CS6700 - Reinforcement Learning

Programming Assignment 1

Report

Neil Ghosh
ME17B060

Q1. Implementation of ϵ - Greedy on the 10-armed bandit testbed.

- To implement ϵ - Greedy on the testbed, we have a set of 2000 randomly generated 10-armed bandit problems. For each bandit problem the action values $q_*(a)$, a = 1, 2, ..., 10, were selected according to a normal (Gaussian Distribution) with mean 0 and variance 1. Each bandit problem was run for 1000 plays. The actual rewards are sampled from a normal distribution with mean $q_*(a)$ and variance 1.

- Now, we compare the performance of the testbed with various values of ϵ (0, 0.01, 0.5 and 0.1). The graph shown below is plotted between the obtained reward (averaged over 2000 bandit problems) and time steps. The mean of the arms are estimated by taking the average of all the rewards of that particular arm upto that step.

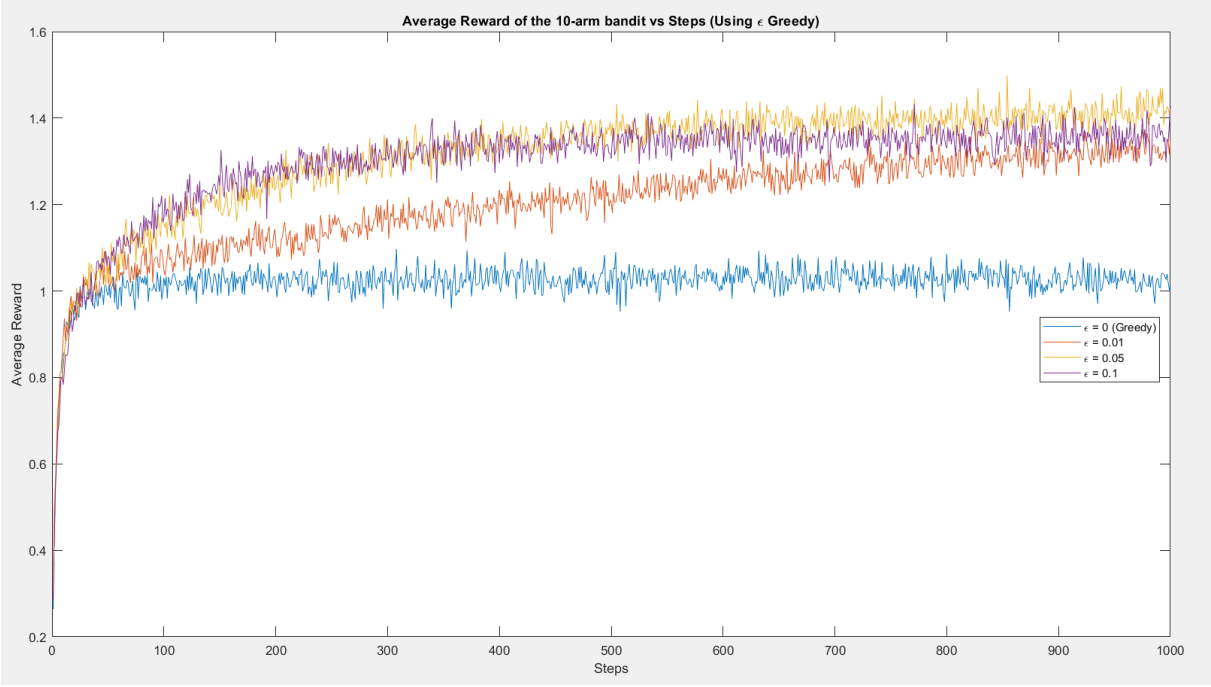

Figure 1: Performance of ϵ-greedy on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. Each bandit problem is run for 1000 plays.

- As is it seen from the graph, the performance is worst for ϵ=0 (Greedy). Initially, it performs better as the non-greedy methods are exploring for some portion of the time which results in poorer rewards for them. Eventually, all the non-greedy methods find out the optimal arm and hence, get higher rewards later on while the greedy method is stuck with sub-optimal arms.

- Amongst the non-greedy methods, ϵ=0.01 has the slowest learning rate as it explores only 1% of the time while exploiting for the remaining 99%. ϵ=0.05 and ϵ=0.1 have similar performance. ϵ=0.05 has a slightly better performance later on as after finding out the optimal arm, it explores only 5% of the time as compared to 10% for ϵ=0.1.

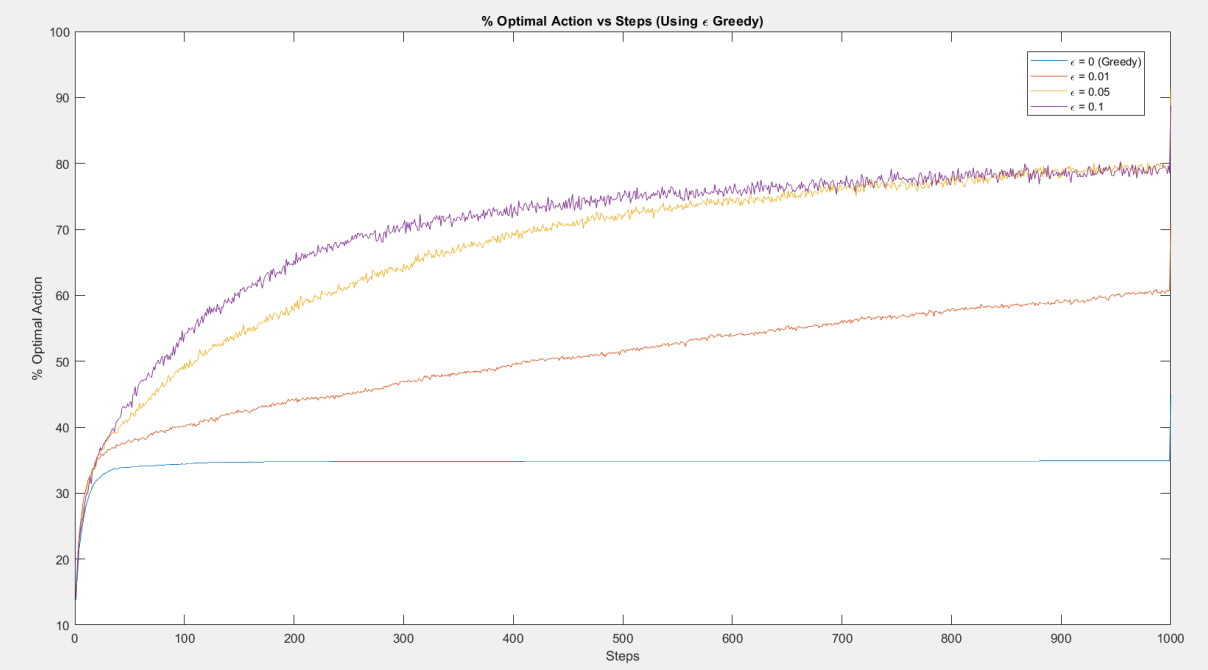

Figure 2: Percentage of times the optimal action is picked (Using ϵ-Greedy).

- The greedy method picks the optimal arm only about one-third of the time because the initial rewards obtained from these arms were poor in most cases. ϵ=0.1 finds out the optimal arm the earliest so, it performs better in the early stages. Later on, after ϵ=0.05 has found out the optimal arm, it starts performing better as it explores less and exploits more as compared to ϵ=0.1.

Q2. Implementation of the Softmax Algorithm on the 10-armed bandit testbed.

- We now implement the softmax algorithm on the same testbed. The performance of the algorithm is compared for different values of temperature (τ = 0.25, 0.5, 0.75 and 1.0). Instead of picking the greedy action, the softmax algorithm selects an arm on the basis of the Gibb's Distribution which is shown below.
- $$\pi_i(t+1) = \frac{e^{\hat{\mu_i}(t)/\tau}}{\sum_{j=1}^{k} e^{\hat{\mu_j}(t)/\tau}}$$
  where, $\pi_i(t+1)$ is the probability of selecting arm "i" at step t+1 and $\hat{\mu_i}(t)$ is the estimate of the mean of the arm "i" at step t. $\hat{\mu_i}(0)$=0 for all i (Initial estimate).
- Softmax is an asymptotic algorithm which means that all the estimates of the arms will converge to their true means as $t \to \infty$.

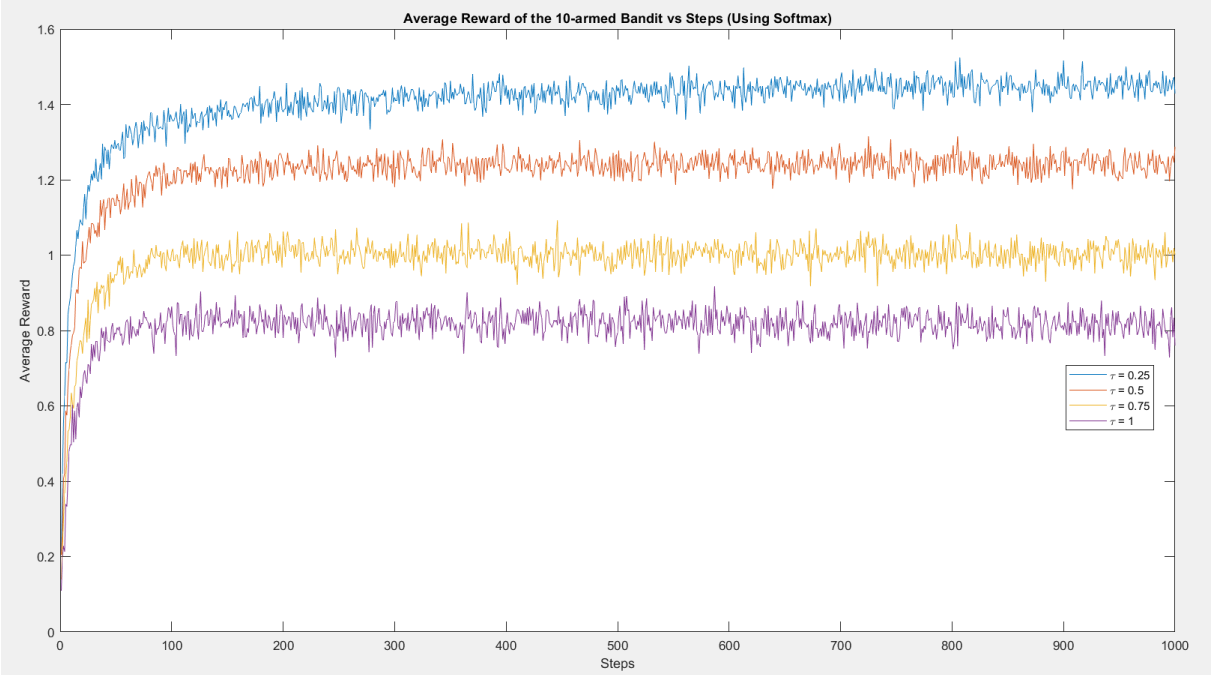

Figure 3: Performance of the Softmax Algorithm on the 10-armed testbed.

- On the basis of the above graph, it is observed that the performance on the testbed depends on the value of temperature (τ). For low values of τ (close to 0) the algorithm will start behaving greedily. The arm with the highest estimate will have a probability close to 1 while all the arms will have a probability close to 0.
- For larger values of τ, algorithm will start picking actions with increased randomness (non-greedy). As $\tau \to \infty$, all the arms will have almost an equal probability of being picked. An average reward close to 0 is expected for such a case.

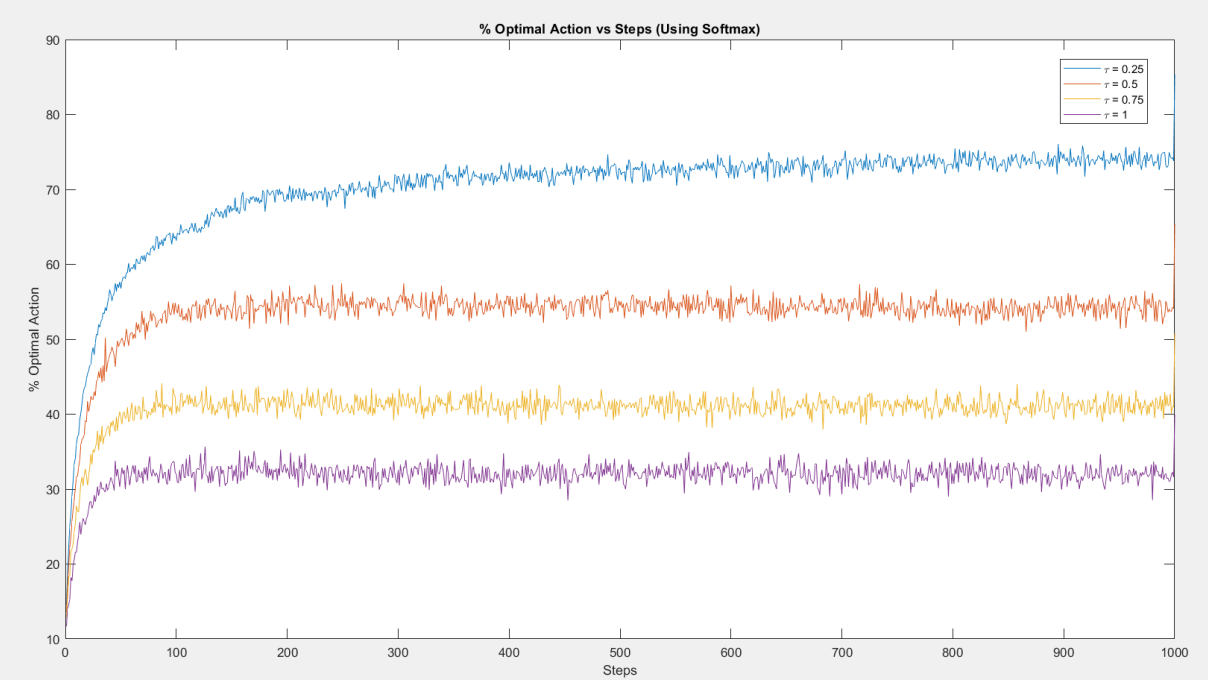

Figure 4: Percentage of times the optimal action is picked (Using Softmax).

- On the basis of the above graph, it is observed that the % of times the optimal action is picked increases with decrease in temperature. This is observation is because for higher values of temperature there is increased randomness which results in the optimal arm being picked fewer times.

Q3. Implementation of the UCB1 Algorithm on the 10-armed bandit testbed.

- Now, we implement the UCB1 algorithm on the same testbed. We compare the performance of UCB1 for different values of c (0.1, 1.0 and 2.0) to ϵ-greedy (ϵ=0.1) and softmax (τ=0.5).

- The UCB1 algorithm is the implementation of OFU (Optimism in the Face of Uncertainity). It uses the fact that we cannot guarantee that our estimate of the mean is very close to the true mean so, we have to build a confidence interval around our estimate where the true mean has a very high probability of lying. Instead of selecting the action with highest estimate, we pick the action with highest upper bound as it has a chance of a higher reward.

- $A_t = argmax_a[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}]$
  where, A_t is the action selected at step t, $Q_t(a)$ is the estimate of arm "a" at time t, $N_t(a)$ is the number of times arm "a" has been picked before step t and c (>0) is a parameter used to control the size of the confidence bounds.

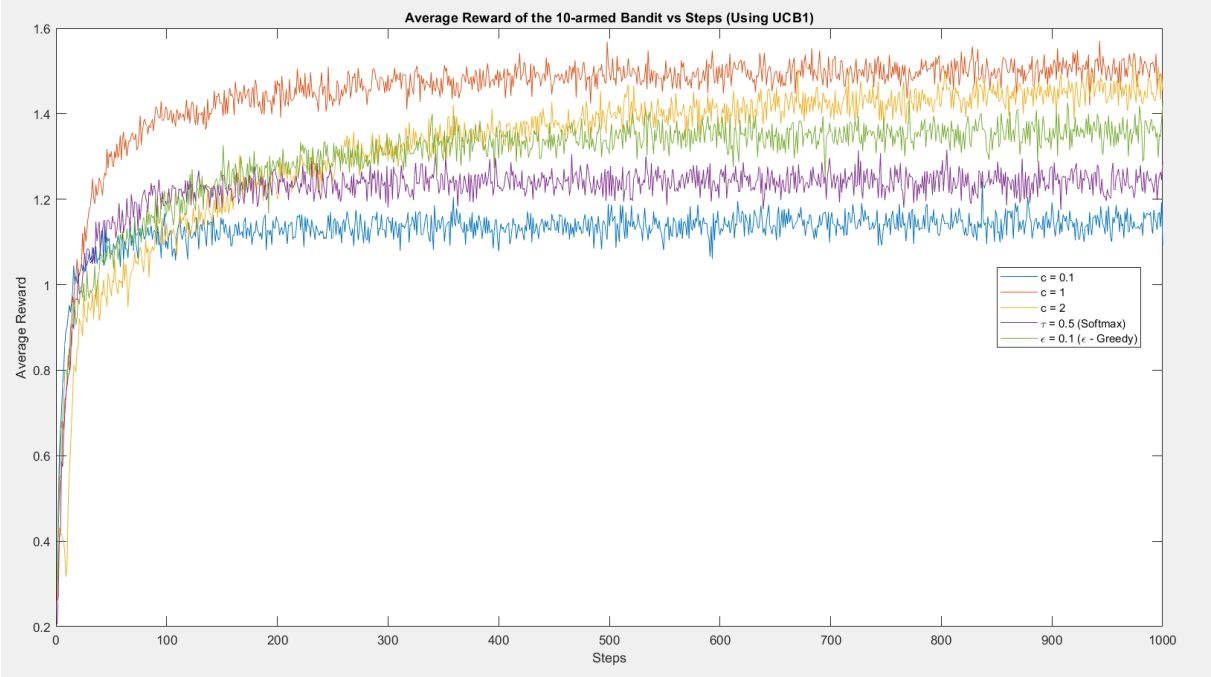

Figure 5: Performance of the UCB1 Algorithm on the 10-armed testbed.

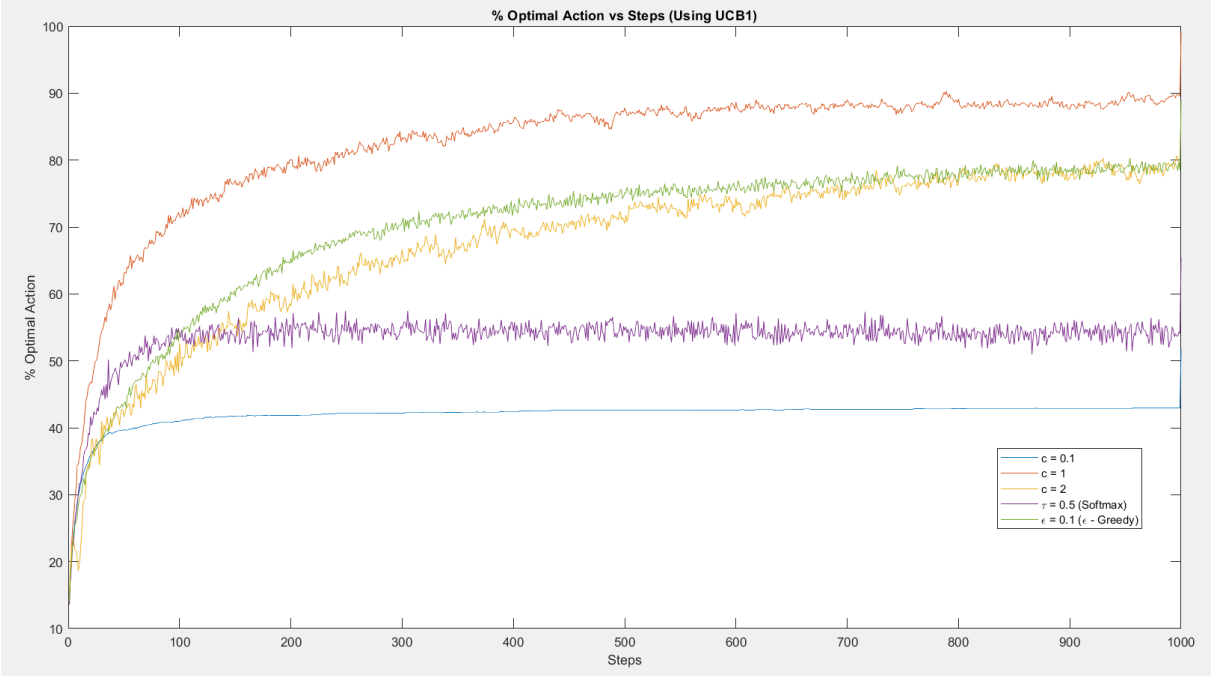

Figure 6: Percentage of times the optimal action is picked (Using UCB1).

- UCB1 is not an asymptotic algorithm because it does not guarantee that all the estimates will converge to the true means as $t \to \infty$, unlike ϵ-greedy and softmax.
- If a action has picked a lot of times, $N_t(a)$ will have a high value which leads to reduction in the confidence band. This is consistent with the fact that on taking more samples from an arm, the estimated mean is closer to the true mean. As the time step increases, the uncertainity term increases and so does the confidence bound.
- UCB1 with c=1 and 2 perform better when compared to ϵ-greedy and softmax algorithms. For low values of c like 0.1, UCB1 behaves similar to the greedy action. Initially, UCB1 will select arms with more randomness as $N_t(a)$ is low for most arms as they have not been picked a lot of times which results in an increased value of the uncertainity term.
- UCB1 keeps on using $Q_t(a)$, $N_t(a)$ and t to vary the exploitation term for each arm with time so, generally it performs better than ϵ-greedy and softmax.

Q4. Implementation of the Median Elimination Algorithm on the 10-armed bandit testbed.

- The Median Elimination Algorithm is an (ϵ, δ) - PAC Algorithm. This means that the algorithm returns an action which is within ϵ of the optimal action with atleast $1 - \delta$ probability.

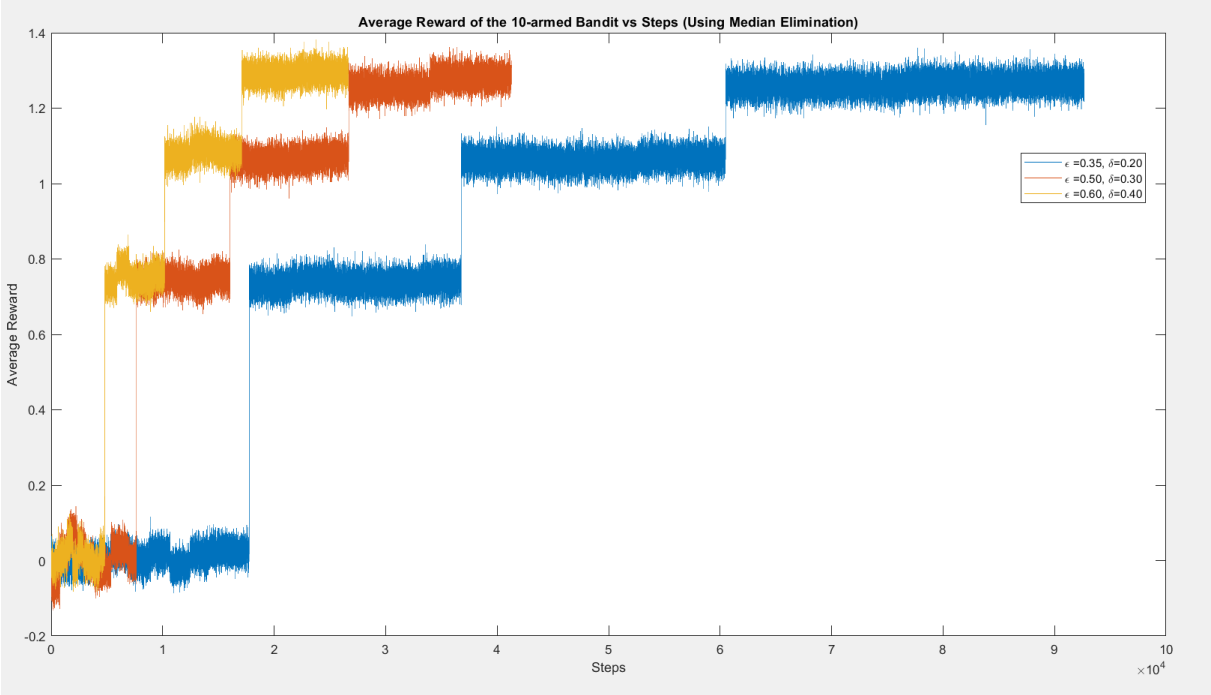

Figure 7: Performance of the Median Elimination Algorithm on the 10-armed testbed.

- The sample complexity of the algorithm is $O(\frac{n}{\epsilon^2}\log\frac{1}{\delta})$ where, n is the no. of arms. In every round it eliminates half of the arms so, the algorithm runs $\log(n)$ times.
- It is observed from the above graph that on increasing the value of ϵ, we require less number of samples as we have a large band where the returned action is close to the optimal action. Similarly, on reducing the value of δ, we require more number of samples as the returned action has to ϵ close to the optimal action with atleast $1-\delta$ probability.
- The computational cost of the median depends on the sorting algorithm used to find the median. Numpy's method for computing the median uses quicksort which has complexity of $O(n\log(n))$. Since, the number of arms are less (n=10), this is not the rate determining step. But, if we increase the number of arms to a high value (above 1000), it is better to use sorting algorithms having complexity of $O(n)$.

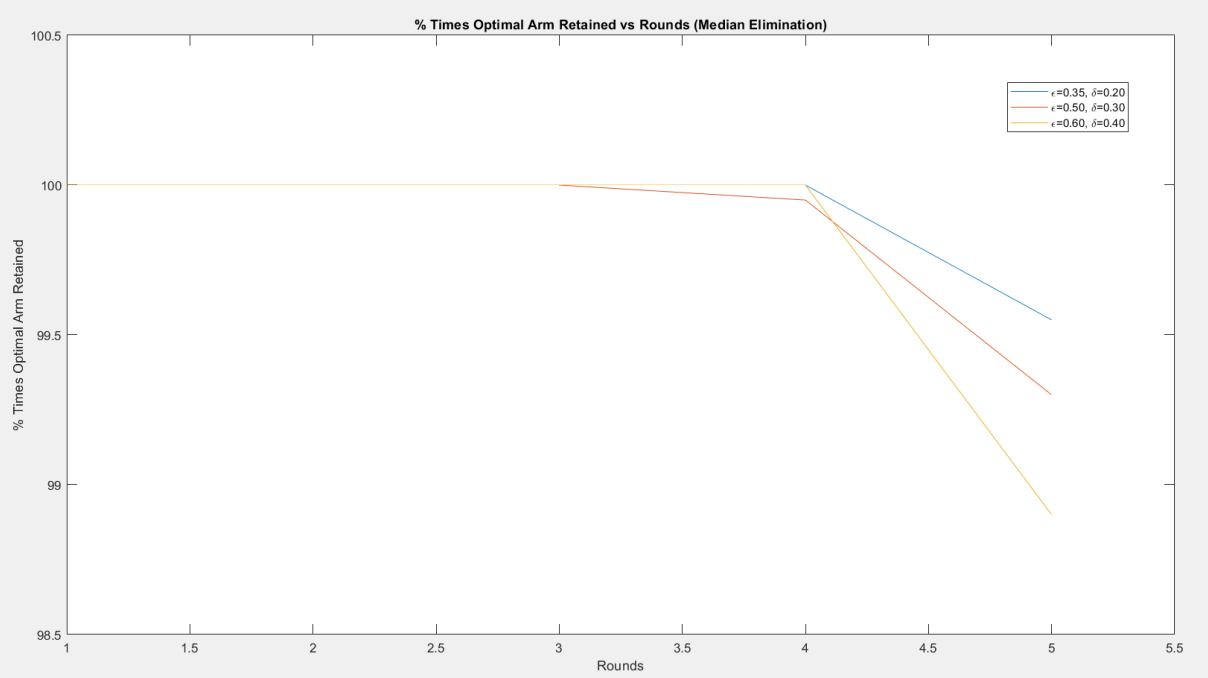

Figure 8: % of times MEA retains the Optimal Arm.
(Averaged over 2000 10-arm bandit problems)

- It is observed from the above graph that for higher values of ϵ and δ, the % of times the optimal arm is retained is relatively less as we are creating a large band around the optimal arm in which the returned action can lie.
- The Median Elimination Algorithm is compared with UCB1, Softmax and ϵ-Greedy in the above graph. MEA is an action-elimination algorithm which means that in every step it tries to determine which arm to eliminate while all the other algorithms try to determine which action to pick in every step.

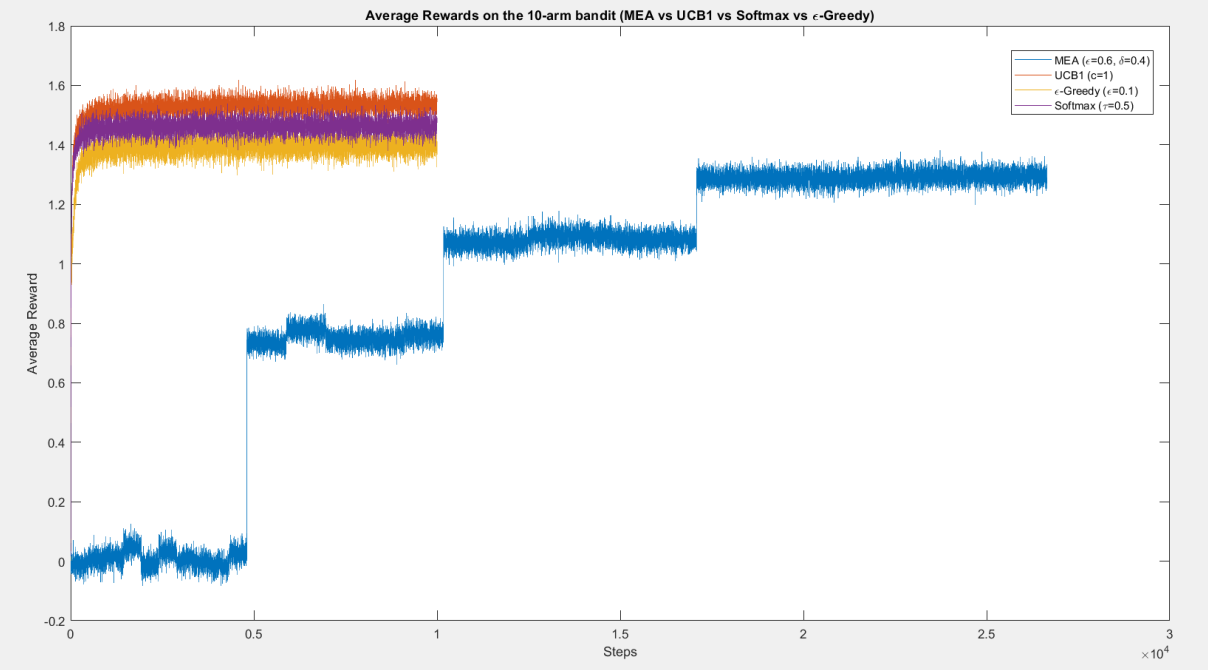

Figure 9: Performance of MEA vs UCB1 vs Softmax vs ϵ-Greedy on the 10-arm testbed. (UCB1, Softmax and ϵ-Greedy were run for 10000 steps for the comparison)

- MEA can be used when the number of pulls is large as it can find out the optimal arm and then always select the optimal arm for the remaining time. If the number of pulls is less than the sample complexity for MEA, it is better to use some other algorithm like UCB1 as it will have less value of regret.

Q5. Implementation of the algorithms on the 1000-armed bandit testbed.

- The performance of UCB1, Softmax and ϵ-Greedy Algorithms are shown in the graph below. The time steps is increased from 1000 to 10000 for the algorithms to correctly identify the optimal arm. The reward plotted is the average of 2000 1000-armed bandit problems.

- It is observed that ϵ-Greedy and Softmax perform better than UCB1 initially as they exploit with more probability than they explore which results in higher rewards. UCB1 initially tries to select those arms randomly which have not been pulled enough which results in lower immediate rewards.

- In the long run, UCB1 starts performing better as ϵ-Greedy and Softmax are not able to explore enough as there are a large number of arms so, they keep on pulling sub-optimal arms.

- It is observed from the graph below that UCB1 picks the optimal arm about 90% of the time at the later stages while ϵ-Greedy and Softmax pick the optimal arm about 35% and 20% of the time respectively.

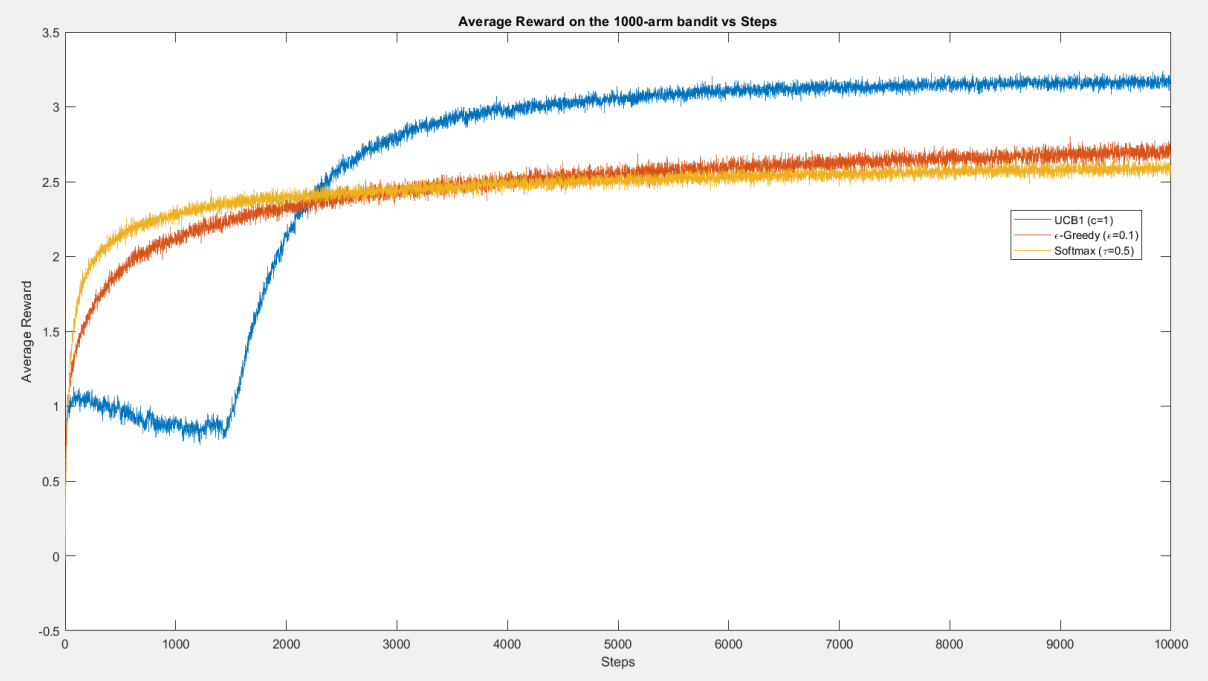

Figure 10: Performance of the UCB1, Softmax and ϵ-Greedy on the 1000-arm testbed.

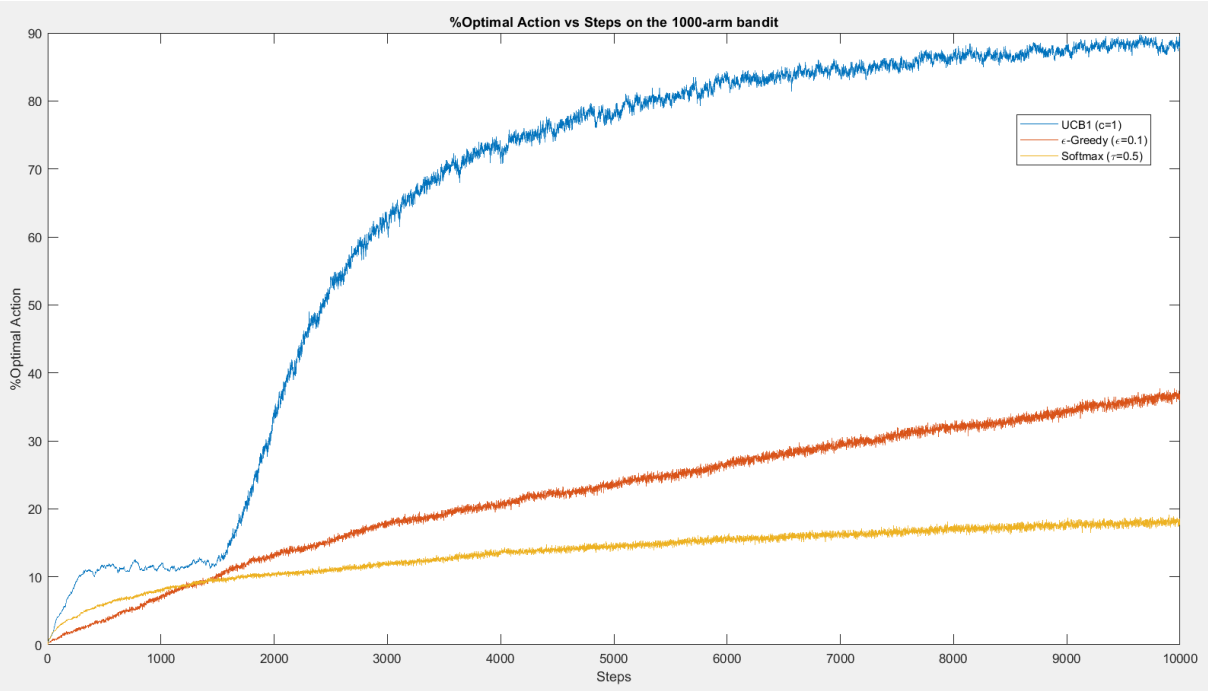

Figure 11: Percentage of times the optimal action is picked for the 1000-arm testbed.

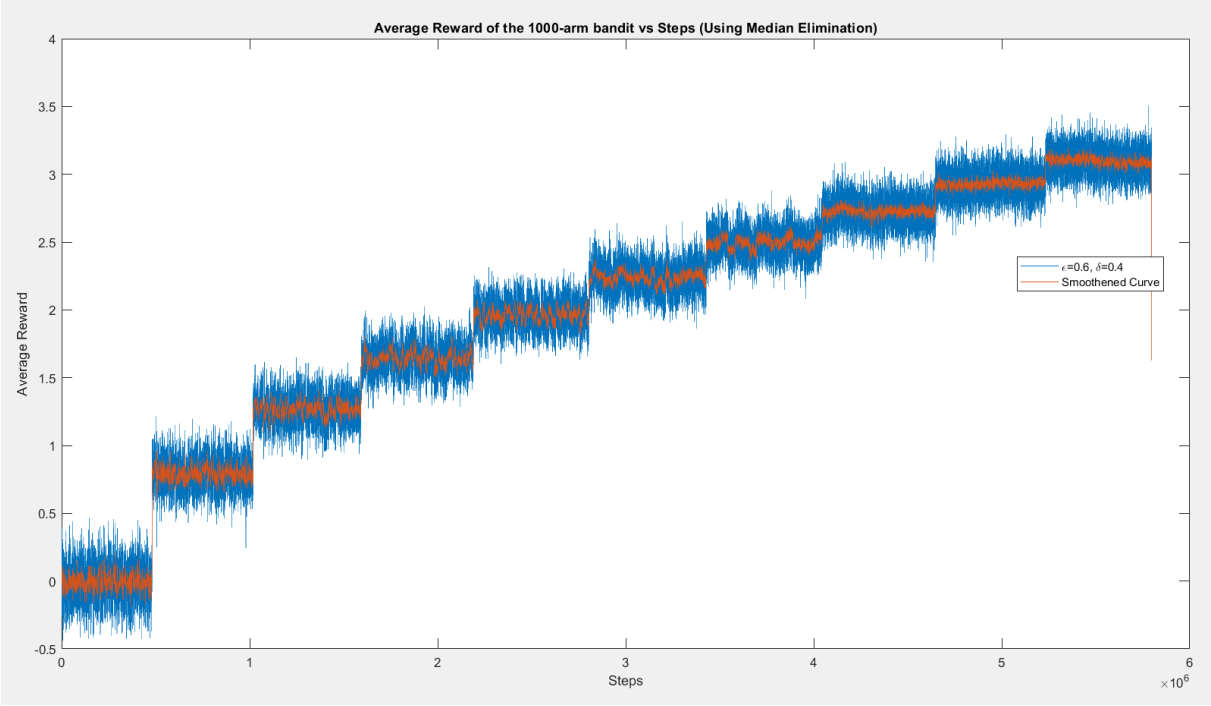

Figure 12: Performance of the Median Elimination Algorithm on the 1000-armed testbed.

- Now, the Median Elimination Algorithm is implemented on the 1000-arm testbed. The values of ϵ and δ are taken as 0.6 and 0.4 respectively which results in about 5.8 million steps. The reward which is plotted is averaged over 200 1000-arm bandit problems. The obtained rewards have been smoothened to reduce the noise.
- Similar to the 10-arm case, on reducing the value of ϵ and δ the number of steps are increased. A large value of (ϵ, δ) is chosen to reduce the computational cost.