Deterministic Policies

Reward Scheme

Move out of bounds = -2

Originally this was set to -10 though I realized that any value lower than the in bound value would cause it to avoid going out of bounds, and higher rewards would cause the sarsa values to diverge less when chosen.

Move to goal state = +100

In bound move = -1

Q-Learning

Variables

Values: Discount = 0.9, epsilon = 0.3, alpha = 0.5

Results: Optimal policy ~ 9000 episodes, required time ~ 4.2s

Overall, I found after some testing that higher epsilons increase exploration and thus the time of each episode, but also decreases the required episodes. You can increase epsilon and decrease the episode number or vise versa.

For example,

Values of: Discount = 0.99, epsilon = 0.5, alpha = 0.5

Give results: Optimal policy ~ 5000 episodes, required time ~ 3.8s

I found these values to be consistent enough, there is a chance of getting a slightly off optimal policy (spoken about below), but 2 runs will mostly guarantee optimal policy.

Optimal policy

[['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['D']]
[['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['D']]
[['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['D']]
[['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['G'], ['R'], ['D']]
[['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['D'], ['L'], ['L']]
[['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['D'], ['L'], ['L', 'U']]
[['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['R'], ['U'], ['U'], ['L']]

Notes

When converging, the state (9,6) shown here as ['L', 'U'] is the final policy to converge. When run at 8000 episodes (about 4.0s on my computer) there is still a small chance of the final policy returning only ['U'] in this state. When run at lower episodes this may also return only ['L'] (around 5000 episodes 2.6s) and past 8000 episodes it becomes very rare not to get ['L', 'U']. With a small amount of luck, the optimal policy will can be found around 7000 episodes. The reason to opt for discount of 0.9 rather than 0.99 is also because this makes the chance of missing the double action higher.

Overall, you can run at 7000 episodes if you don't care about the double action, and 9000 if you want a more consistent convergence. Overall maximum runtime ~4.2s, (this helps to measure the number of steps per episode, longer time for same number of episodes points to longer episodes).

Also note, the times recorded are all recorded with no changes in any methods, and all run on the same machine. The purpose of these times is just to compare with other times. Not to note the speed of the algorithms as this can vary by a large percentage based on the methods and machine.

Results
~ 9000 for consistent convergence
< 4.67 x 10^{-4} avg seconds per episode.
optimal policy takes 15 steps to reach goal

Sarsa

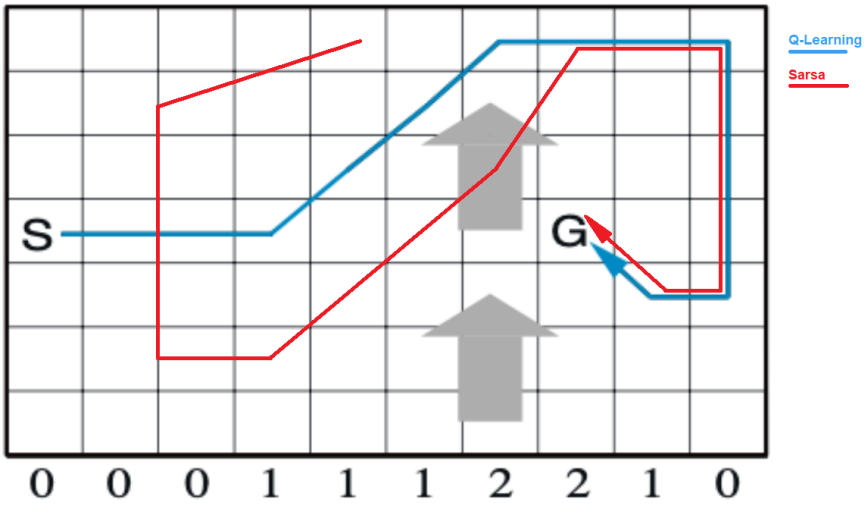
Optimal policy

After a few days of testing, the sarsa algorithm refuses to converge to a single policy. No values of discount, epsilon, and alpha will consistently give the same policy in under 10 000 000 episodes. Using a decaying epsilon value based on 1/t allows values to converge to a single policy but this policy is not consistent over multiple runs at any episode count under 10 000 000. Decaying epsilon based on 1/t also runs into the problem of premature convergence to a looping policy causing an infinite episode. A simpler decaying epsilon based on 1/episode stops this from happening though it causes a slower convergence based on episode count.
For the purposes of comparison, the decaying $\varepsilon = 1/episode$ will be used with discount of 0.9 and alpha of 0.5.

Efficiency Comparison
The algorithm was run 50 times and the time taken for training and the number of steps to the goal for the final policy were recorded per run.
Results:
Average time required = 5.3 seconds
Average steps to goal = 17.9 steps
We can see that the overall time required is quite a bit higher than the q-learning times with ~1.3 times longer average runtime.
The average steps to the goal for a trial of 10 Q-Learning runs was 15.0, compared to the sarsa average of 17.9, ~1.2 times worse. We can see very clearly that Q-Learning is the optimal algorithm for a deterministic state-space and offline training.
Finally, and most obviously, sarsa convergence is far more complicated and requires exponentially more episodes.

Policies
This is an example policy for sarsa
[['D'], ['D'], ['D'], ['L'], ['L'], ['L'], ['R'], ['R'], ['R'], ['D']]
[['R'], ['R'], ['D'], ['L'], ['L'], ['R'], ['R'], ['U'], ['R'], ['D']]
[['R'], ['D'], ['D'], ['U'], ['R'], ['R'], ['R'], ['R'], ['R'], ['D']]
[['R'], ['R'], ['D'], ['R'], ['R'], ['R'], ['D'], ['G'], ['L'], ['D']]
[['D'], ['R'], ['D'], ['L'], ['R'], ['R'], ['U'], ['U'], ['L'], ['L']]
[['D'], ['R'], ['R'], ['R'], ['R'], ['R'], ['D'], ['D'], ['R'], ['U']]
[['R'], ['R'], ['R'], ['R'], ['L'], ['D'], ['U'], ['U'], ['U'], ['L']]
No overall optimal policy could be determined. For the purpose of comparison, similar patterns were very clear over the runs as shown on the next page.

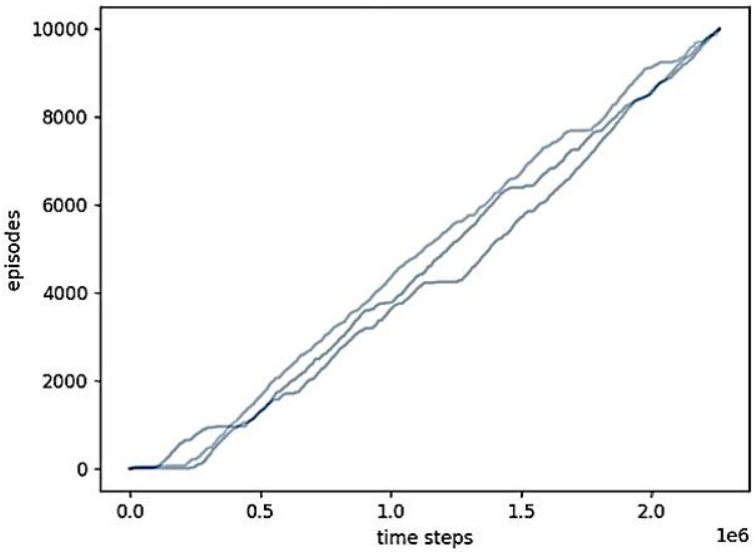
Policy Comparisons



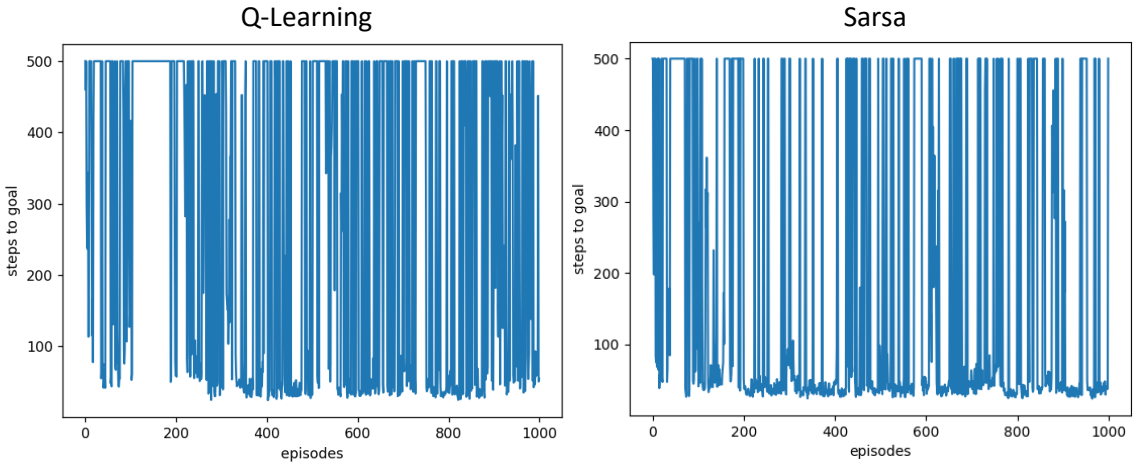
We can see here that while Q-learning will go straight towards the goal as fast as possible, sarsa tends to avoid being at the top of the grid in the wind. We can see sarsa's risk aversion as it moves to the bottom of the grid before going through the windy area. This is because it is trying to stay away from the grid borders as these may incur larger negative rewards (costs) as the agent moves out of the boundaries. Other than this we can see that sarsa follows the same path to the goal once reaching the top right of the grid.

Stochastic Grid

Now obviously we are not going to run the algorithm for enough episodes to come close to convergence for a stochastic grid. Instead, to compare the algorithms we can see the progression of policy evaluation as episodes go on. As wee can see here over the course of three runs of Q-Learning for 10 000 episodes:



The slopes of the lines are nearly linear after the first small section of episodes (~0.25 x 10^6 time steps). We can also see that there are a few very long episodes through the course of the total runs. It seems that the algorithm will very often run into looping episodes. To compare Q-Learning to Sarsa, we can look at the average number of steps required to reach the goal over 10 trials, per episode.



(Each trial will run 500 steps maximum before breaking therefore y values of 500 can be considered bad policies)

Results

Q-Learning, time required: 75.4 seconds, avg steps to goal: 279.0325000000001

Sarsa, time required: 331.9 seconds, avg steps to goal: 173.3528

We can see here that both algorithms very quickly reach their maximum potential within the first 300 episodes, after this both Q-Learning and Sarsa will shift back and forth between reasonable and bad policies. Q-Learning generally has more bad policies (500+ steps) when compared to Sarsa. However, Sarsa will very commonly run into episodes that take very long to complete. In testing I tended to lower the epsilon value for Sarsa, and found that this decreased the average episode length, but did not entirely stop the occurrences of the long episodes.

Overall, Sarsa seems to be the better algorithm at maintaining a decent policy over the course of multiple episodes. Though over many episodes Sarsa takes far longer to run.

In practice, finding a good policy would likely be easier if the algorithm is set to break after a policy is found that lies within an acceptable range. The best that the stochastic algorithm seems to find is around 30 so we can set the range to be any average number of steps over the course of 10 runs that is less than 30. We can see here the average number of episodes required to reach this range.

Q-Learning, avg number of episodes required: 192.73

Sarsa, avg number of episodes required: 116.03

As expected from the previous graphs Sarsa seems to consistently beat Q-Learning in finding a good enough solution first.