# ASSIGNMENT 3

**SHUBHAM LOHIYA**
**18D100020**

8th November 2020

—

CS 747

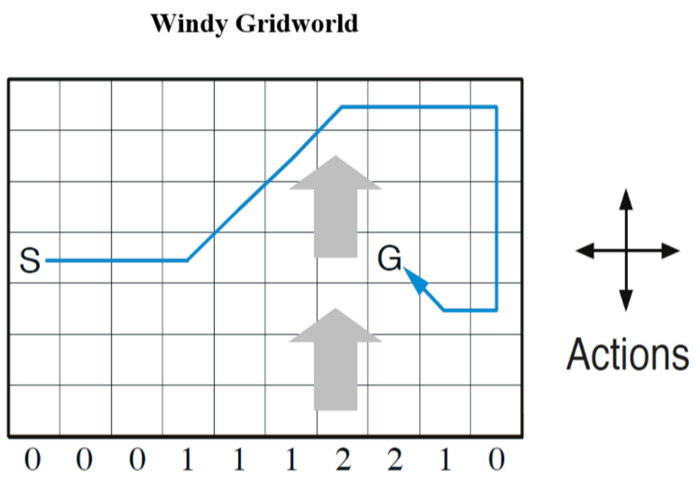
—

Shivaram Kalyanakrishnan

## INTRODUCTION

This document is a report for the third assignment of the CS 747 – Fundamentals of Intelligent and Learning Agents course offered at IIT Bombay.

This assignment is based on comparison of various off-policy and on-policy reinforcement learning algorithms like Q-Learning, Sarsa, Expected Sarsa. It also analyses the effect of having a stochastic environment (random wind) on the performance of the agent.



## TASK 1

### WINDY GRIDWORLD IMPLEMENTATION

The simulation of the agent in a windy gridworld is as described in Example 6.5 of the book [Reinforcement Learning: An Introduction by Andrew Barto and Richard S. Sutton].

Here is a short description of the codebase.

`gridworld.py`: This contains the `WindyGridworld` class which defines the world layout, the start and the end states, the wind and allowed actions for the agent. It also contains a `move` function which dictates the transition (next state) of the agent when it performs a particular action from a particular state.

`agent.py`: This contains the `Agent` class which defines the agent's behavior. An instance of this class has a `WindyGridworld` instance as an attribute. It also contains functions to run an episode in the gridworld (from start state to the end state) using one of the algrithms – Sarsa, Q-Learning, Expected Sarsa. An instance also has the `q_val` attribute which tracks the action value function over episodes. The class also has a play member function which runs the agent on the algorithm of choice for 200 episodes.

`main.py`: This file contains the driver code for the simulation. Usage is described in `README.md`. The script takes command line arguments which define the type of simulation to run and the plot. This script also contains a `run` function which runs the simulation of choice over 10 random seeds and averages the observations across them to give the final data for the plot.

Hyperparameters Chosen:

Learning rate (alpha): This was chosen as 0.5 as described in the textbook example. It gave satisfactory results.

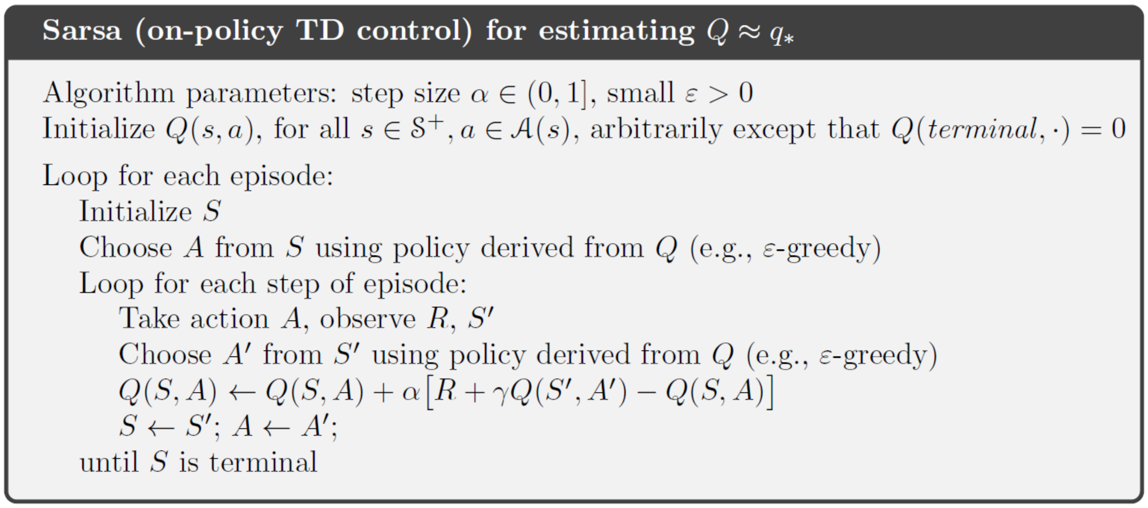
Exploration parameter (epsilon): This was also chosen as 0.1 as described in the textbook example. It gave satisfactory results.

Discount (gamma): This was set to 1 as this is an undiscounted episodic task.

Reward: This was chosen as -1 for every move to incentivize the agent to reach the goal state faster to accumulate minimum negative reward.

Here is a short overview of the algorithms that the agent can choose from. Implementation is as described in the textbook:

Sarsa:

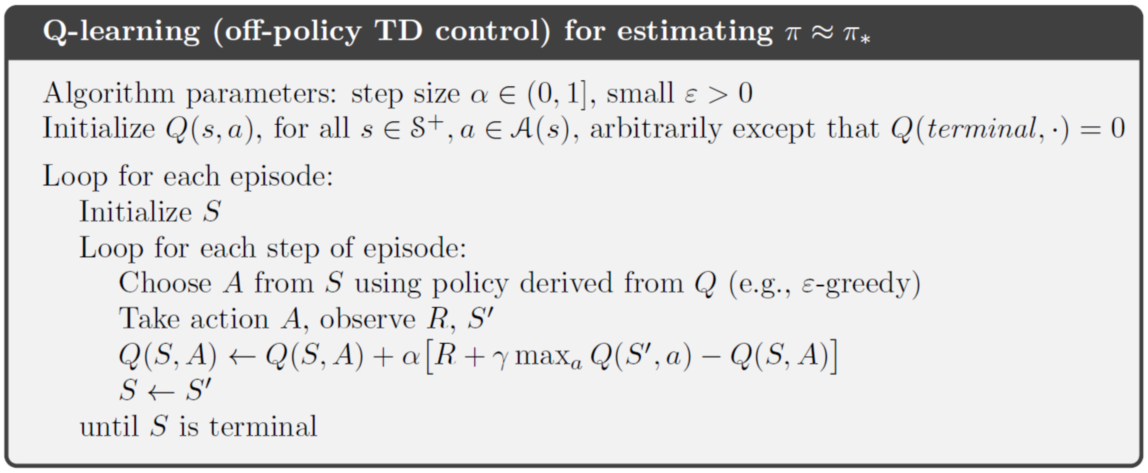


**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Q-Learning:



**Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

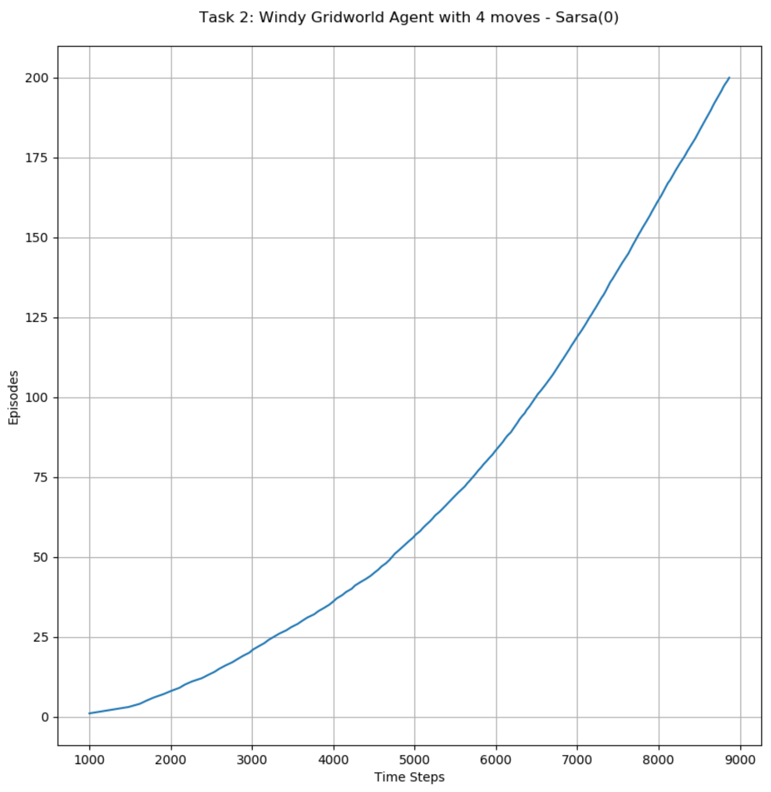
Expected Sarsa:

Implementation is similar to Q-Learning but instead of bootstrapping with respect to the action that has the maximum action value for the next state, it bootstraps with respect to the expectation of the next action value under the epsilon greedy strategy. This strategy has a weight of $1 - \left(1 - \frac{1}{n_a}\right)\varepsilon$ for the greedy action (max action value) and a weight of $\frac{\varepsilon}{n_a}$ for all the non-greedy actions.
We can clearly observe that if epsilon is set to zero, expected-Sarsa is exactly q-learning.

## TASK 2

### SARSA(0) AGENT WITH 4 MOVES AND DETERMINISTIC WIND (BASELINE)

Here is the plot of episodes vs time-steps obtained by running the agent under these conditions for 200 episodes averaged over 10 random seeds:



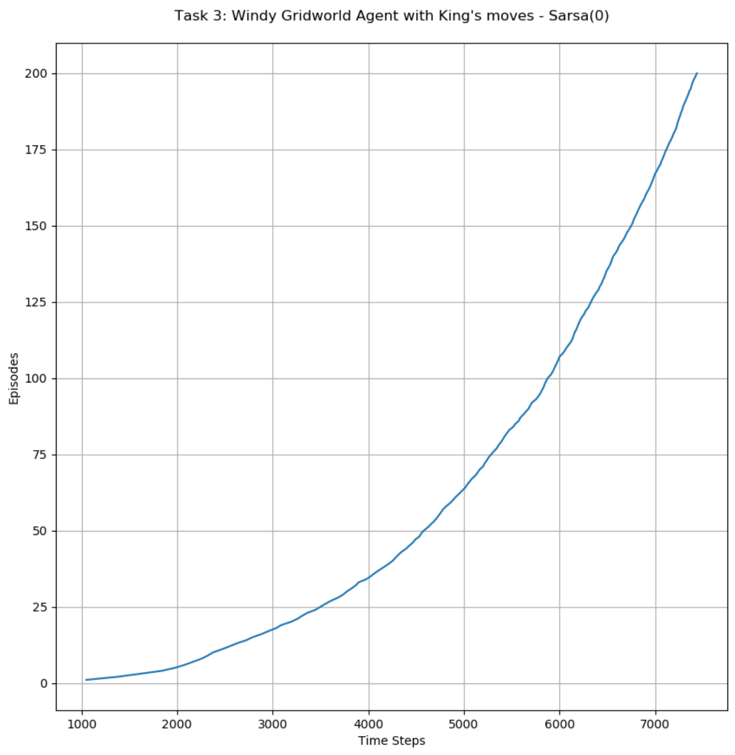
**Observations and Inferences:**

- We clearly see that the obtained plot is similar to one given in the text book example
- The initial increasing slope is indicative of the agent's learning and the decreasing average episode length. The slope later on become almost constant thus making the relationship linear. This is expected as after a significant number of episodes, the agent has learnt all it can and now, it's average episode length can't go down any further. Expected time to go from the start to the end state has become stable
- This expected average length of an episode in the long run corresponds to the best possible path that the agent can take under given constraints

## TASK 3

### SARSA(0) UNDER PREVIOUS CONDITIONS WITH KING'S MOVES

Here is the plot of episodes vs time-steps obtained by running the Sarsa(0) agent for 200 episodes averaged over 10 random seeds:



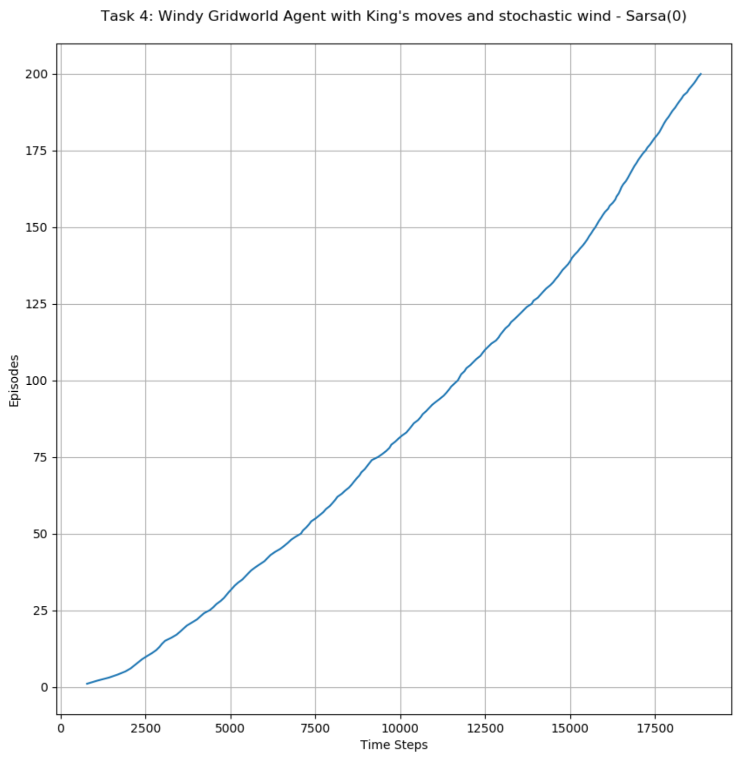
**Observations and Inferences:**

- Allowing 4 extra moves drastically reduces the number of time steps for the same number of episodes as compared with the baseline plot (task 2)
- The average length of the episode is the long run is considerably lower (7 vs 15 on inspection) than in the baseline mode as evidenced by the higher long-term slope of the plot as compared to that in task 2
- This is consistent with our expectation as due to the increase in the number of allowed moves, the agent has more options and hence can get to the goal state faster. Since the previous 4 moves are allowed anyway, we also expect the agent to perform no worse. Hence, it can only perform better or the same. The result of this simulation confirms that king's moves allow the agent to perform way better than the 4 moves setting.

## TASK 4

### SARSA(0) WITH KING'S MOVES AND STOCHASTIC WIND

Here is the plot of episodes vs time-steps obtained by running the Sarsa(0) agent under stochastic wind for 200 episodes averaged over 10 random seeds:



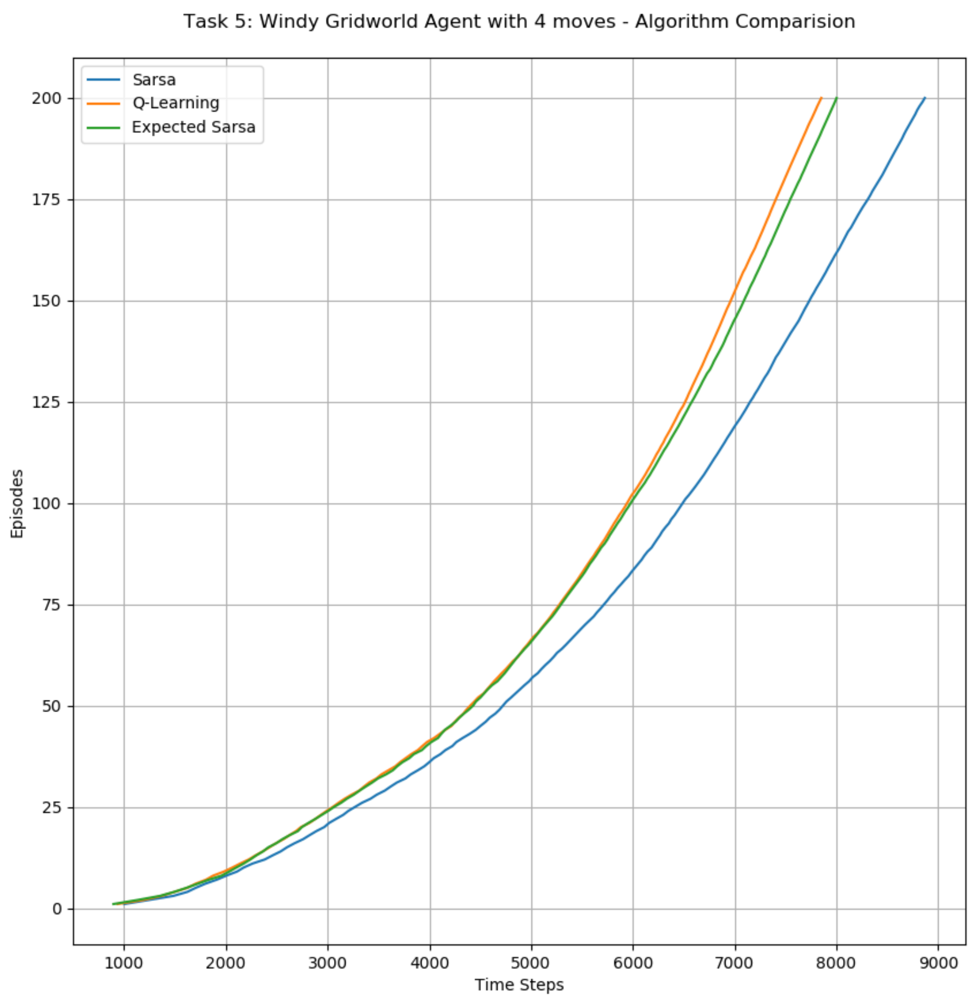
**Observations and Inferences:**

- We observe that the agent learns slightly only in the first few episodes and then there is a mostly linear relationship between episodes vs time-steps
- This is because the agent quickly learning what to do in the columns where there is no wind but for columns which have stochastic wind, it is unable to learn an optimal policy due to the stochastic nature of the transition. Hence even after a lot of episodes, the agent is unable to bootstrap from past experience to choose actions that allow it to get to the goal state faster. Thus, the expected episode length is constant and larger than in the first two cases (as evidenced by slope comparison between this and the last two tasks). The stochastic wind doesn't allow any actions to be observed as better closer to the goal due to variation in transitions.

## TASK 5

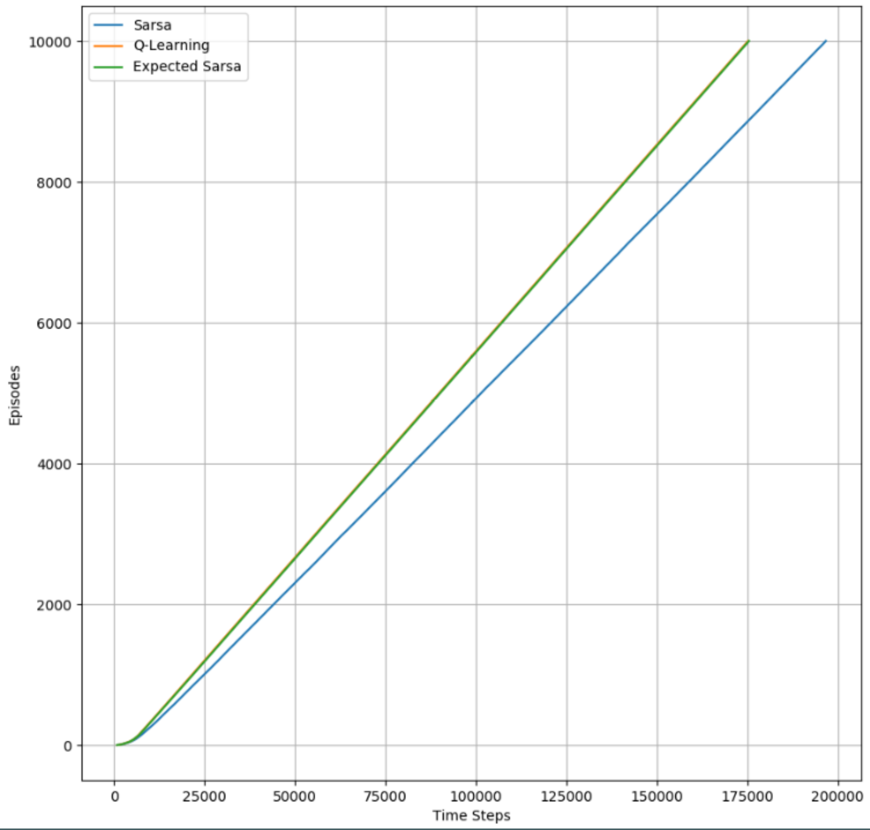
### COMPARISION OF ALL 3 ALGORITHMS FOR THE BASELINE CASE

Here is the plot of episodes vs time-steps obtained by running the agent under all three algorithms for the baseline case and for 200 episodes averaged over 10 random seeds:



**Observations and Inferences:**

- Expected Sarsa is consistently better than Sarsa. This is expected (no pun intended xD) because Expected Sarsa makes the update on every time step that Sarsa "expects" to (hence the name expected Sarsa), i.e. the action that determines the update of Sarsa comes from an epsilon-greedy action selection policy. The expectation of this bootstrapping is exactly what expected Sarsa explicitly implements and hence it converges faster.
- Q-Learning  and Expected Sarsa have very similar performance in the first few episodes whereas eventually, Q-learning gives better performance and converges faster to the optimal policy. This is because Q-learning is an off-policy method and it directly learns the optimal policy whereas Sarsa and Expected Sarsa are on-policy methods and they try to learn the action value function under current policy while also updating this policy on the basis of the learned function.
- Let's look at very long run performances of algorithms in the plot below for 10000 episodes. Q-Learning and Expected Sarsa are almost the same. The difference in performance between Sarsa and Expected Sarsa is because we are not annealing the epsilon and the learning rate over time. While Expected Sarsa update step guarantees to reduce the expected TD error, Sarsa could only achieve that in expectation (taking many updates with sufficiently small learning rate). Through this perspective, there is little doubt that Expected Sarsa should be better.


Sarsa
Q-Learning
Expected Sarsa
Episodes
Time Steps