CS747 - Assignment 1

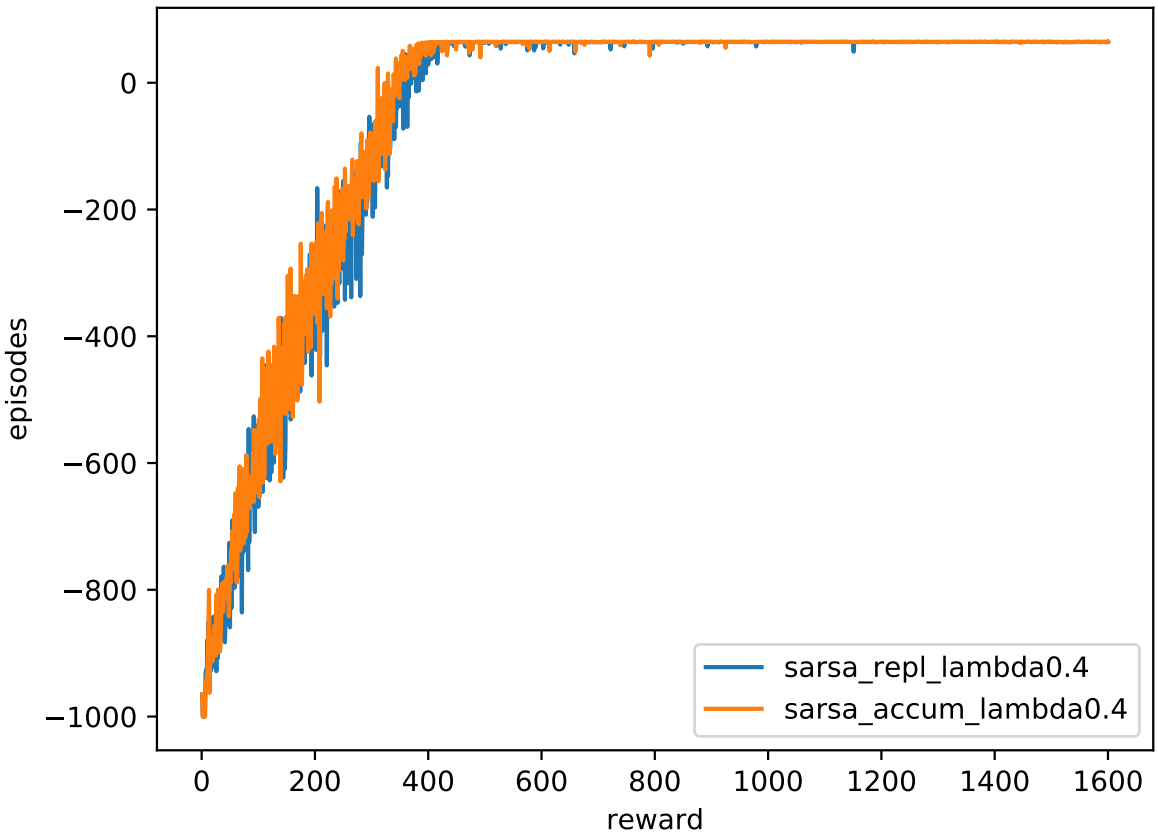
Kalpesh Krishna
140070017
kalpeshk2011@gmail.com

1 Implementation & Results

Both algorithms use a learning rate $\alpha = 0.1$ and $\epsilon = 1/e$, where e is the episode number. Higher values of α lead to faster convergence but with a lower accuracy (for $\lambda > 0.6$). These parameters have been kept constant in this analysis. All the following plots have been averaged across 50 random seeds.

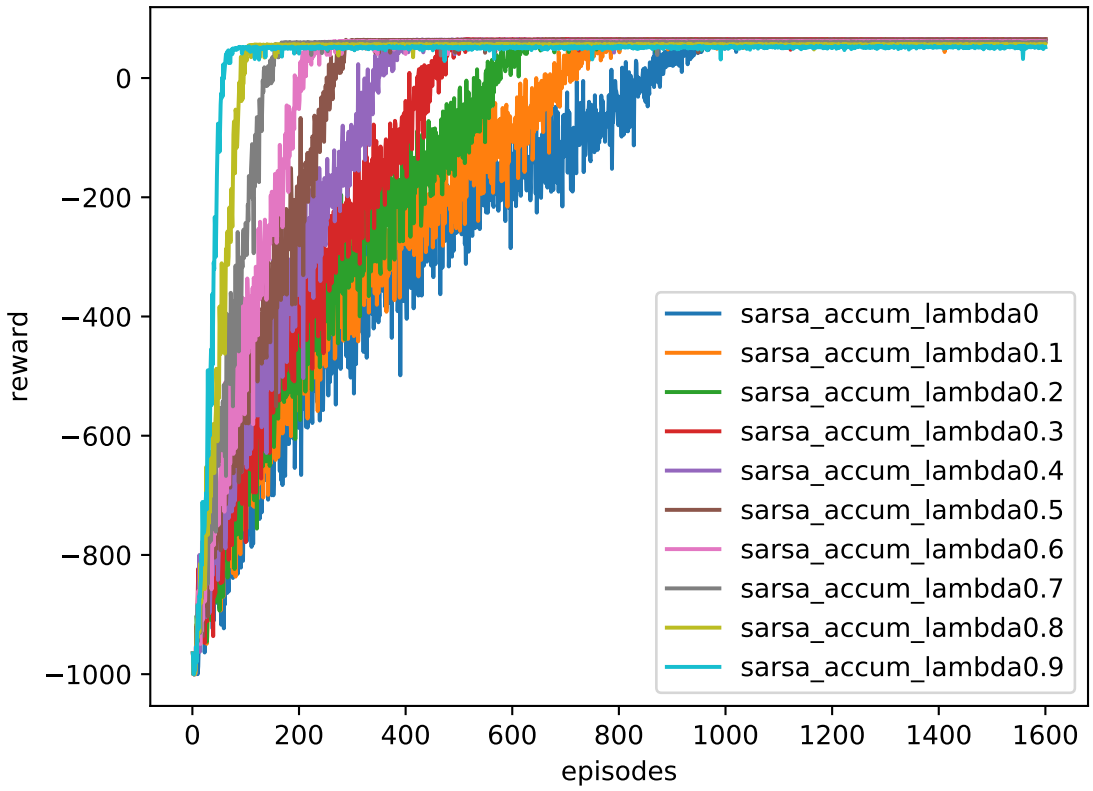
1.1 Accumulating vs Replacing

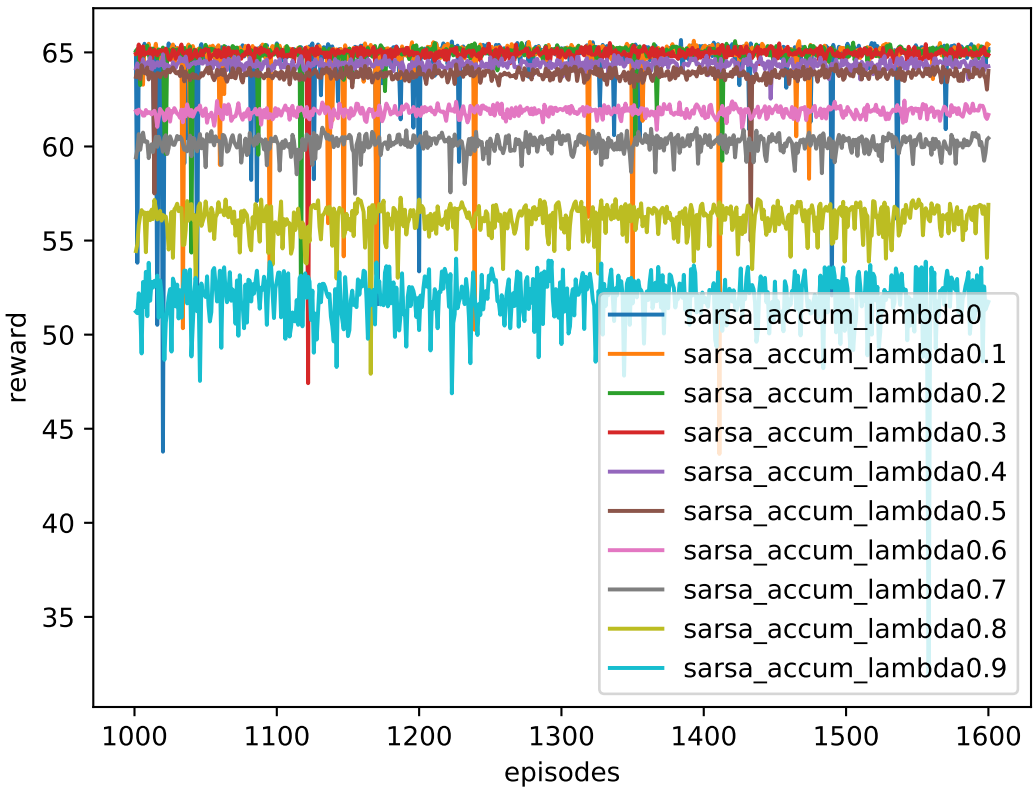
In both instances and across all ten λ values, no significant difference was observed by using replacing traces in place of accumulating traces. Here is the plot for instance 0, $\lambda = 0.4$.

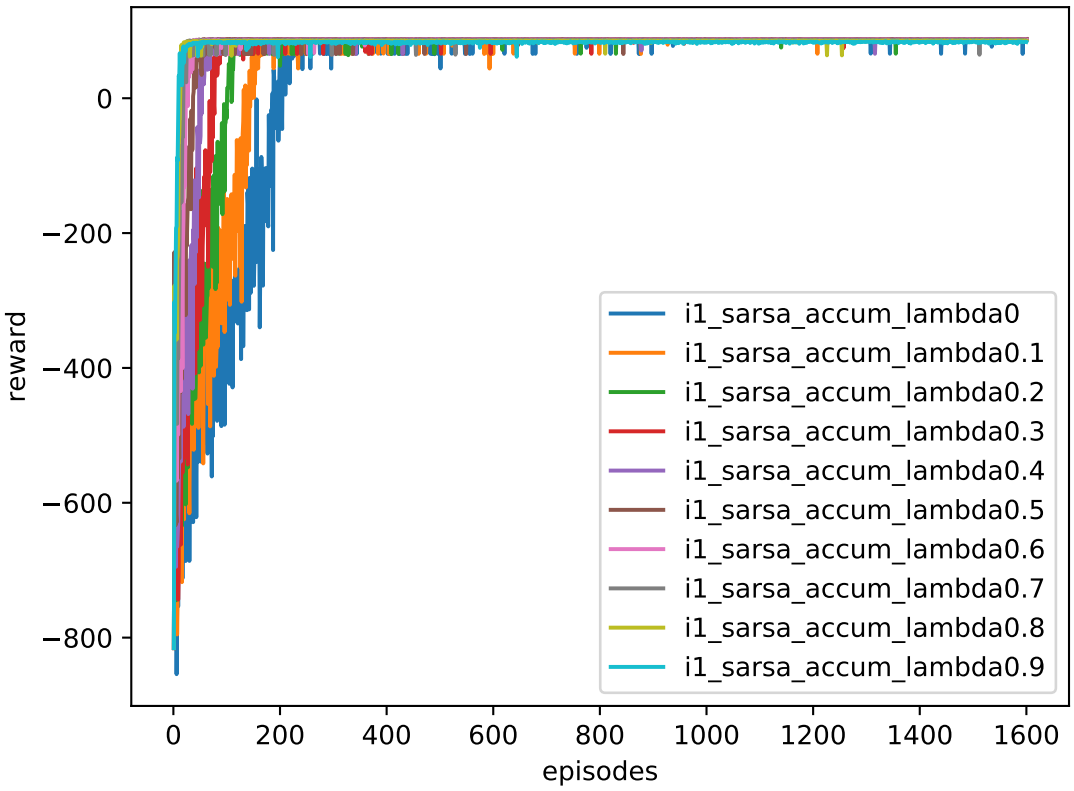


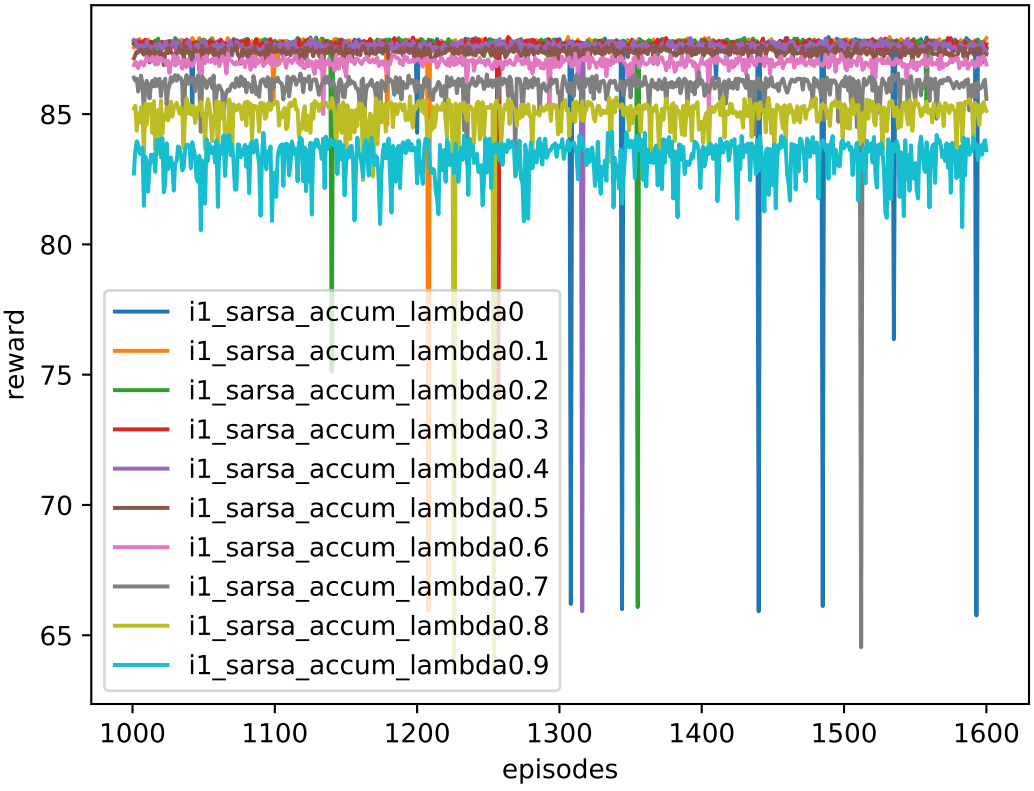
1.2 Variation vs λ

For both instances a similar trend was observed. Here are the plots for instance 0 and instance 1 (each followed by a zoomed in version of the figure after convergence,


reward
episodes
0
−200
−400
−600
−800
−1000
0
200
400
600
800
1000
1200
1400
1600
sarsa_accum_lambda0
sarsa_accum_lambda0.1
sarsa_accum_lambda0.2
sarsa_accum_lambda0.3
sarsa_accum_lambda0.4
sarsa_accum_lambda0.5
sarsa_accum_lambda0.6
sarsa_accum_lambda0.7
sarsa_accum_lambda0.8
sarsa_accum_lambda0.9


reward
episodes
65
60
55
50
45
40
35
1000
1100
1200
1300
1400
1500
1600
sarsa_accum_lambda0
sarsa_accum_lambda0.1
sarsa_accum_lambda0.2
sarsa_accum_lambda0.3
sarsa_accum_lambda0.4
sarsa_accum_lambda0.5
sarsa_accum_lambda0.6
sarsa_accum_lambda0.7
sarsa_accum_lambda0.8
sarsa_accum_lambda0.9

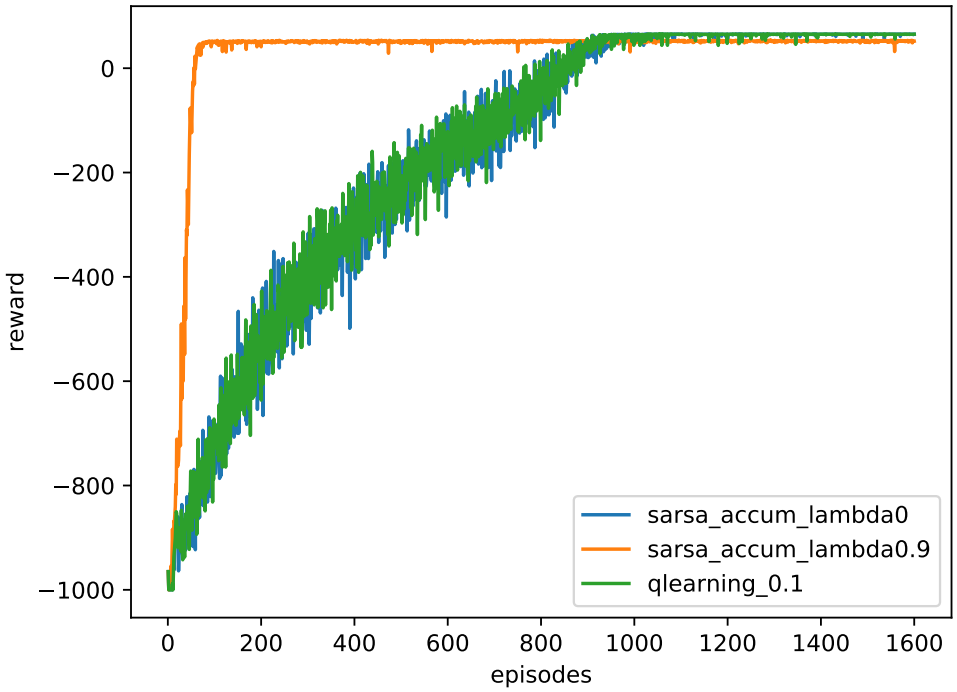

reward
episodes
0
−200
−400
−600
−800
0
200
400
600
800
1000
1200
1400
1600
i1_sarsa_accum_lambda0
i1_sarsa_accum_lambda0.1
i1_sarsa_accum_lambda0.2
i1_sarsa_accum_lambda0.3
i1_sarsa_accum_lambda0.4
i1_sarsa_accum_lambda0.5
i1_sarsa_accum_lambda0.6
i1_sarsa_accum_lambda0.7
i1_sarsa_accum_lambda0.8
i1_sarsa_accum_lambda0.9

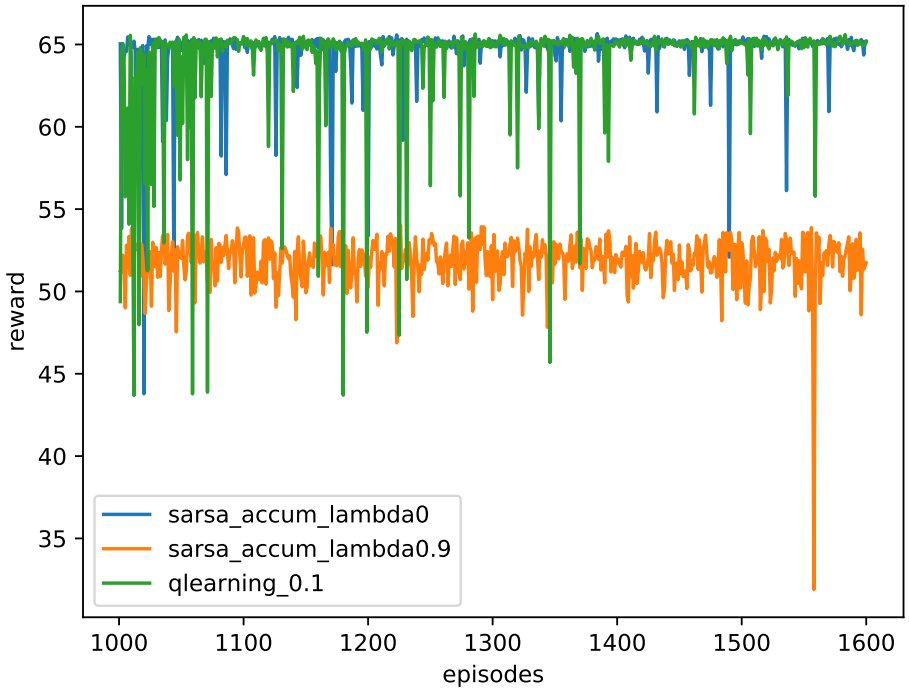

reward
episodes
85
80
75
70
65
1000
1100
1200
1300
1400
1500
1600
i1_sarsa_accum_lambda0
i1_sarsa_accum_lambda0.1
i1_sarsa_accum_lambda0.2
i1_sarsa_accum_lambda0.3
i1_sarsa_accum_lambda0.4
i1_sarsa_accum_lambda0.5
i1_sarsa_accum_lambda0.6
i1_sarsa_accum_lambda0.7
i1_sarsa_accum_lambda0.8
i1_sarsa_accum_lambda0.9

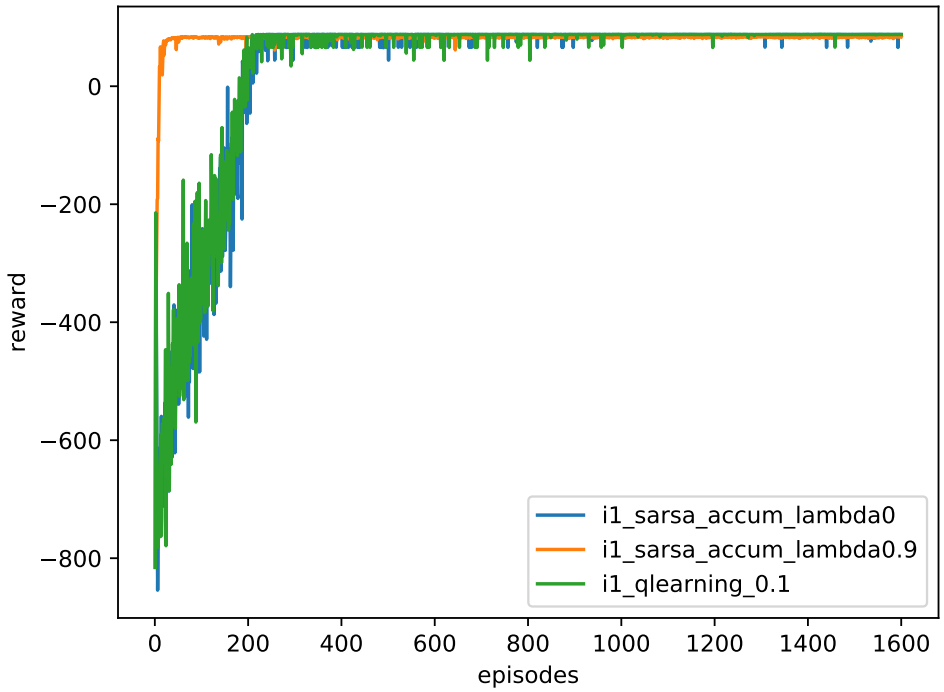
As the graphs indicate, higher values of λ lead to faster convergence but produce sub-optimal rewards. Despite averaging across 50 random seeds, there is a large amount of variation in the curves. For subsequent experiments, we keep $\lambda = 0.0$ and $\lambda = 0.9$ in the plots.

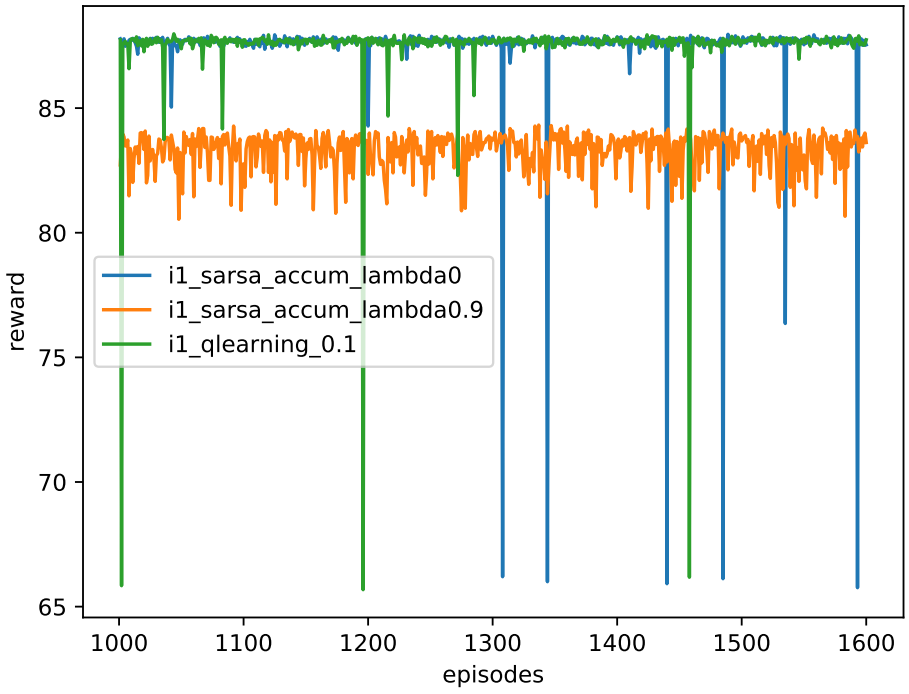
1.3 Q-Learning vs Sarsa

Again, plots for both instances followed by a zoomed in version of the plot after convergence.









In both instances we observe a similar trend - both Q-learning and Sarsa(0.0) converge to the same amount with a similar convergence rate. Sarsa(0.9) converges to a sub-optimal reward faster than both Q-learning and Sarsa(0.0).
This is expected behaviour since Q-learning is similar to Sarsa(0.0) except for the off-policy / on-policy updates. The off-policy update probably converges faster since in the initial stages a sub-optimal policy is being followed and acting greedily with respect to the $Q(s, a)$ makes more sense rather than the action taken by the sub-optimal $\epsilon-$greedy policy.

Faster convergence of higher values of lambda is expected since greater number of $Q(s, a)$ values are being updated in each update.
I was expecting plots to be a lot smoother after 50 random seeds. I'd not expected higher values of λ to converge to a slightly sub-optimal value. I suspect this might be because Sarsa(λ), with high λ, converges to a good policy very quickly, making several updates on $Q(s, a)$. Since Sarsa(λ) is a on-policy method, it makes its updates based on the current action taken, which are derived from a good policy (but not optimal). Due to updates in several dimensions in parallel, a smaller parameter space is explored and the system probably gets stuck at a local minima. A possible solution would be annealing the learning rate.
For smaller values of λ, smaller updates are made on $Q(s, a)$ which slows down convergence. For $\lambda = 0$, only one parameter is updated per iteration. Hence a larger parameter space is being explored, reducing the chances of getting stuck in a local minima.