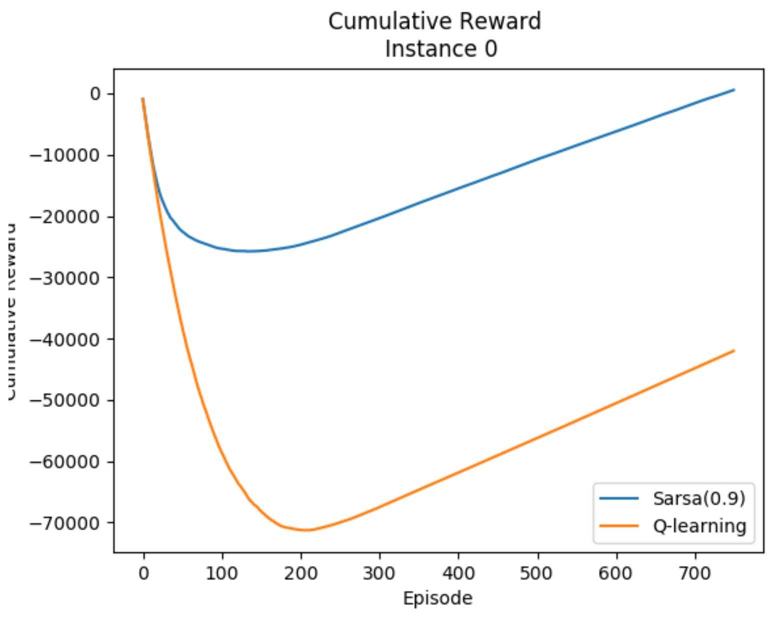
CS 747
Assignment 3 - Report

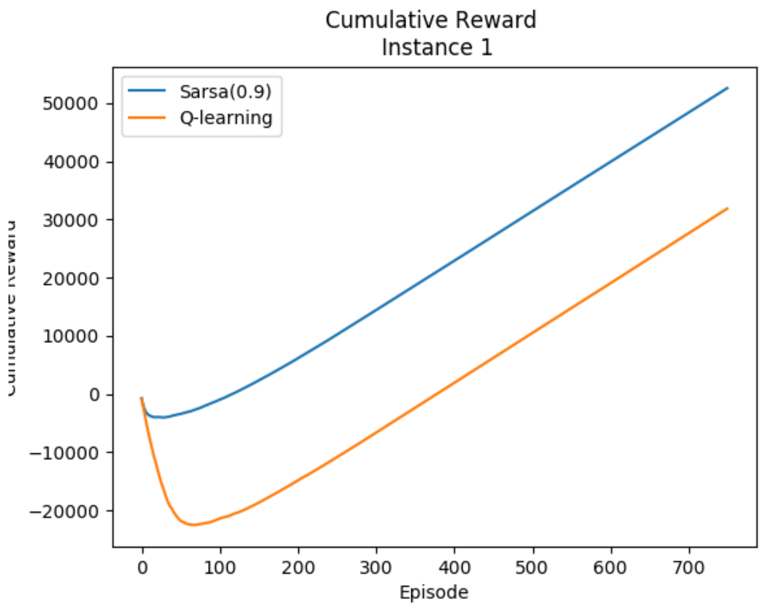
Goutham Ramakrishnan, 140020039

Grid size: 32x32
Algorithms implemented: Sarsa(λ) and Q-learning
Values of λ used for simulation: 0, 0.05, 0.1, 0.2, 0.45, 0.6, 0.7, 0.8, 0.9, 1.0
Value of Gamma was fixed to be 1 for all simulations.
No. of episodes for each simulation: 750

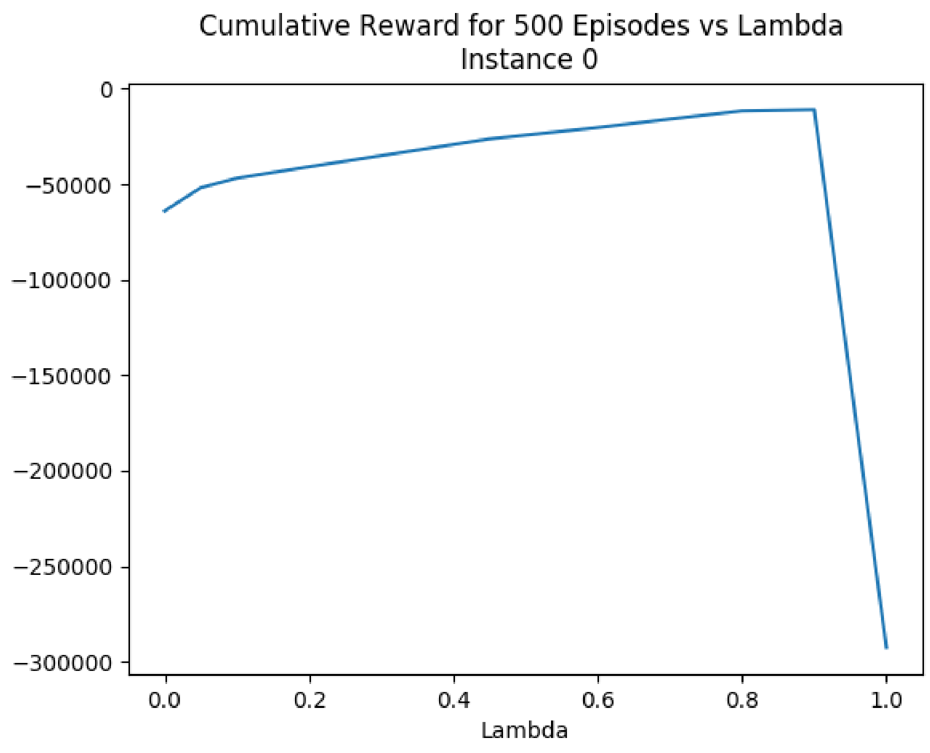
Summary of simulation results:
For both instances of the grid, the best value of λ was found to be 0.9(among the values of λ tried)
All graphs are averaged over 50 simulation runs.

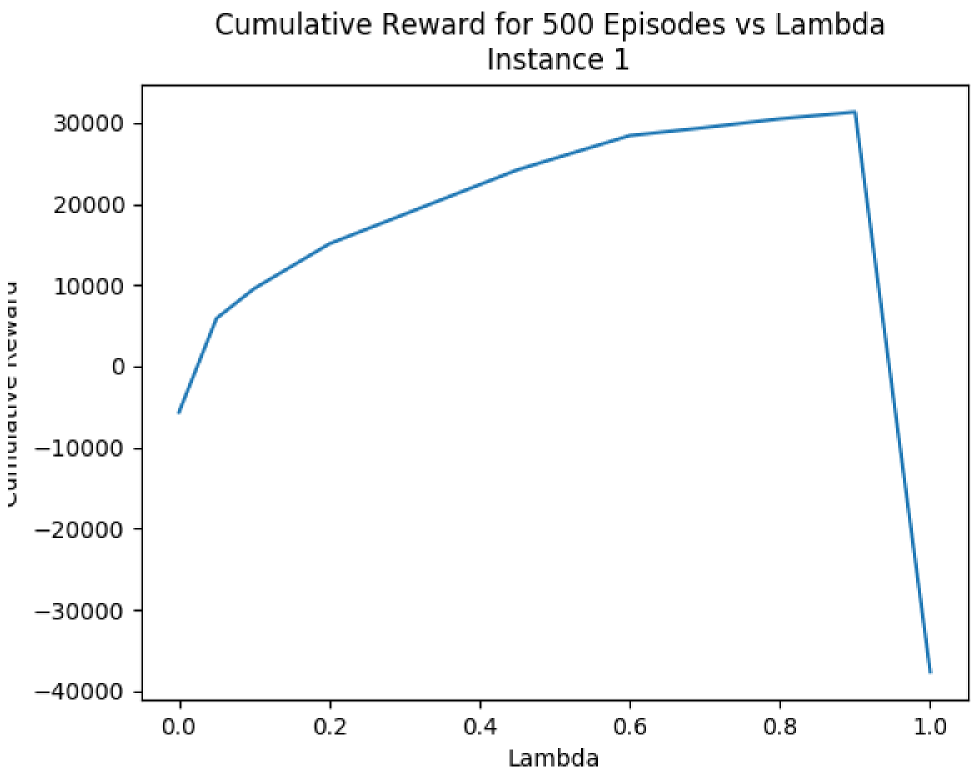
1. Graphs of Cumulative Reward vs Episodes



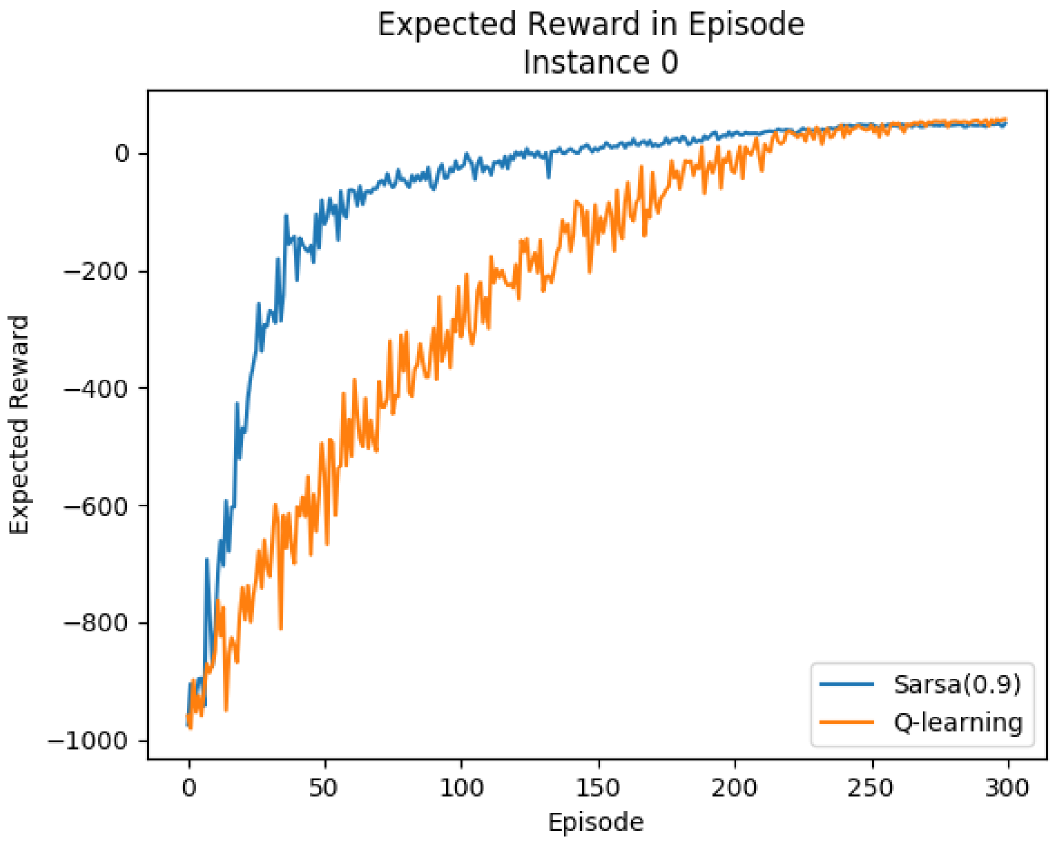


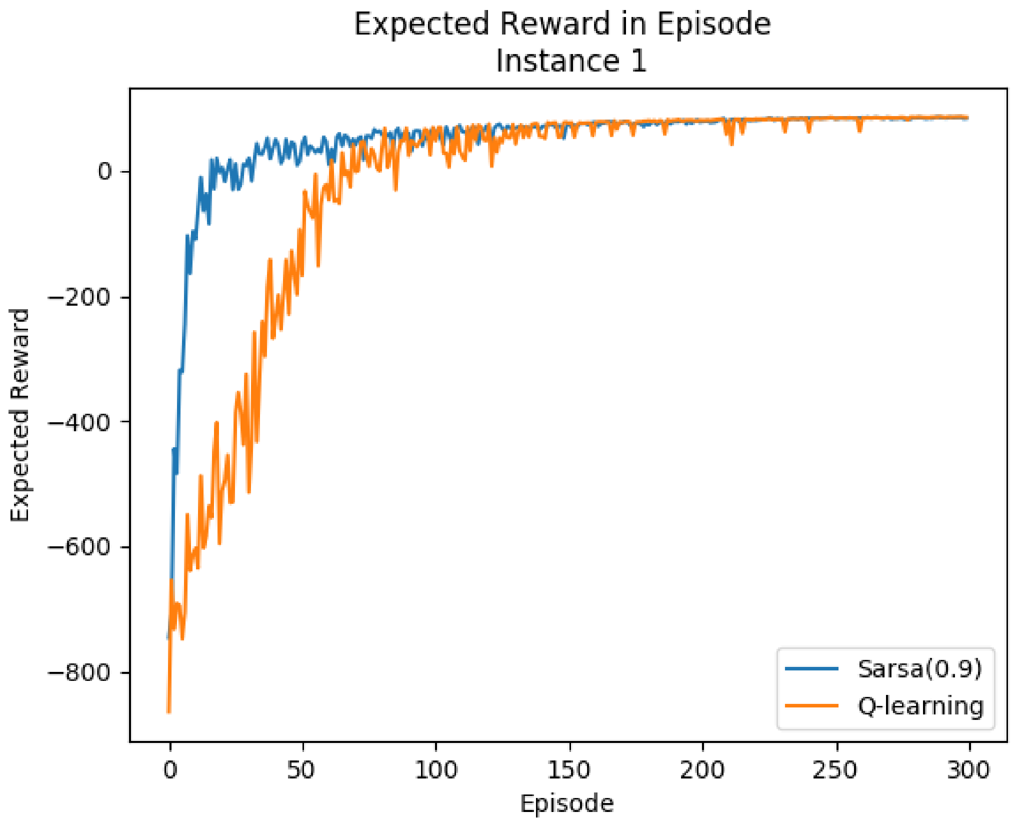
2. Graphs comparing cumulative reward over 500 episodes for Sarsa(λ) vs λ



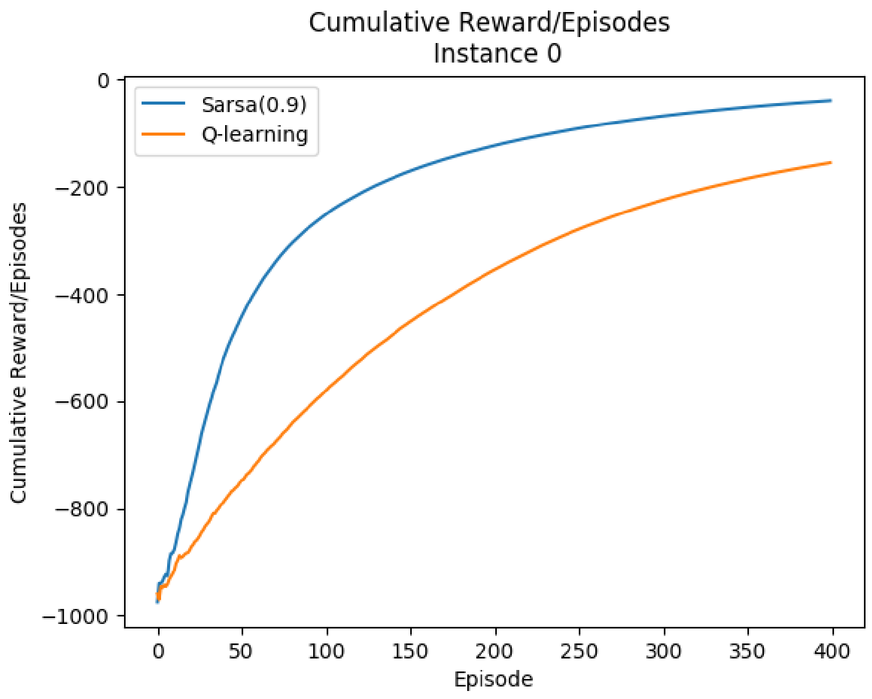


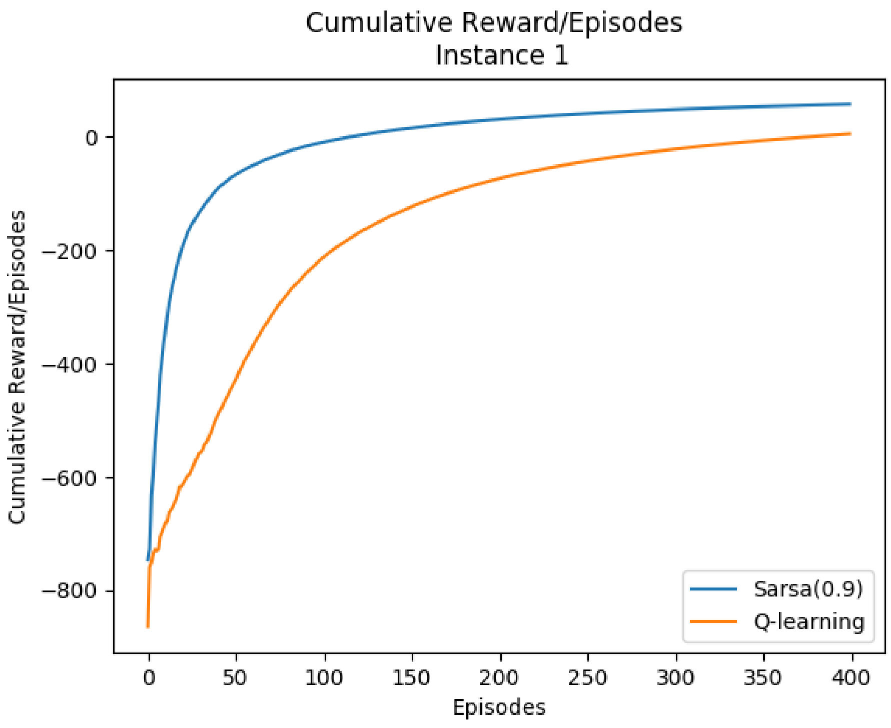
3. Graphs of Expected Reward in a particular episode





4. Graphs of (Expected Cumulative Reward/No of epidoes complete) vs Episodes





Comments, Observations and Interpretations:

- Strategy used: Both learning rate and exploration rate were annealed. Some optimization was done to decide which combination is working well. Replacing traces was used for Sarsa as it performed better. In fact, accumulating traces was not converging for the larger values of λ.
- For Q-learning, the learning rate was initialized to 0.8 and annealing was started after 100 episodes. It was annealed by 0.002 after each episode, and was lower bounded by 0.3. The exploration rate was initialized to 0.5 and annealing was started after 100 episodes. It was annealed by 0.002 after each episode, and was lower bounded by 0.1.
- For Sarsa-learning, the learning rate was initialized to 1.0 and annealing was started after 50 episodes. It was annealed by 0.002 after each episode, and was lower bounded by 0.5. The exploration rate was initialized to 0.25 and annealing was started after 50 episodes. It was annealed by 0.002 after each episode, and was lower bounded by 0.05.

- Instance 1 seems to be a significantly easier grid to solve than Instance 0.
- Sarsa(0.9) outperforms Q-learning in terms of convergence rate, as it can be seen from the first and third set of graphs. It is to be noted though, that both algorithms converge to the same value. Even though Sarsa(0.9) learns much faster, it also takes much more computation to implement. Depending upon the need for convergence speed vs. computational allowance, we can choose one of the two algorithms.
- From the second set of graphs, we can see that performance improves smoothly as λ is increased. There is an aberration at λ=1, which is explained below.
- The performance of Sarsa(λ) is quite poor at λ=0 and improves as λ is increased. This is to be expected as performance increases when we decrease our degree of bootstrapping and move towards Monte-Carlo. Nevertheless, performance saturates with increasing lambda. For both instances λ=0.8 and λ=0.9 showed similar performance.
- What is strange though, is that λ=1(which corresponds to MC) showed extremely poor results for both instances. This is because the value of gamma used for simulation was 1. This would mean that in the update rule for e(s)(a) {e(s)(a) = gamma*λ*e(s)(a)}, there is no decay in the trace as both λ and gamma are 1.
- When Gamma was set to a smaller value, the Sarsa(1) algorithm began to show better results. For example, for Gamma=0.99, it obtained a cumulative reward of around -70000 for instance 1 over 750 episodes. For Gamma=1, the correspoding cumulative reward was -290000. It is safe to say that if Gamma is decreased further, the performance will improve.

Attached:
Shell script to start client: startclient.sh
Primary python script: agent.py
client.py is used to start coordinate with agent.py
Folder results contains all simulation results, for both instances, for both algorithms.
Python script used to compile simulation results: compileResults.py

References:

→ Python Documentation
→ http://incompleteideas.net/sutton/book/ebook/node77.html
→ Sutton and Barto
→ StackOverFlow