CS 747 - Foundations of Intelligent and Learning Agents
Assignment 1

Gagan Jain
180100043

September 25, 2020

Algorithms

This section is dedicated to making particular remarks about my implementation of all the algorithms.

Epsilon Greedy

This algorithm explores with a small probability ϵ uniformly at random. A list of means of each arm is maintained and updated at every time step. It is initialised with all zeros. At each time step, it generates a random number between 0 and 1, checks if this is greater than ϵ and accordingly exploits the arm with the maximum mean or explores respectively. In case of multiple arms having equal maximum mean, an arm is chosen at random from all the maximal arms. Once this is done, the arm is sampled and according to the reward obtained, the means are updated.

UCB

This algorithm takes into account the means as well as the uncertainity in the mean by keeping track of how many times the arm has been sampled from the start. For the implementation, an array of means and an array of UCBs (means summed up with the corresponding uncertainities terms) are stored and updated. The algorithm starts with round robin sampling to have some initial estimate of the UCBs for each arm. At each time step, the arm with the maximum value of UCB is chosen and sampled and the mean of that arm and the UCB values of all the arms are updated. Again, in case of ties, they are broken randomly.

KL-UCB

This algorithm is similar intuitively to UCB for the fact that the means and the confidence bounds are used to decide which arm is to be sampled next. The only implementational change is the way this is defined. In order to avoid unboundedness during the implementation, two round robins are carried out and then the estimates are calculated by solving an equality which gives us a reward estimate for each of the arm at each time step. Based on that, we sample the arm with the maximal value, breaking ties randomly.

Thompson Sampling

This algorithm relies on a belief of the actual means of the arms, estimated using a Beta distribution over the number of successes and failures for each arm. At each time step, the arm with the maximum belief is sampled and the beliefs are updated according to the reward that it achieves. We start with a round robin to have some notion of belief initially.

Thompson Sampling with Hint

Now we have been provided with the actual means of the arms in sorted order. The benefit of this is that now we know that our beliefs come from a set of discrete values, instead of a continuous distribution from 0 to 1. This streamlines our efforts towards deciding which arm to sample and is thus expected to work better than Thompson

sampling. So, for each arm i, we maintain a array of beliefs depicting its probability of corresponding to the j^{th} arm in the sorted true means array. Initially, we assume each arm to be equally probable in terms of corresponding to any element in the true means array. Assuming the true means array is sorted in ascending order, the last element is the one with the greatest reward probability. Since we wish to find out which of our arms correspond to maximal probability, we take each arm's belief of being the maximal arm and then sample the one which has the maximum belief. Based on the reward we get, we multiply the beliefs of the chosen array by the probability of being the optimal one. If we do not get any reward, we multiply the belief by the probability of not being the optimal arm and thus weaken the belief for that arm. Since we are working with numbers between 0 to 1, we normalise the beliefs after each step so that the beliefs do not diminish.

T1 Plots

We now look at the plots generated for the first four algorithms and make observations.

- The three algorithms ucb, kl-ucb and Thompson-sampling perform much better than the epsilon greedy algorithm for the case of two arms. In case of more arms, the bad performance of the ucb algorithm as compared to epsilon-greedy can be associated to the fact that the ucb algorithm has a logarithmic regret with a coefficient which is very high as compared to the coefficient of time complexity of epsilon-greedy regret which is epsilon-dependant as well and therefore for the chosen value (0.02), it is quite small and thus performs better than the ucb algorithm for certain cases.
- The algorithms Thompson sampling, and kl-ucb have a very small coefficient associated with the dominant term of their time complexity and thus achieve very low regret in all the instances. The regret decreasing and even going negative for some timesteps can be thought of as a dominant random behaviour of the environment, which gave more than the expectation of the optimal arm every arm the optimal arm is pulled. It also depicts the high confidence level of the algorithm on its belief of optimal arm.

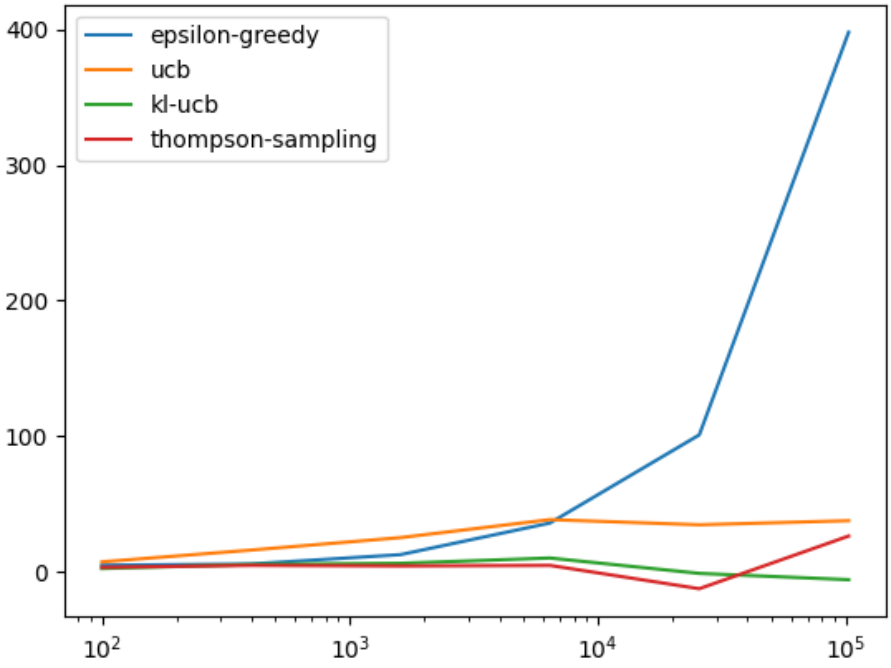

Figure 1: Instance 1

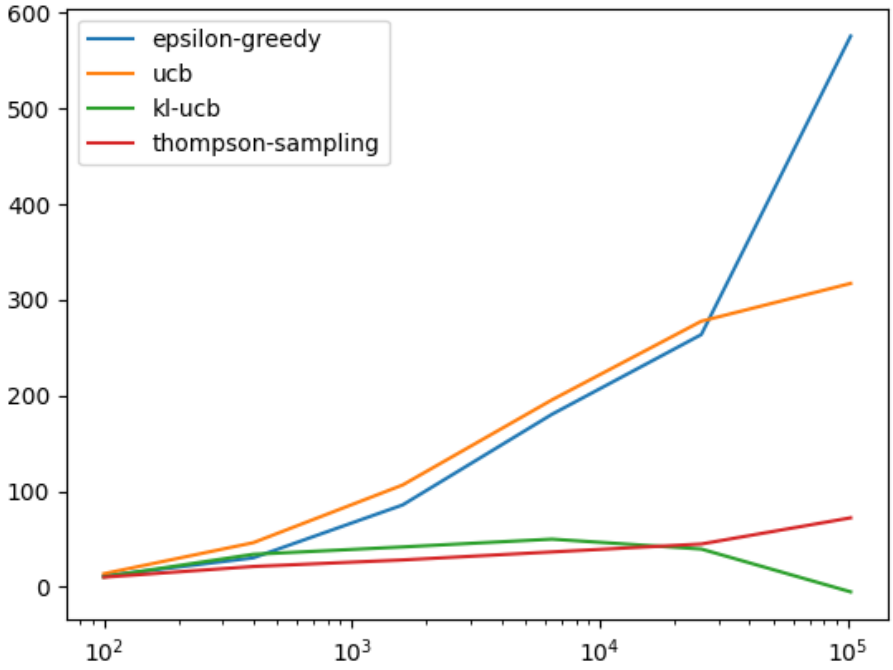

Figure 2: Instance 2

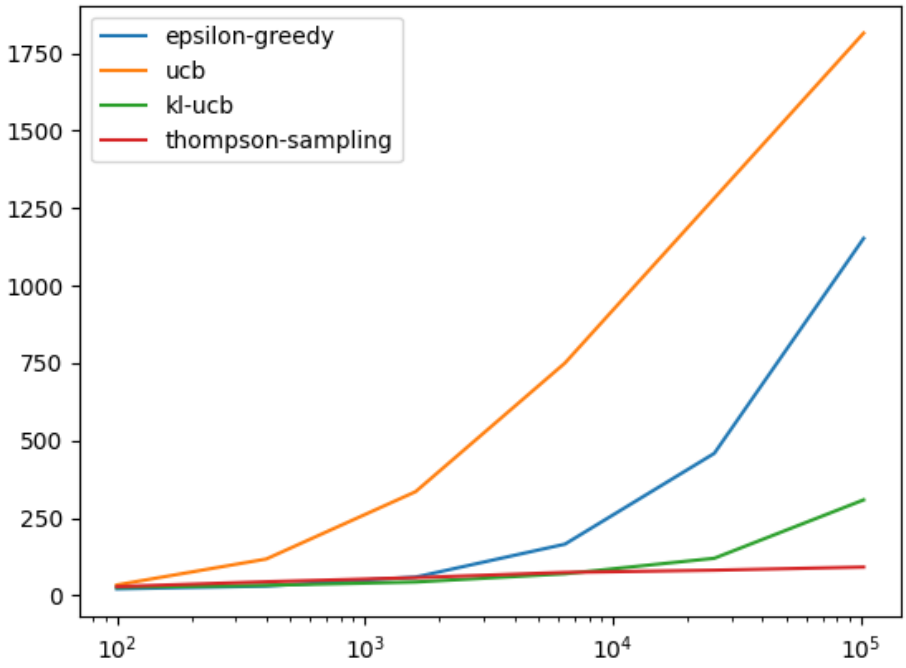

Figure 3: Instance 3

T2 Plots

We now look at the plots generated for the algorithms - *Thompson sampling* and *Thompson sampling with hint* and make observations.

- It is important to note here that the unclear trend of regret when viewed on a small scale is due to the fact that the regret achieved by these algorithms is generally very small. So, the sensitivity to random fluctuations is more and thus one very bad instance is capable of causing a lot of change in the regret averaged over 50 timesteps as well. Also, similar to the case in T1, at some time instances, we have the regret decreasing and even going negative which could be again associated to the randomness and sensitivity notions. For averaging over a greater number of seeds, this should depict a more clear trend.
- The algorithm which had access to the true means of the arms performed much better than the regular Thompson sampling algorithm which samples on a continuous set. Sampling on a finite, discrete set led the algorithm to get more closer to the actual means of the arm and act very optimally.
- There is only one data point where the algorithm without hint performed slightly better (Instance 1, horizon = 25600). But it has to be noted that the results on this particular data point are already randomness driven as the regret is already negative and the optimal arm pulled is giving greater reward than expected. But, in general, for a large enough horizon with averaging over a large number of random seeds, the algorithm with hint is expected to perform better every time going by the results achieved for all the other data points.

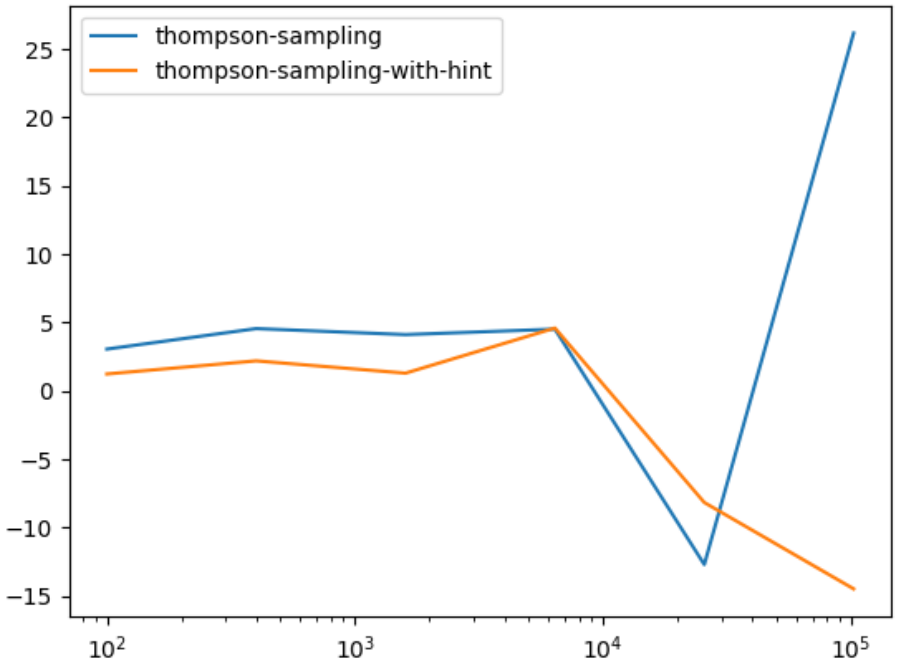

Figure 4: Instance 1

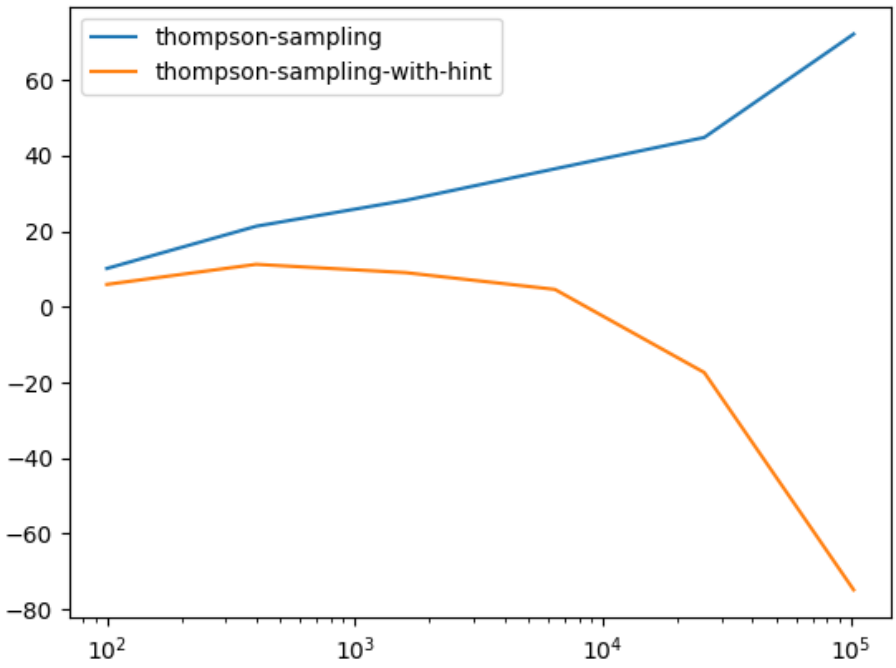

Figure 5: Instance 2

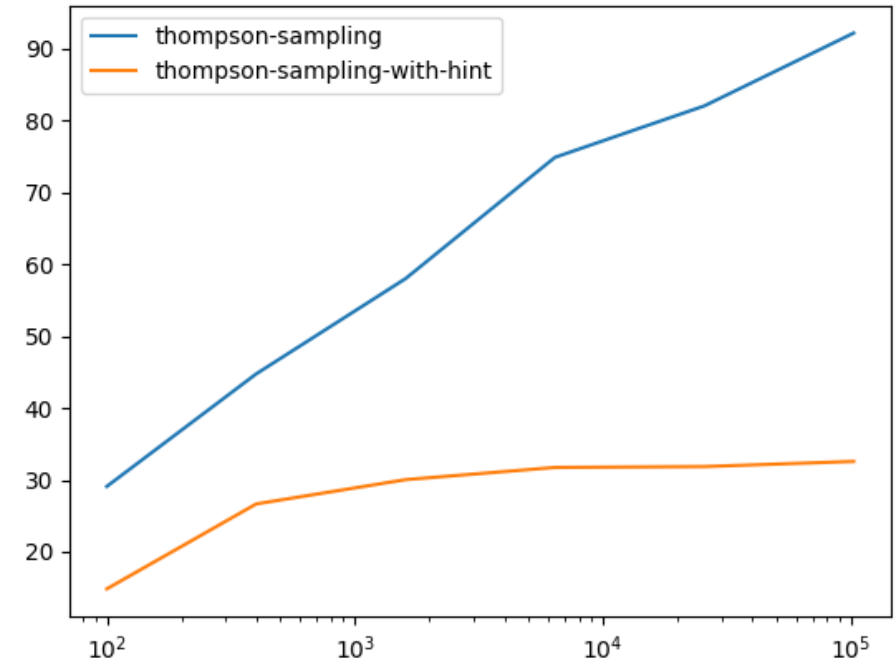

Figure 6: Instance 3

T3 Experiment

We start this section by noting down the values of regret achieved for the three values of epsilon that we chose for every instance.

Epsilon	**0.001**	**0.005**	**0.02**	**0.2**
I1	226.0	**114.82**	398.02	4089.82
I2	2935.04	1052.1	**575.68**	4100.42
I3	2162.76	1253.18	**1153.0**	8518.0

We can observe that the central value of epsilon is giving us a better regret than either of the two other ones. This is something that can be instance dependant as well but here are some general observations which are applicable for every bandit instance. Note that the notion of "how large" and "how small" epsilon can depend on the horizon length and the instance itself, but the intuition is as follows:

- For a very small epsilon value, the epsilon greedy algorithm relies too much on its beliefs of the optimal arm and thus keeps exploiting almost everytime. This might result in a greater regret because it does not explore enough to find out which arm is actually the optimal one and keeps sampling the wrong arm instead. (There might be lucky cases, but very less probable).
- For a very large epsilon value, the algorithm explores too much. Even after having a reasonable guess about the optimal arm, it keeps exploring even in the later time steps, leading to random sampling and greater regret.
- Given these two extreme cases with bad performance, it becomes crucial to tune the parameter epsilon for this algorithm in order to get good reward and low regret on the bandit instance.