Reinforcement Learning Course Project

Partha Sarathi Mohapatra
EE18D703

4th February 2020

The 10-arm bandit testbed was implemented to compare the performance of different algorithm, where for each bandit problem the true action values ($q_*(a)$, for $a = 1, 2, ..., 10$) were selected from a Gaussian distribution having mean 0 and variance 1 i.e. $\mathcal{N}(0, 1)$. And for choosing any action a the actual reward was sampled from the normal distribution with mean $q_*(a)$ and variance 1 i.e. from $\mathcal{N}(q_*(a), 1)$. This run was done for 2000 different bandit problems and the average of these were used for comparison.

1 Greedy and ϵ-greedy Action Selection Method

1.1 Average Performance and Optimality Plots

First we applied both greedy and ϵ-greedy methods (with $\epsilon = 0.1$ and 0.01) to the 10-arm bandit testbed and compared the results in the following plots.

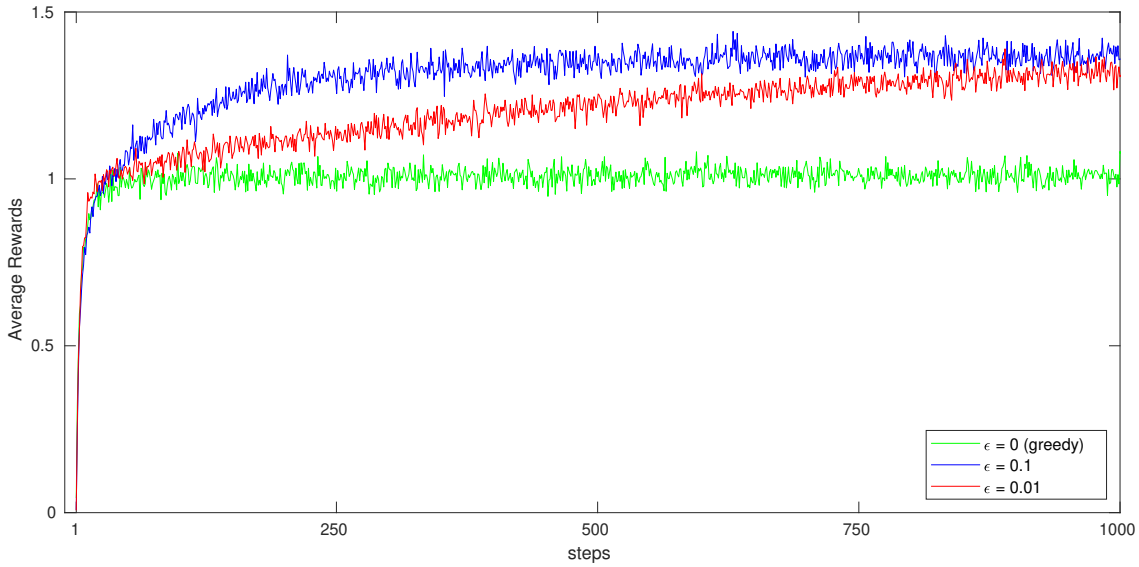

Figure 1: Average reward comparison of greedy and ϵ -greedy for the 10-arm bandit testbed

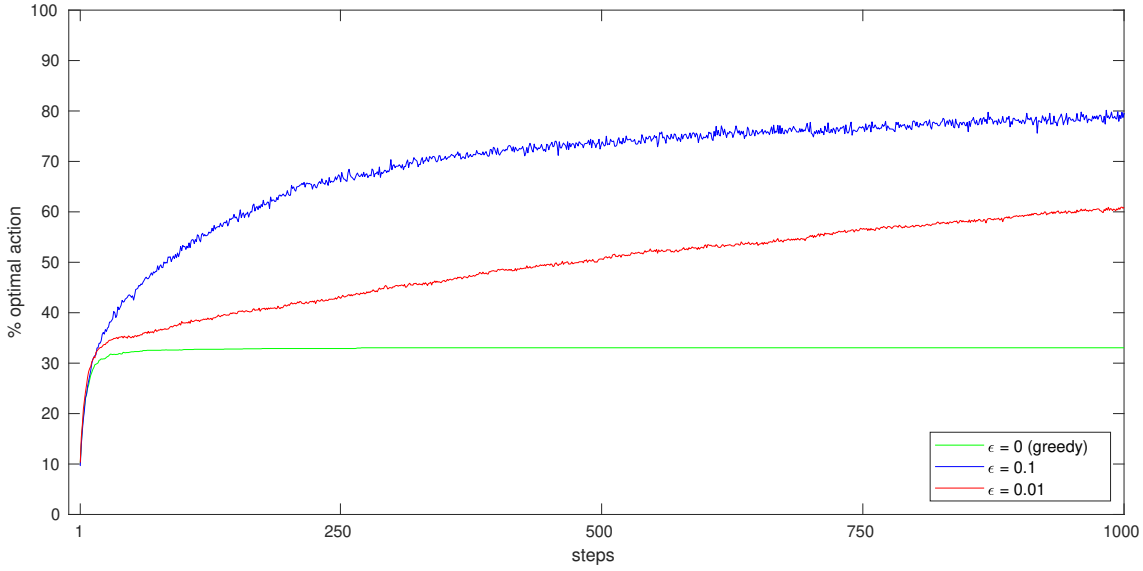

Figure 2: Percentage optimal action comparison of greedy and ϵ -greedy for the 10-arm bandit testbed

1.2 Observations

From the above plot we observe the followings:

(i) Performance of greedy algorithm is worse as it got stuck in suboptimal solution (here it settle around 1, whereas optimal one is approximately 1.55).

(ii) For both ϵ values the results are better as compared to greedy in long run.

(iii) In 1000 steps result for $\epsilon = 0.1$ is better than that of $\epsilon = 0.01$.

2 Softmax Action Selection Method Using Gibbs Distribution

We implement the softmax action selection method using Gibbs distribution for temperature values (T) of $1.5, 1.0, 0.1$ and 0.01, where probability of each action a selection is determined as follows:

$$\mathbf{Pr}\{A_t = a\} = \pi_t(a) = \frac{e^{\frac{Q_t(a)}{T}}}{\sum_{b=1}^{10} e^{\frac{Q_t(b)}{T}}}$$

where $Q_t(a)$ is the estimated action value of action a at time step t.

2.1 Average Performance and Optimality Plots

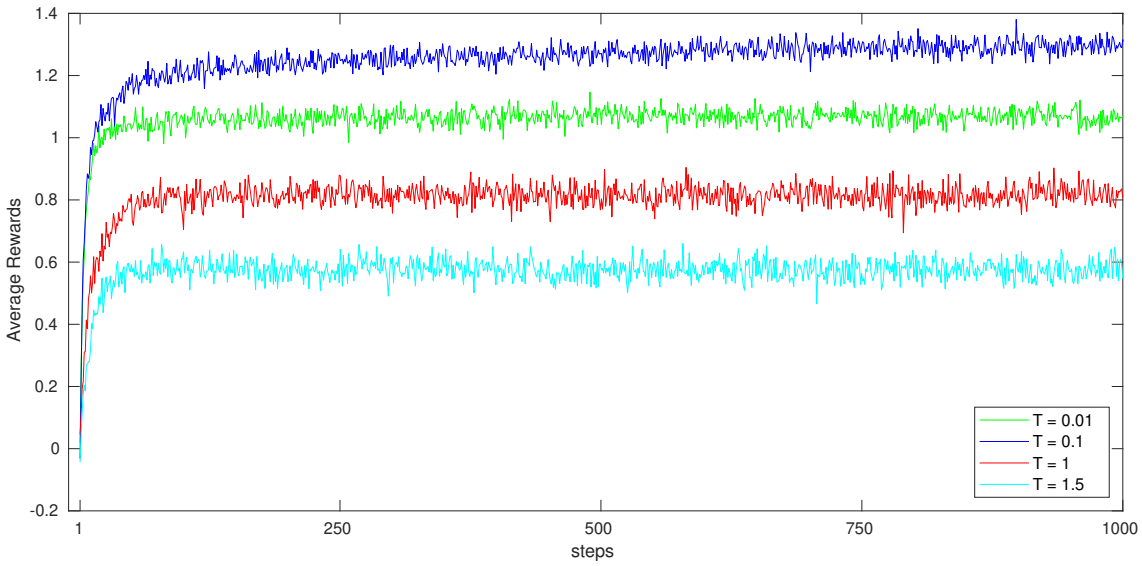

Figure 3: Average reward comparison of softmax for different temperature (T) values

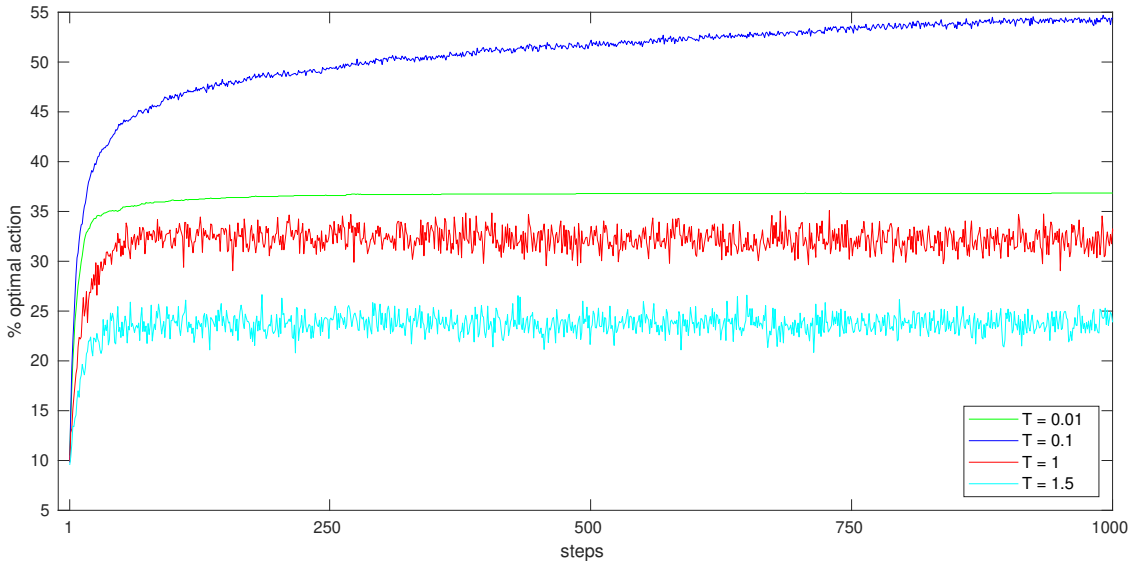

Figure 4: Percentage optimal action comparison of softmax for different temperature (T) values

2.2 Observations

Followings are the important observations from the above plot:

(i) As T is decreased the performance gets better.

(ii) But for very small T softmax become similar to greedy algorithm and thus gets stuck in suboptimal solution (that is why performance for $T = 0.01$ is poor compared to that of $T = 0.1$).

(iii) For large T the action choice become more and more random and the performance degrades and reward approaches the mean of the all reward distribution.

3 UCB1 Algorithm

3.1 Average Performance and Optimality Plots

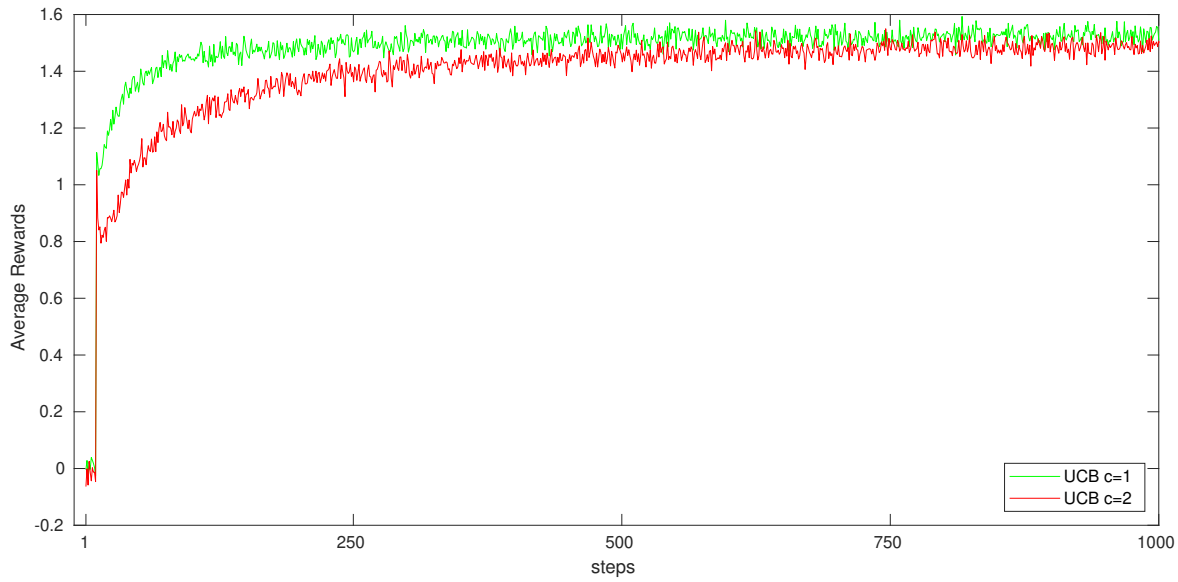

Figure 5: Average reward comparison of UCB1 for different c values

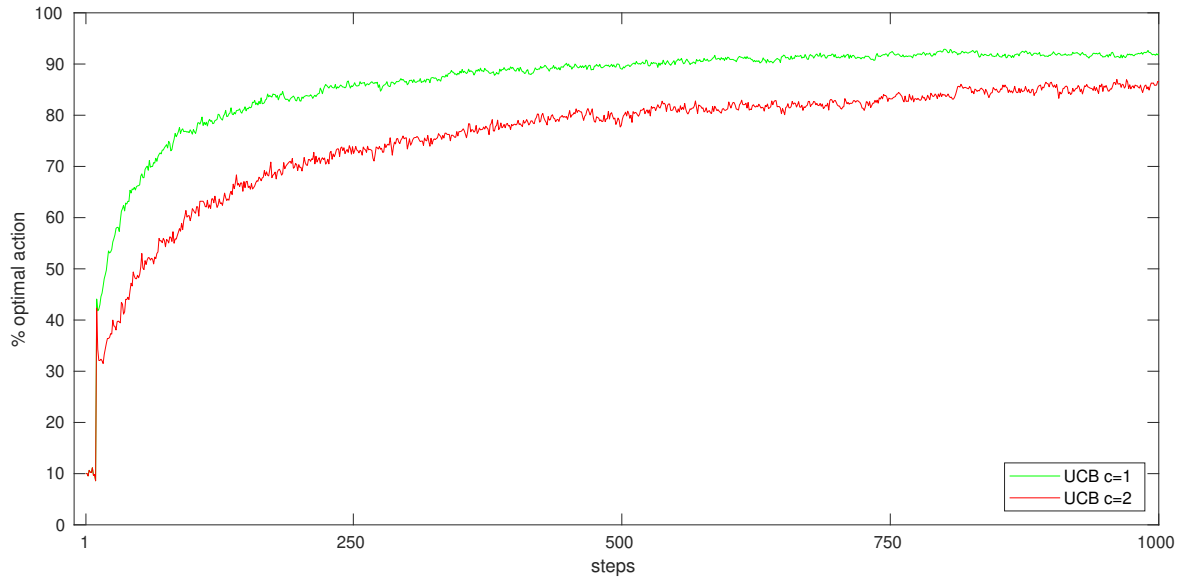

Figure 6: Percentage optimal action comparison of UCB1 for different c values

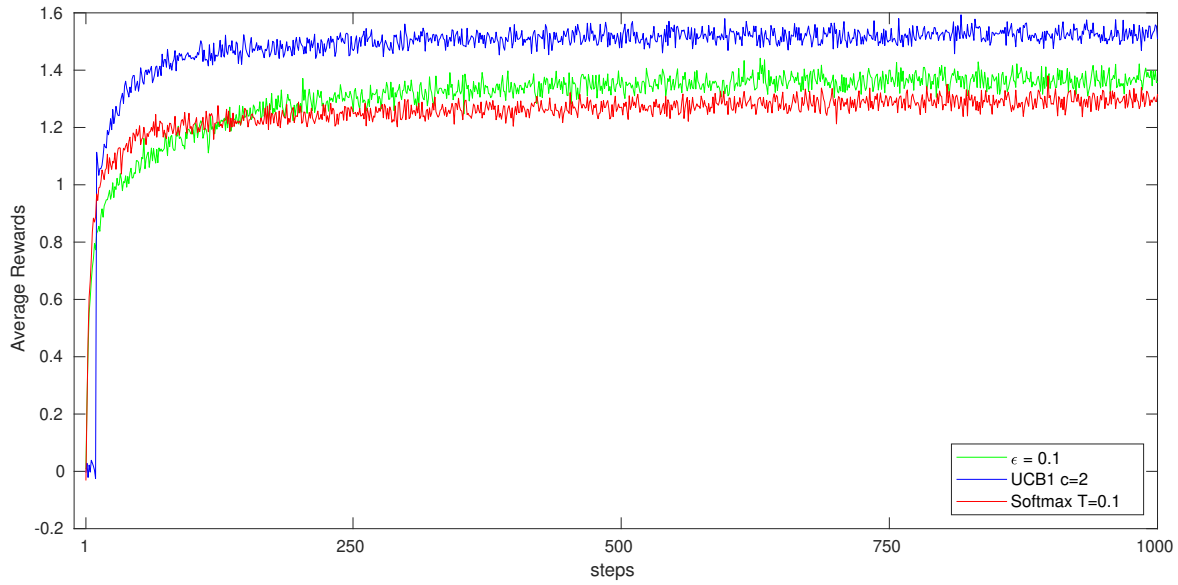

Figure 7: Average reward comparison of ϵ-greedy, softmax and UCB1

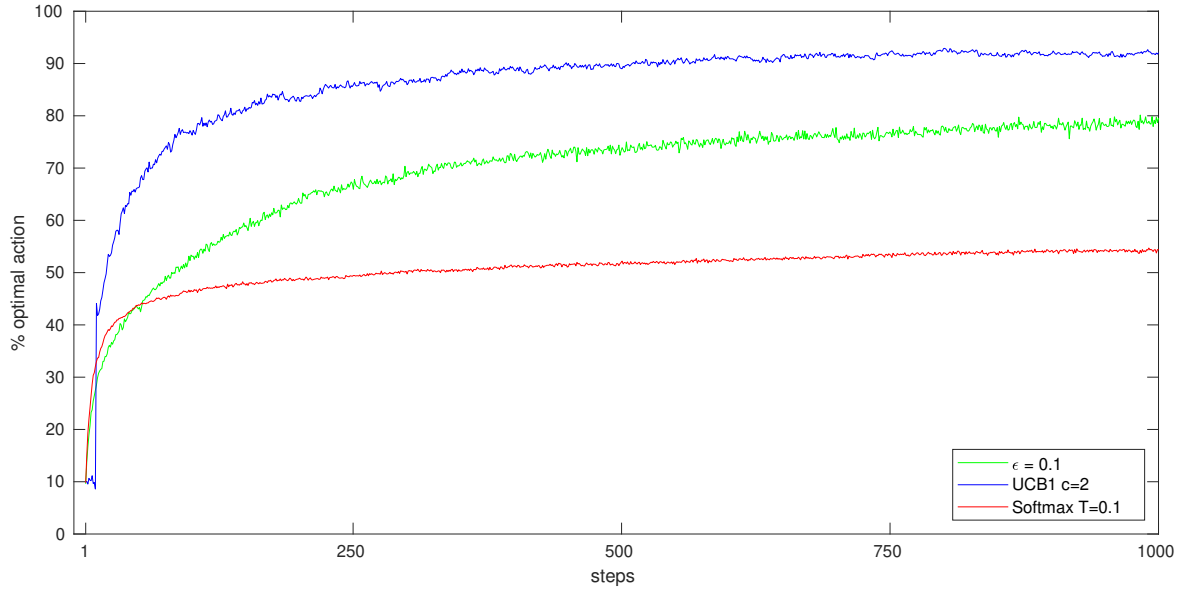

Figure 8: Percentage optimal action comparison of ϵ-greedy, softmax and UCB1

3.2 Observations and Answers to Questions

(i) From the figure (5) we observe that UCB1 algorithm produce a distinct spike at 11th step, which is more prominent for $c = 2$ than $c = 1$.

(ii) UCB1 algorithm tries each arm once in first 10 steps (number of arms is 10 here), so the quantity $Q_t(a) + c\sqrt{\frac{lnt}{N_t(a)}}$ is finite for each arm a and thus in the 11th step the arm having highest estimated $Q_t(a)$ is pulled as the second term $c\sqrt{\frac{lnt}{N_t(a)}}$ is same for each arm. This sudden increase in average reward produces a spike at 11th (total arm+1) step, which decreases after that, as the crude estimates gets better in subsequent steps.

(iii) From the comparison of UCB1 ($c = 2$), softmax ($T = 0.1$) and ϵ-greedy ($\epsilon = 0.1$) in figure (7), we notice average reward performance of UCB1 is better as compared to others.

(iv) Percentage optimal action selection is also higher in case of UCB1 than that of softmax and ϵ-greedy as observed from figure (8).

(v) In UCB1 for exploration, non-greedy actions are selected accordingly to their potential for being optimal (as compared to ϵ-greedy where non-greedy action selection is random), so the average rewards come out to be higher.

4 Median Elimination Algorithm

4.1 Average Performance Plots

The performance of Median Elimination Algorithm (MEA) for different values of ϵ and δ are compared in figure (9) for 20000 time steps (even after getting the optimal arm in lesser number of steps this one arm is pulled for the remaining steps).

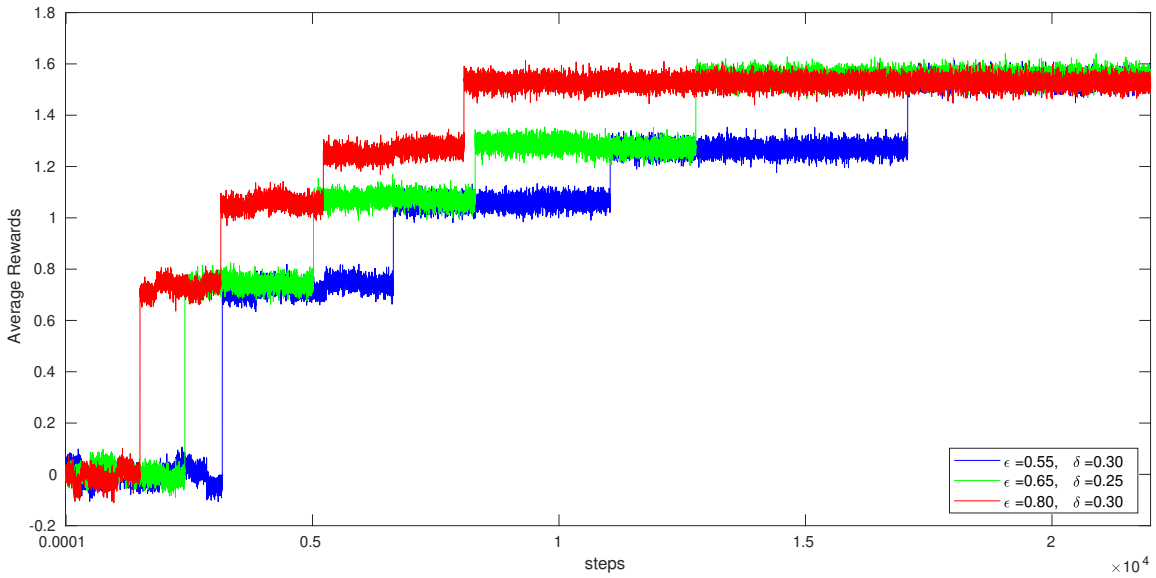

Figure 9: Average reward comparison of Median elimination algorithm for different ϵ and δ values

In the figure (10) we made comparison of average rewards for ϵ-greedy ($\epsilon = 0.1$), softmax ($T = 0.1$), UCB1 ($c = 2$) and Median Elimination Algorithm ($\epsilon = 0.8$ and $\delta = 0.3$). Since MEA takes large numbers of time steps, we fix the time steps (10000 steps) for MEA first and then run each one for that exact number of steps.

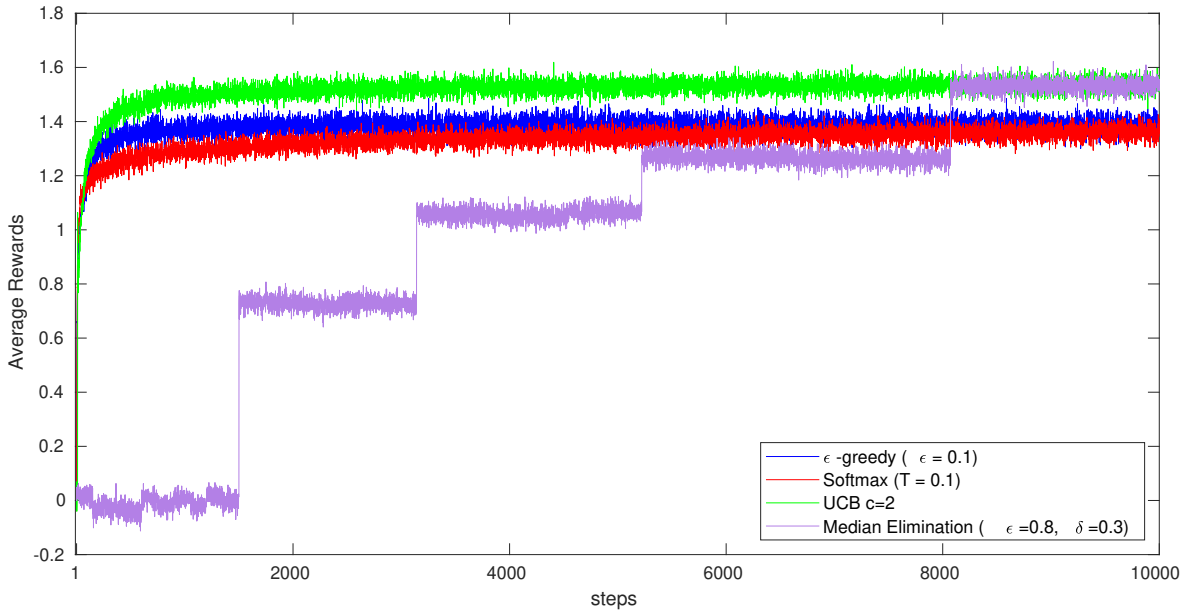

Figure 10: Average reward comparison of ϵ-greedy, softmax, UCB1 and Median elimination algorithm

4.2 Observations

(i) From the figure (9), we observe that as ϵ and δ values are made smaller the number of steps required to find optimal arm increases.

(ii) Performance of UCB1 and MEA are better as compared to ϵ- greedy and softmax as can be seen in figure (10).

(iii) To guarantee PAC optimality MEA takes large number of steps as compared to other algorithms in 10-arm bandit problem.

(iv) Since in MEA after all elimination round, one probably approximately correct optimal arm remains, its average reward is higher as compared to ϵ-greedy and softmax. In the later two algorithms, beside the optimal arm, all arms are also sampled with nonzero(though small) probability, making their average reward smaller.

4.3 Answers to Questions

(i) Computational cost of computing median of any list having n elements using a sorting algorithm is $O(n \log n)$.

(ii) For large n value this median computation can be the rate-determining step.

(iii) Yes, it can be made faster. In fact we can use deterministic median selection algorithm with $O(n)$ to make it faster when the number of arms is high.

(iv) The pseudo-code [1] for deterministic median selection algorithm **select**(L, k), which uses divide and conquer strategy takes a list L and returns kth smaller number (here k is the median position), is given below:

[1] https://en.wikipedia.org/wiki/Median of medians

Algorithm 1 Deterministic median selection algorithm

select(L, k)
partition L into subsets $S[i]$ of five elements each (there will be $n/5$ subsets total).
for $i = 1, 2, \ldots, n/5$ **do**
    $x[i] =$ **select**$(S[i], 3)$
end for
M = **select**$(x[i], n/10)$
partition L into $L1 < M, L2 = M, L3 > M$
if $k <=$ length$(L1)$ **then**
    return **select**$(L1, k)$
else if $k >$ length$(L1)$+length$(L2)$ **then**
    return **select**$(L3, k-$length$(L1)$-length$(L2))$
else
    return M
end if

(v) We can find the computational cost $T(n)$ of deterministic median selection algorithm as follows:

We divided all numbers n into group of 5, so there will be $n/5$ groups. Sorting and finding median of each group (having 5 number each) will be of $O(n)$. We then find the median of the medians ($\frac{n}{5}$ numbers), call it pivot, which have computational time $T(\frac{n}{5})$.

Half of the medians ($n/10$) will be left of the pivot and in each group 3 elements are less than or equal to the median of that group (as each group has 5 element). Thus in total $\frac{3n}{10}$ elements are smaller than the pivot. So in the the worst case, the algorithm may have to recurse on the remaining $\frac{7n}{10}$ having complexity $T(\frac{7n}{10})$.

So, $T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$

Let assume, $T(n) < c * n$ and check if some finite $c > 0$ exist. We also have $O(n) = a * n$ for some $a > 0$

By induction

$$\begin{aligned} T(n) &= T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + a * n \\ &\leq c * \frac{n}{5} + c * \frac{7n}{10} + a * n \\ &= \left(\left(\frac{9}{10}\right) * c + a\right) * n \\ &\leq c * n \end{aligned}$$

where, the last line is true if $((\frac{9}{10}) * c + a) \leq c$, which holds for $c \geq 10 * a$. So the time complexity is $O(n)$.

5 1000-arm Bandit Problem

For 1000 arm bandit problem we choose $\epsilon = 0.8$ and $\delta = 0.3$ for MEA and computed its running steps and run other algorithms same number of times to compare, but now we take the average over 200 bandit problem and used smoothing filter to speed up the running time.

5.1 Average Performance Plots

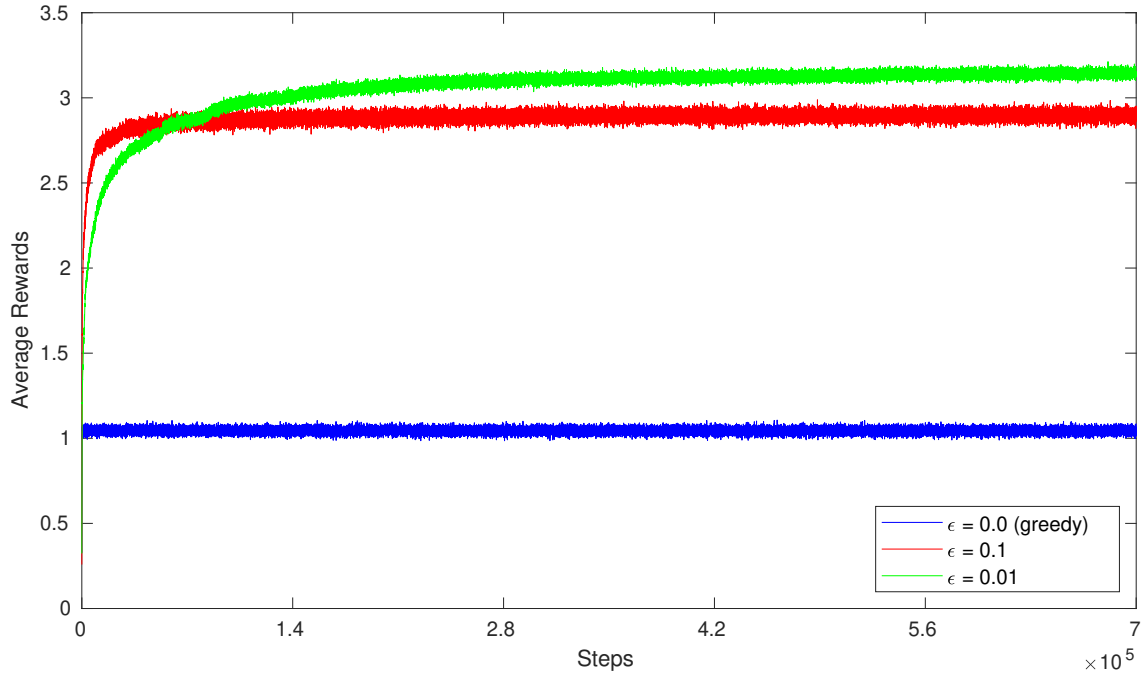

Figure 11: Average reward comparison of greedy and ϵ -greedy for for 1000 arms

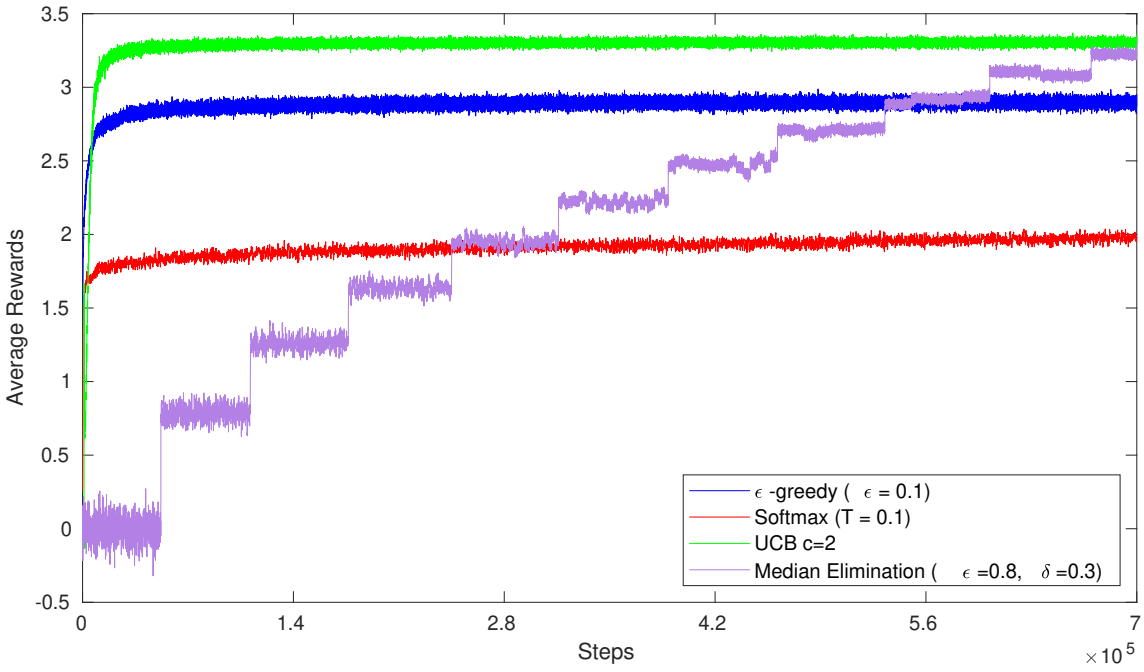

Figure 12: Average reward comparison of ϵ-greedy, softmax, UCB1 and Median elimination algorithm for 1000 arms

5.2 Observations

(i) As the true rewards of the arms are sampled from $\mathcal{N}(0, 1)$, with more and more arm (samples) the probability of having an arm with true rewards more than 3 ($= \mu + 3\sigma$) increases.

(ii) So we observe optimal average reward around 3.35 in case of 1000 arm where for 10 arm it is around 1.55.

(iii) Also as observed from figure (11), greedy average reward is very poor (around 1.1, and ϵ-greedy with $\epsilon = 0.01$ performs better than $\epsilon = 0.1$.

(iv) From comparison figure (12) we find that UCB1 ($c = 2$) and MEA performs very well, whereas performance of softmax (for $T = 0.1$) is poor.