to escape the grid (reward of -1 in all other cases).



The plot here along with the previous plot gives greater insights. Q-learning assumes that the agent is following the best possible policy without taking into account what the agent is actually doing. SARSA on the other hand looks ahead (on policy) to the next action that the agent would take and updates the action value accordingly. This resemblance with Cliff Walking problem is visible in the first half of the time steps. In the latter half, again the assumption of following the best possible path helps it eventually find optimal policy faster than SARSA (as in the previous plot).