Date: 04/02/2020

Programming Assignment 1
Report

CS6700: Reinforcement Learning Prajjwal Kumar—NA16B115

Question 1

EPSILON-GREEDY ALGORITHM

1 Solution

1.1 ALGORITHM:

First,we choose each arm of all bandit at least once and find the optimal arm of each bandit. Then, for further runs,with (1-epsilon) probability, we choose optimal arm(arm having highest reward estimate i.e. EXPLOITATION and with epsilon probability, we choose any arm randomly i.e. EXPLORATION and thus we update the reward estimates.

1.2 PLOTTING:

The actions are chosen according to **probability values** in one run of all bandits and average of their rewards is taken over the 2000 bandits. And these rewards are plotted against runs. This graph is plotted for **different epsilon(ϵ) values**.
For example, if arm-1 is chosen in bandit-1 and arm -4 in bandit-2 in 1st run, we take the average of those 2 and plot it against 1st run.

1.3 TIMING:

For epsilon (ϵ)	Total Computing Time
0.000000	40.340857
0.100000	40.717481
0.200000	41.129973
0.300000	41.499757
1.000000	42.763742

Table 1: Computation cost at different ϵ values

1.4 EXPLANATION:

As on one run, we are plotting average of rewards of action chosen of all bandits, it is going to improve along the way as our algorithm learns which action to choose to get max reward.
For different epsilon values, like ϵ = **0.1,0.2 and 0.3**, it's 90 percent, 80 percent and 70 percent exploitation i.e. choosing the best arm till now. So, the **graph goes towards optimality very fast but then stays under it for a very long time**.
For ϵ = **0**, it's completely **exploitation** i.e. always choosing the best arm that leads to very high average reward in start but the chances of reaching optimality in this case is very less as it' only possible when algorithm picks the best arm in first run only.

For $\epsilon = \mathbf{1}$, it's completely **exploration** i.e. choosing the arms randomly without any algorithm, so the rewards of it stay less only and same is reflected in graph.

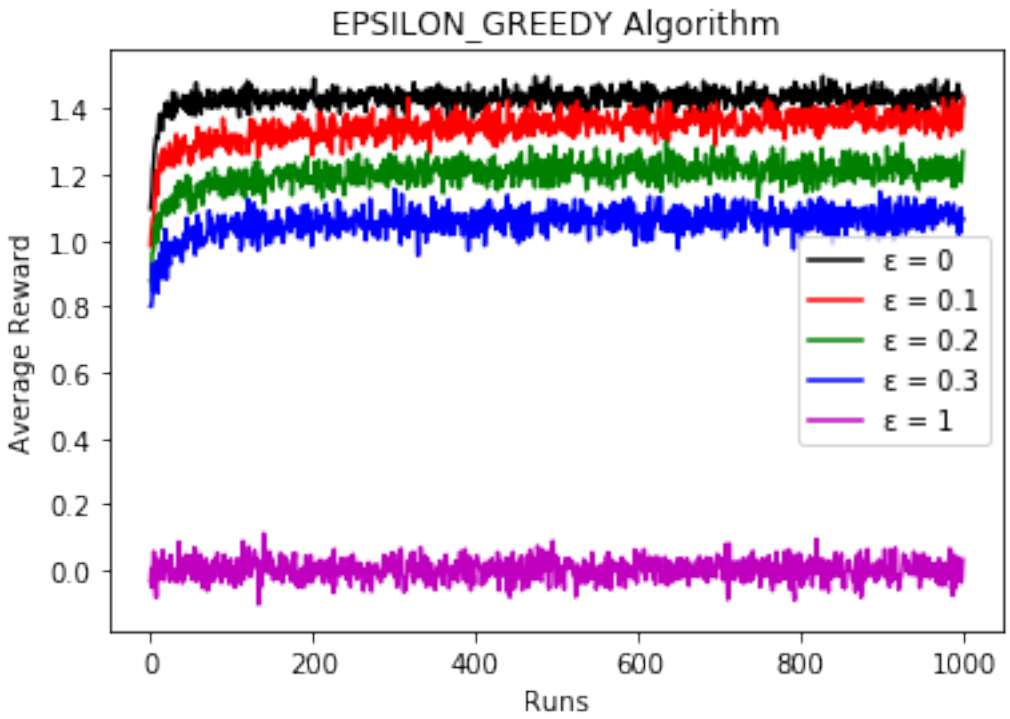

Figure 1: Average perfomance of Epsilon-Greedy algorithm on the 10-armed testbed

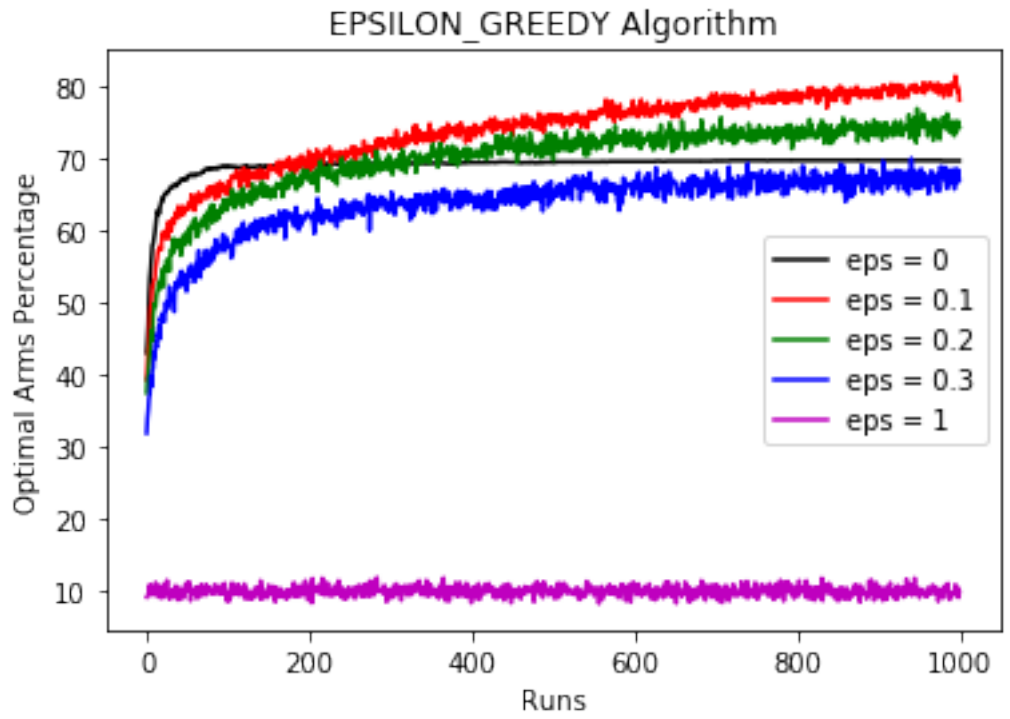

Figure 2: Optimal arms percentage of Epsilon-Greedy algorithm on the 10-armed testbed

Question 2

SOFTMAX

---

2 Solution

2.1 ALGORITHM:

Here, we are applying softmax arm selection method using the Gibbs distribution on the 10-armed testbed.
We choose the arms on the basis of probabilites. These probabilities are calculated using softmax function. We calculate it by dividing exponential to the power of Q/T where Q = estimated reward for each actions and T = Temperature.
Probabilities of each action at time step t, given by

$$P_t(a) = \frac{e^{Q_t(a)/T}}{\sum_i e^{Q_t(a_i)/T}}$$

2.2 PLOTTING:

The actions are chosen according to **probability values** in one run of all bandits and average of their rewards is taken over the 2000 bandits. And these rewards are plotted against runs. This graph is plotted for **different temperature values**

2.3 TIMING:

For temperature (T)	Total Computing Time
0.010000	205.366792
0.020000	203.711631
0.100000	204.407109
1.000000	205.043371

Table 2: Computation cost at different Temperature values

2.4 EXPLANATION:

As we can see in the above graph, T = 0.1 reaches highest average reward and T = 1 is lowest one. From this, we can say that T = 1 randomly picks arms and thus has lowest average rewards and T = 0.1 keeps a balance between exploration and exploitation and gives highest average value.
It is so because when **temperature is very high, probabilities of choosing all actions will get very small** and thus it becomes **random selection**.
While in case of **low temperature (as close to 0)**, numerator part in the softmax function i.e. **reward estimate becomes dominant for deciding probabilities** and thus it keeps a **balance between exploration and exploitation.**

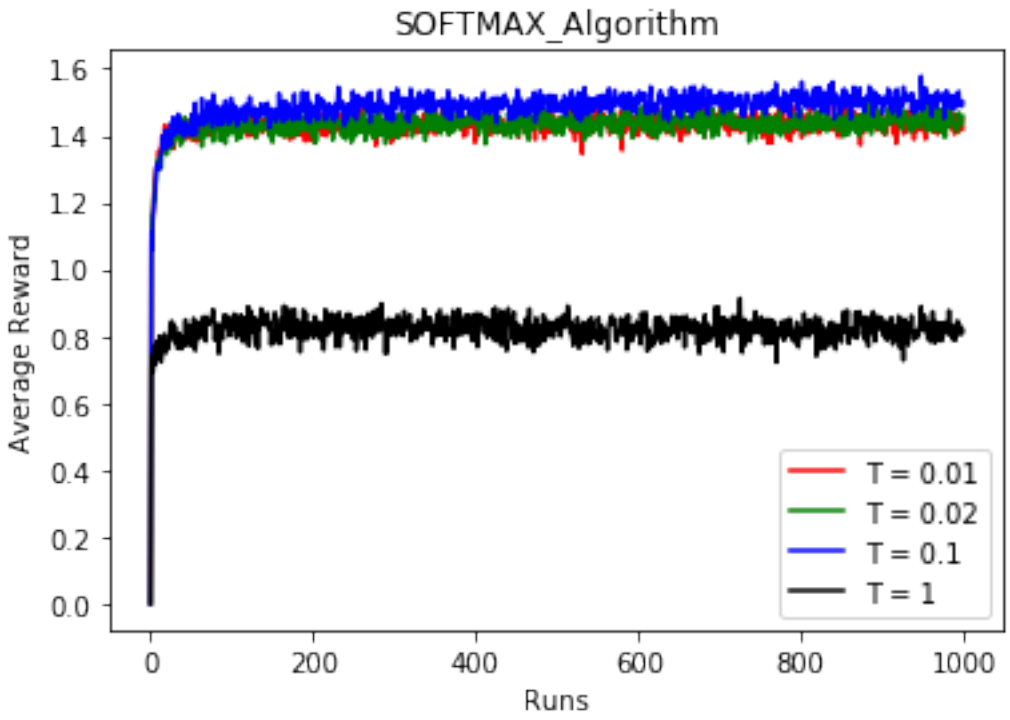

Figure 3: Average perfomance of Softmax algorithm on the 10-armed testbed

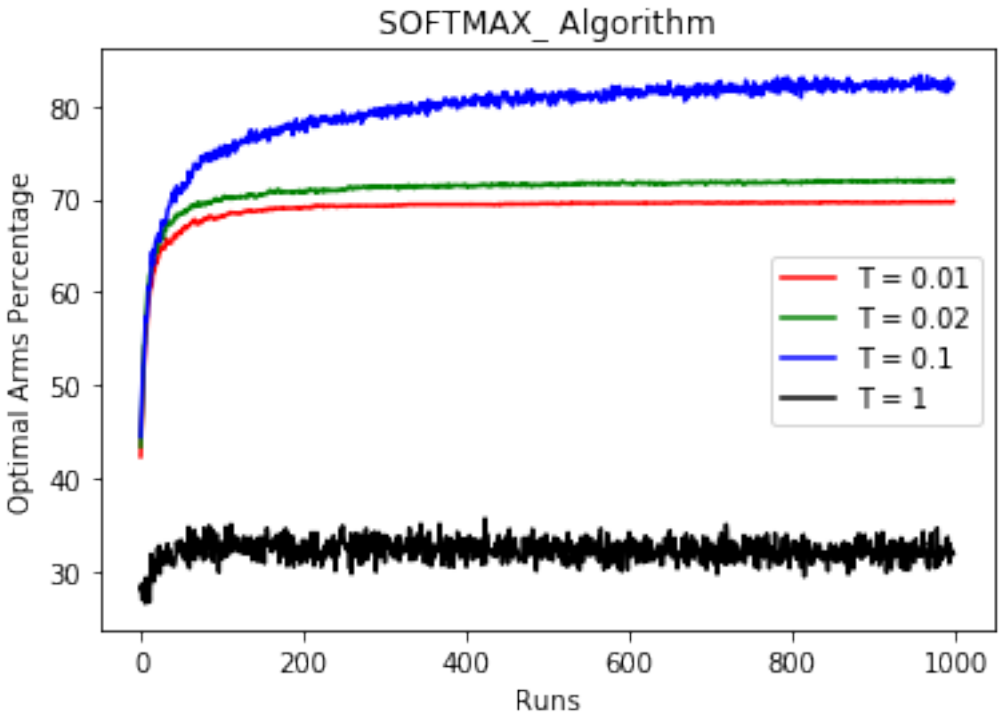

Figure 4: Optimal arms percentage of Softmax algorithm on the 10-armed testbed

Question 3

UCB1

---

3 Solution

3.1 ALGORITHM:

The algorithm identifies the upper confidence bound of each arm of the bandit. First, we pick all the arms once to get estimates and then we pick optimal arms using UCB1 algorithm. Reward estimates are calculated using a function. The parameter 'score' includes the estimate of an arm 'a' at any time step is calculated using the formula given by:

$$score(a) = Q_t(a) + \sqrt{\frac{c \log(t)}{n_a}}$$

In it, the variable (n-a) is denoting no of times an action is chosen till that run. So, if we don't choose all actions once in start, we will get a form of (1/0). That's why in first 10 runs, 10 arms are chosen and thus, average rewards start increasing after first 10 runs as then we choose optimal arms using UCB1 algorithm.

3.2 PLOTTING:

The actions are chosen according to **score values** in one run of all bandits and average of their rewards is taken over the 2000 bandits. And these rewards are plotted against runs. This graph is plotted for **different c(degree of exploration) values**. We are plotting from 11th run as first 10 runs are simply chosen to get estimates.

3.3 TIMING:

For Degree of Exploration (c)	Total Computing Time
0.100000	72.642862
2.000000	76.317130
5.000000	75.44480

Table 3: Computation cost at different 'c' values

3.4 EXPLANATION:

As c denotes degree of exploration, low values of c results into exploitation making this a greedy algorithm and high values of c shows exploration.
Same thing is reflected in average reward graph as c = 2 graph has slightly higher average reward than other 2.
In optimal percentage graph, again in c=2 case, it improves very significantly in comparison to others.

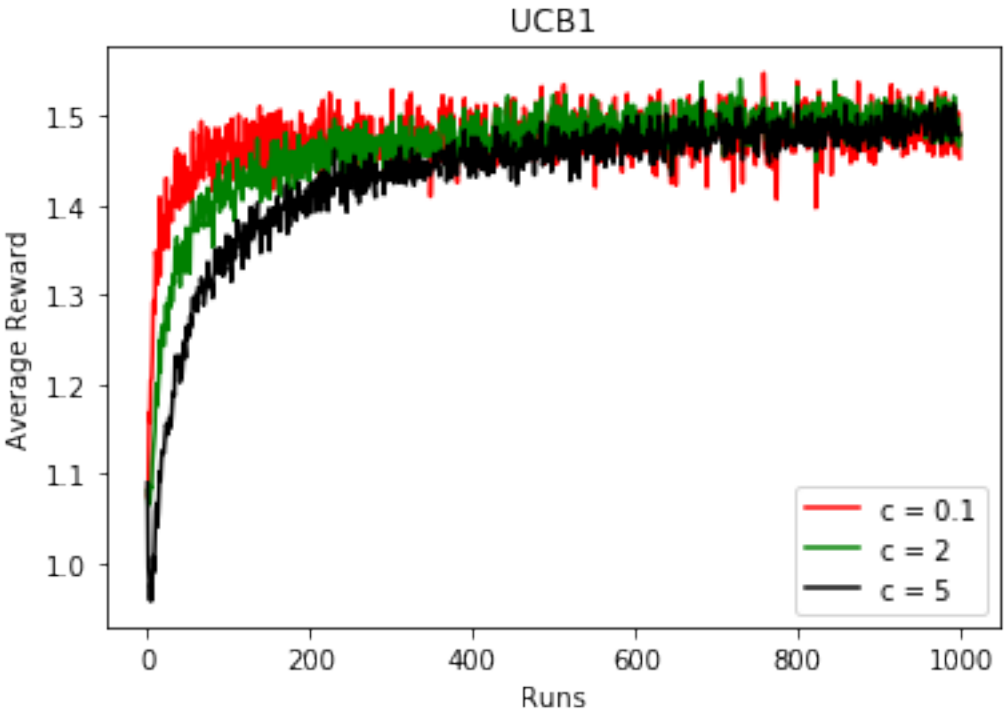

Figure 5: Optimal Percentage of UCB1 algorithm on the 10-armed test bed

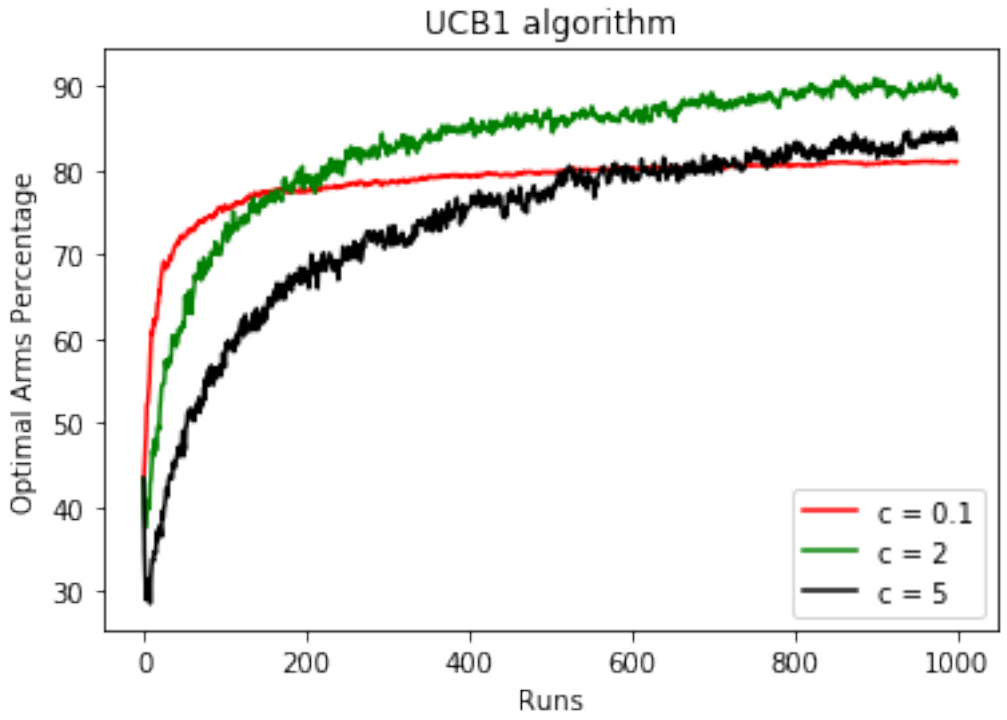

Figure 6: Average performance of UCB1 algorithm on the 10-armed test bed

3.5 COMPARISON WITH Epsilon Greedy and Soft max

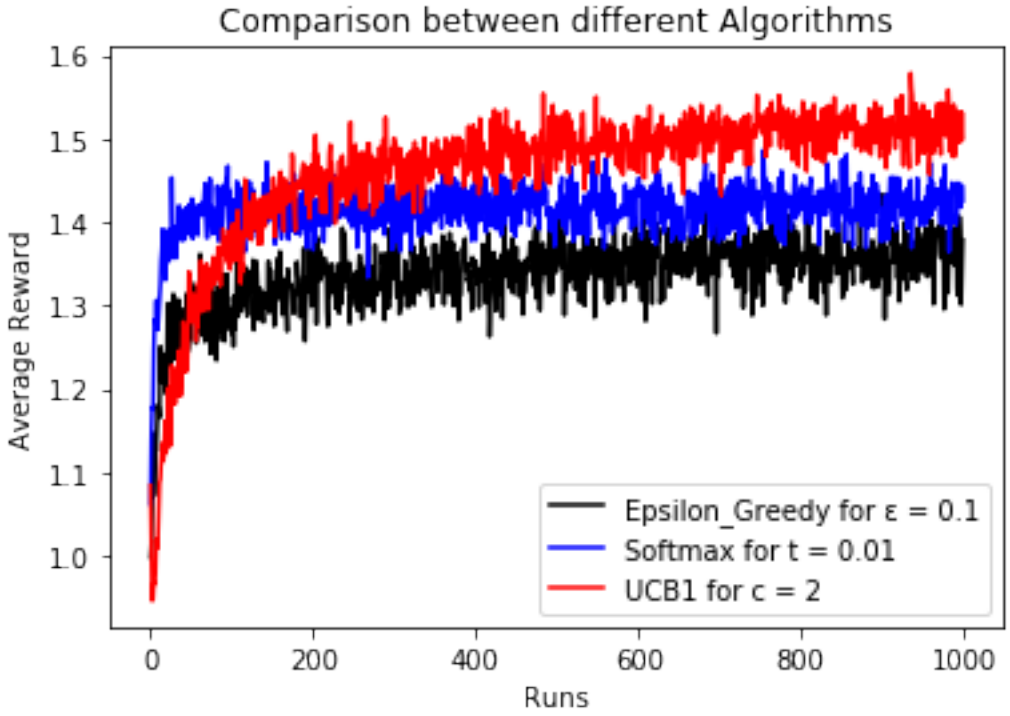

Figure 7: Average performance of different MAB algorithms

As seen in graph, UCB1 performs best and epsilon greedy shows lowest average reward.
It is so because as in epsilon greedy, sometimes we pick random arms even though they have very less rewards. while in case of UCB1, we always choose according to formula given in it and highest estimates. So, it is better than epsilon greedy one.

Question 4

MEDIAN ELIMINATION

---

4 Solution

4.1 ALGORITHM:

In this, we sample all actions of each bandit a specific no of times which is given by:

smpl = log10(3.0/delta)*4/(eps**2)

Then for every bandit, we pick only those actions for next round which have rewards values lesser than the median value of that bandit and then we sample those actions against "smpl" no of times.

4.2 PLOTTING:

One run is defined when all actions are chosen for a bandit. Then we take the average of reward values of all actions of all bandits in that run and plot it against the runs. This is so because we get an optimal arm list when we remove the actions lesser than median values. This graph is plotted for **different epsilon(ϵ) and delta(**δ values

4.3 TIMING:

Epsilon	Delta	Median Computing Time	Total Computing Time
0.6	0.4	0.001764	2.312055
1.2	0.6	0.002094	0.562181
1.2	0.8	0.002472	0.522198

Table 4: Computation cost of Median Elimination method

As we can see in above table, the median computing time in different ϵ and δ parameters is very less than total computing time. So, we can say that **median computing is not a rate determining step** for this algorithm.
We can see that when ϵ and δ values are lowest, the computation time is highest. So, for making the algorithm faster, we can take **high ϵ and δ values.**

4.4 EXPLANATION:

The graph is step-wise increasing. As we plot the average rewards of all actions over all bandits against, it represents the horizontal line. And sudden increase in average reward indicates the run when we drop the arms having reward estimates lesser than that of median and keep only optimal arms. Also, different values of epsilon and delta give different no of runs. Among different values of ϵ and δ parameters of the algorithm, the average reward can be seen being improved significantly with $\epsilon = 0.6$ and $\delta = 0.4$.

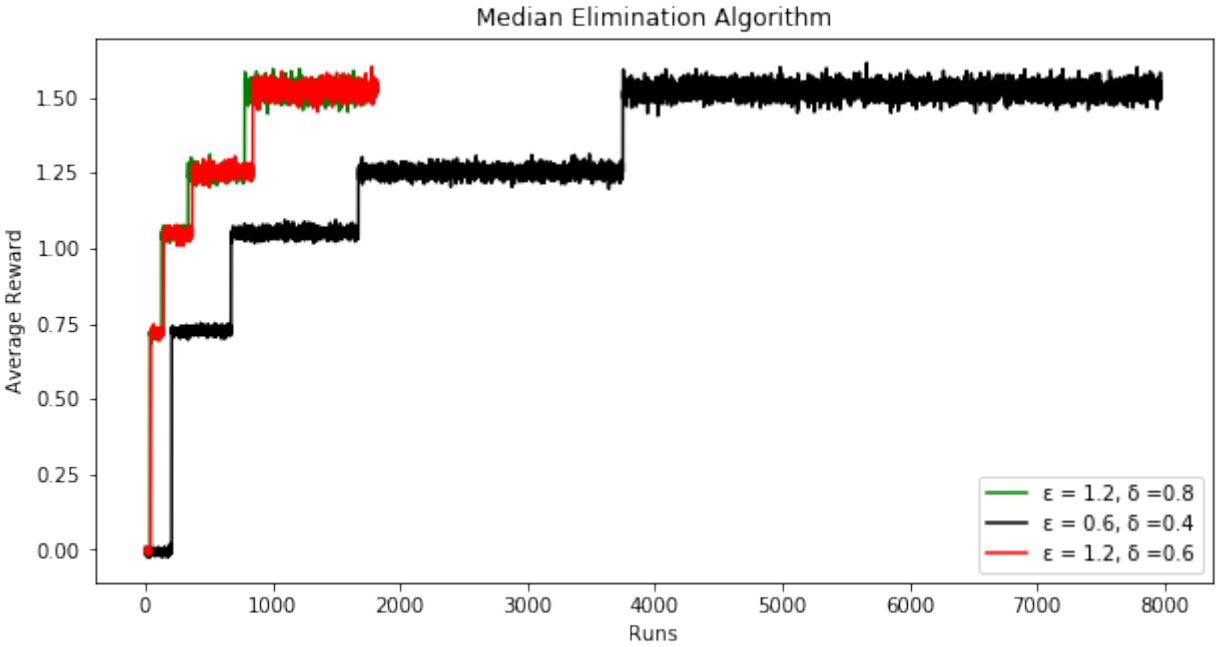

Figure 8: Average perfomance of Median Elimination algorithm on the 10-armed testbed

4.5 Comparison of Median Elimination with other MABs algorithms

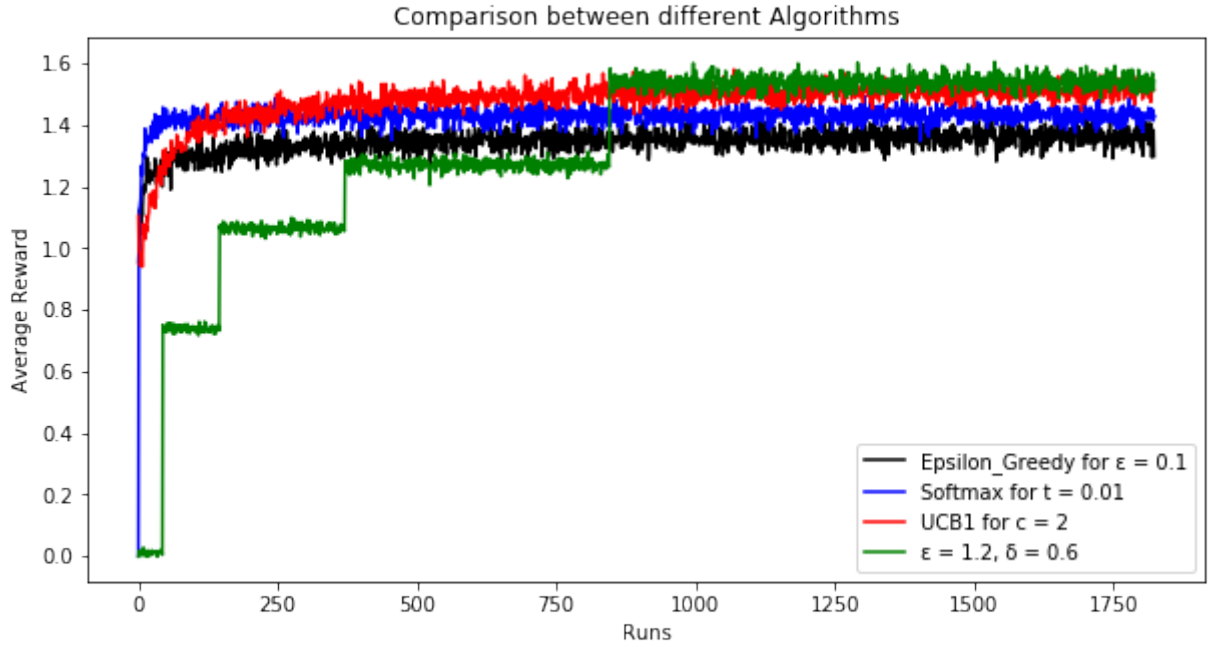

Figure 9: Average perfomance of all MABs Algorithms on a 10-armed testbed

Average reward curve of **median elimination reaches the optimal before other algorithms**. Other algorithm also converges to optimal but may need more number of iterations.

Question 5

1000 Arm-bandit setup

---

5 Solution

In all algorithm, we make a reward distribution of all arms in each bandit. So, when based on some algorithm, an arm is selected and rewards are generated from the distribution we made. For selecting an arms, we need to calculate estimates of rewards which is done by iterating it for a no. of times(like 1000) for each arm.
When no of arms of a bandit grows (say to 1000 here) while we have same no of iterations i.e. 1000, any algorithm won't have enough iterations to make reward estimates for all arms. So, we won't know which arms are optimal and won't be able to pick them. So, we need to **increase no of iterations a lot** so that we get proper estimates of rewards and when we do that, **the computational cost of the algorithm will be very high.**
