The Windy Gridworld Problem

CS747: Foundations of Intelligent and Learning Agents
Programming Assignment 03

Report submitted by: Aaron John Sabu

1 Introduction

The assignment deals with the implementation of different algorithms (Q-Learning, Sarsa, Extended Sarsa) for the reinforcement learning problem by computing the optimal path to reach an end state from a start state in the midst of a vertical wind that varies in magnitude with the horizontal location of the agent. This problem has been adopted from the Windy Gridworld task given as Example 6.5 in 'Reinforcement Learning' ed. 2 (2018) by Sutton and Barto. The first part deals with the direct implementation of the Sarsa algorithm with 4 moves and a variant called 'King's Moves' using 8 moves, along with a scenario in King's Moves where the wind is stochastic. The second part involves the direct implementation of the Q-Learning and Extended Sarsa algorithms.

2 Tasks

The gridworld is expressed as an episodic MDP where the action value functions and wind (or mean wind) values are stored in Q and *Winds* respectively for each state. All the algorithms have been implemented in *gridworld.py* running on Python 3.8. The problem statement consists of the following tasks:

- Sarsa algorithm with 4 moves ('Normal Moves') and non-stochastic wind
- Sarsa algorithm with 8 moves ('King's Moves') and non-stochastic wind
- Sarsa algorithm with 8 moves ('King's Moves') and stochastic wind
- Q-learning algorithm with 4 moves ('Normal Moves') and non-stochastic wind
- Extended Sarsa algorithm with 4 moves ('Normal Moves') and non-stochastic wind

All tasks are similarly implemented with the update function given extra parameters, such as the next state's action a^{t+1} for Sarsa, and the next state's policy for Extended Sarsa. Q-learning, being a model-free algorithm, does not require any extra parameter from the next state and can be calculated using the learning rate, discount, present state, present action, reward, and next state.
The algorithms have been implemented using the *algorithm* function. Here, the stochastic wind has been modeled by adding a random choice from $[-1, 0, 1]$ to the current wind value in each state. The pseudo-codes provided in the textbook have been followed along with the addition of wind.
The program runs a *main* function which runs the algorithms that are assigned to it via a list named *Options* and the corresponding graphs are plotted. Moreover, for the non-stochastic wind cases, the optimal path is printed along All tasks involve starting at $(3, 0)$ and ending at $(3, 7)$ where indices are numbered from 0 and the order is (p_y, p_x) when counted with the topmost leftmost point as the origin. The value of ϵ is taken as 0.1, the discount $\gamma = 1$, and the learning rate $\alpha = 0.4$.

3 Results and Observations

As is visible in the following plot that averages the values computed for 50 random seeds, the obtained results are similar to those given in the textbook.

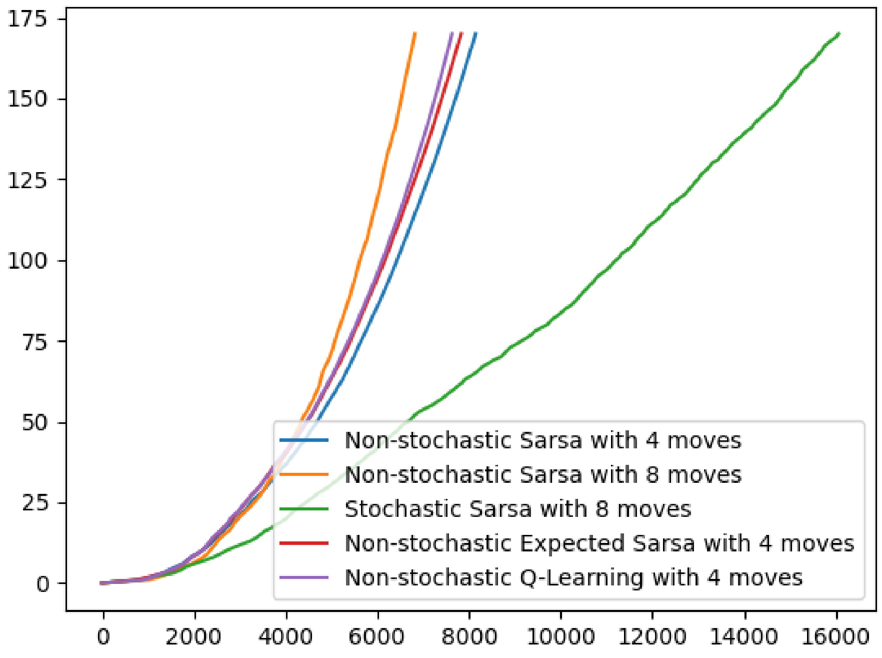
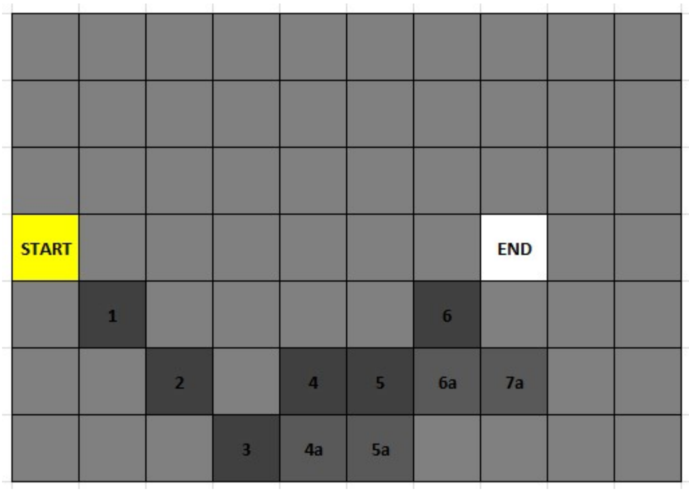
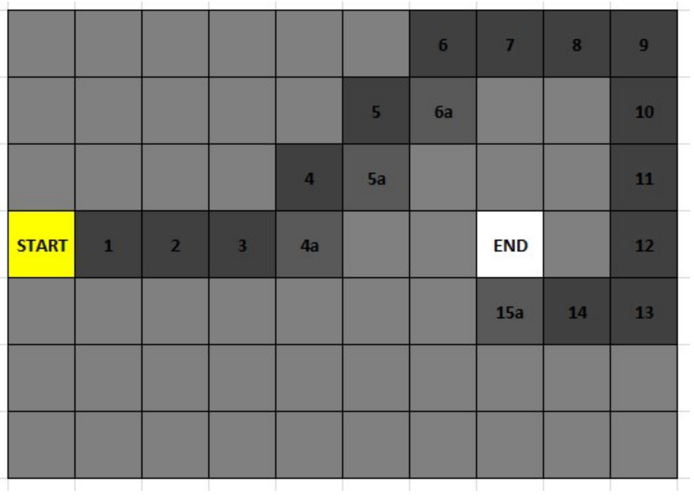


Figure 1: Comparison of the five schemes

As expected, the Sarsa algorithm with 8 moves and non-stochastic wind shows fastest convergence since there are less number of states involved in each episode although the number of actions for each state is doubled. Q-learning and expected Sarsa show similar behavior but are both faster than Sarsa since the latter involves a non-comprehensive prediction of the next action while expected Sarsa considers the entire policy and Q-learning is based on a known policy
Given in the first part of the next figure is the path obtained by the deterministic-wind King's Moves variant, while all other algorithms (apart from the stochastic wind variant, where a definite path cannot be determined) give the path as shown in the second part. The points marked with an *a* represent the points where the agent would have moved in the absence of wind given the previous state and action.



(a) Path 01: King's Moves



(b) Path 02: Normal Moves

Figure 2: Final Paths