CS747: Assignment 1

Shubham Anand Jain, 17D070010

September 23, 2020

1 Comparison of various algorithms - T1

In this section, we compare various algorithms for the multi-armed bandit problem, on three different test sets, and plot them against horizon values. We also briefly explain the implementation of these algorithms.

1.1 Epsilon Greedy

In the implementation of the epsilon-greedy algorithm, we set the initial empirical means of all the arms as 1, and update the empirical mean of the sampled arm at each iteration. This ensures that an arm which hasn't been pulled before is given preference over any sub-optimal arm.

For breaking ties, we take the smallest index with the maximum empirical mean.

1.2 UCB

We set the initial upper bound for each arm as infinity, and update it after it has been sampled atleast once. Thus, in effect we perform a round robin algorithm at the start.

Same as the above, we break ties by taking the smallest index with the maximum empirical mean.

1.3 KL - UCB

In KL-UCB, similar to UCB, we set the initial q value for each arm as infinity. This ensures a round-robin start for the algorithm; when an arm has been sampled atleast once, the actual q-values are computed.

We choose $c = 3$, the "free parameter" in KL-UCB. To compute the q value, a binary search is performed, upto a precision of $2 * 10^-4$. This only takes about $13 - 14$ iterations of computing the KL-divergence, so it is a balance between the speed of results and being very accurate.

Similar to before, ties are broken by taking the smallest index with the maximum q.

1.4 Thompson Sampling

In Thompson sampling, the initial condition is not an issue since our original belief is already that we have an arm with mean $= \frac{1}{2}$. We break ties by taking the smallest index with the largest sampled value.

For getting the beta distribution values, we use numpy.random.beta (all such references have been clearly mentioned in references.txt).

1.5 Results & Analysis

We will analyze the trends in the below graphs according to our theoretical expectations. Our predictions, theoretically, might reasonably be as follows: Epsilon Greedy should have the most regret (since it is not sub-linear), followed by UCB, and KL-UCB and Thompson sampling should have the least regret, matching Lai-Robbin's lower bound.

Of course, this is just a general expectation across bandit instances; for a specific bandit instance, the order could change (the original bounds only talk about worst case!). Thus, it is clear that we may get some unexpected results; however, we will try to motivate this to the best of our ability.

We see that, for instance 1 the trends seem to match our expectation. Thompson sampling seems to have a slight fall in regret between the horizons of 10^3 and 10^4, but this is just an artifact of a small sample size.

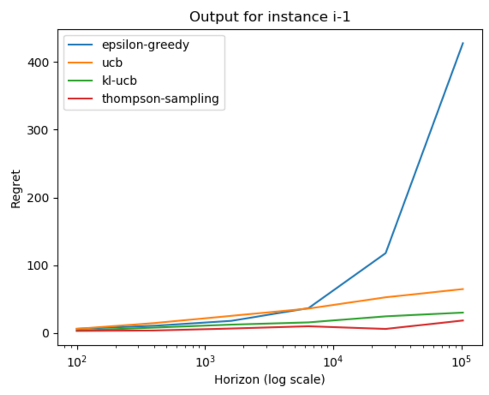

Figure 1: T1, Instance-1 Results

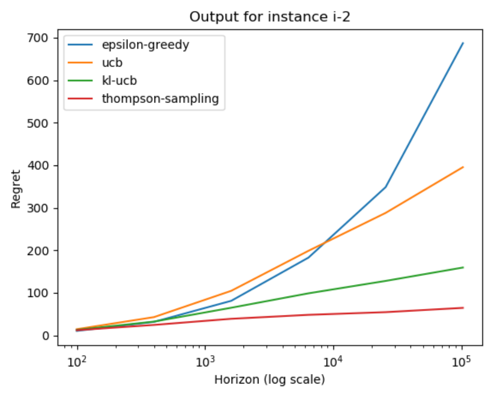

Figure 2: T1, Instance-2 Results

It turns out that instance 2 also seems to follow our expectations. The Epsilon greedy algorithm seems to be changing slope similar to an exponential function, while the other 3 look linear.

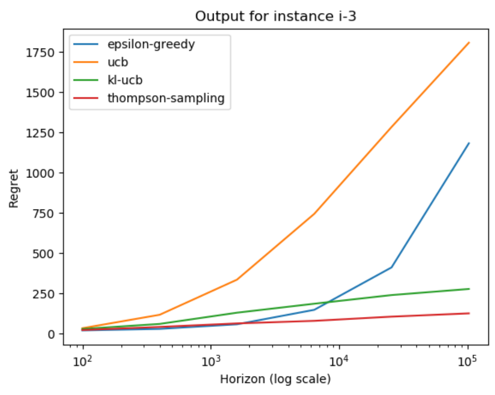

Figure 3: T1, Instance-3 Results

In instance 3, it so happens that UCB performs worse compared to epsilon-greedy for small horizons. However, there are marked increases of slope for epsilon greedy, showing that it is behaving as an exponential question; we can assume that it will overtake UCB at some point. Now, let us try to argue why this happened.

If we look at the arm values of instance 3, it turns out that we have a lot of arms with values close to the optimal. Thus, it seems that UCB pulls all of these arms a high number of times for small horizons; if the difference between the optimal arm and the other arms is very small, the $\sqrt{\frac{logt}{a^t}}$ term tends to make the choice about which arm is pulled. Thus, we expect a lot of sub-optimal pulls at a small horizon in such a case.

On the other hand, epsilon-greedy uses greedy pulls, which tends to work very well if it guesses the arm quickly. Thus, we can expect it to be better than UCB on small horizons, where UCB spends a lot of budget on exploration.

2 Thompson Sampling with Hints - T2

2.1 Motivation behind our algorithm

This section describes the motivation / full procedure we followed to come up with the algorithm described in the next section, and can be skipped without a loss of information.

The hint we are given is the sorted permutation of the original means array, and what we are expected to do is to create a better algorithm based on Thompson sampling, compared to Thompson sampling itself. For this, we analyze what we know about the procedure of this algorithm.

Firstly, we note that the mean of the beta distribution we sample from represents a 'belief' about the true mean of the arm. Thus, what we expect is, knowing some of the arm values, we can somehow do better. Off the top of your head, atleast 3 algorithms come to our mind:

1. Replace the initial valued $\frac{1}{2}$ belief instead with b_{mean}, which is the mean of bernoulli success probabilities of the arms. We expect this to give a better result, since it seems to represent a better belief than the original for most arms. However, this represents a worse belief for the max valued arm if the mean is below $\frac{1}{2}$; and thus, this tends to perform bad in practice.

2. Inspired by the above, replace the initial valued $\frac{1}{2}$ belief with b_{max}; since it represents a better belief for the best arm! However, this tends to perform badly in practice, perhaps because it overestimates the beliefs of the other arms. Surprisingly, choosing b_{min} turns out to work decently well on instances 2 and 3 (but not on instance 1, where it is slightly worse); however, since this seems to be more of a heurestic than a theoretical result, we will not go with this approach.

3. In the above algorithms, we have not used the full extent of information available to us. Considering how to use the full extent of information, we realize that, at any moment if we order the arms by their current empirical mean, then we would expect the true mean of the best current arm to be b_{max}; and so on. Thus, we use the above idea to maintain continuously changing beliefs about true means; at any point, if an arm is the j^{th} largest by empirical mean, we sample from the beta distribution with parameters $\alpha = num_{success}[j] + 2 * b[j], \beta = num_{fail}[j] + 2 - 2 * b[j]$. However, this does not work well in practice; it rewards arms with high empirical mean early on a bit too much.

To remove the deficiencies of algorithm 3, we also considered running normal Thompson sampling for half the horizon, and then running this algorithm. However, that does not seem to help.

Now that we have the intuition that changing the initial beliefs of arms does not seem to work directly, we shift our focus to changing the type of sampling that is occurring. We note that the PDF at a point x in the beta distribution represents the belief that the true mean is x. Thus, we consider discretizing this; we maintain p_{ij}, where p_{ij} is the probability that arm i's true mean is $b[j]$, and $\sum_j p_{ij} = 1$. Then, we sample from this as a discrete distribution, and use that to choose the arm to pull. This idea turns to work out, which we explain in the next section.

Another interesting point is that, we also considered the alternate algorithm: After maintaining these $p'_{ij}s$ such that $\sum_j p_{ij} = 1 \forall i$, we copy these p values to an array r, and normalize r as follows: $\sum_i r_{i1} = 1$, where $b[j]$ is the arm with largest mean. Thus, r_{i1} represents the belief that arm r is the arm with the largest mean. We now sample from this as a discrete distribution. However, it seems this does not work well in practice.

Apart from the final algorithm selected below, we had tried the below algorithm as well:

For the first half of the horizon, we run the naive thompson sampling, while updating the weights of the sampled arm as follows: For each true mean j, if the reward was 1, multiply w_{ij} by $b[j]$, and otherwise by $1 - b[j]$. This directly follows from the PDF of a beta distribution, removing the gamma factors (which are just a constant).

In the second half of the horizon, for each arm, we sample one number - similar to sampling a continuous number in the beta distribution, here we simply sample the discrete means. We sample mean j with probability w_{ij}. Now, across arms we choose the arm which sampled the largest true mean.

This turns out to be better than Thompson sampling, but not by much. However, it gives some motivation to the below algorithm; discrete sampling does not seem to work well in both this and the above cases, hence we try the 'always choosing maximum' analogue of the above algorithm.

Another interesting point: "An Empirical Evaluation of Thompson Sampling" by Chapelle and Li says that better result can be gotten practice at small horizons when you multiply a and b by a constant larger than 1, because of more exploitation and lower exploration. In the above algorithms, this did not work out well, and we have not tested this hypothesis on the final algorithm.

2.2 Algorithm details

Consider the input means to be $b[1], b[2], ...b[n]$, where $n =$ number of arms, $b[1]$ is the largest true mean. We maintain a weight array (say w), where w_{ij} is our belief that arm i has mean $b[j]$, and for all i, $\sum_j w_{ij} = 1$.

At every step, we choose the arm with the highest value of w_{i1}, where w_{ij}'s have already been normalized along arm i. In each iteration, we update the weights of the sampled arm as follows: For each true mean j, if the reward was 1, multiply w_{ij} by $b[j]$, and otherwise by $1 - b[j]$. This directly follows from the PDF of a beta distribution, removing the gamma factors (which are just a constant).

Importantly, after every rewards we also re-normalize the sampled arm's weight function; otherwise, we run into a lot of precision issues at high horizons.

2.3 Results & Analysis

We analyze the results of our algorithm for each of the instances.

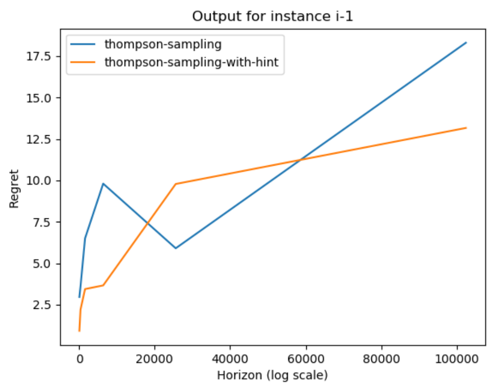

Figure 4: T2, Instance-1 Results

On first look, these results on T1 appear discouraging for our algorithm. However, on a closer look at the data, we realize the following:

- The actual values for a single seed are of the order of $100s$, whereas here the average regret is of the order of $10s$. Thus, the standard deviation for the data is very high - two or three data points in one direction or the other could strongly bias the data.
- 50 seeds is not really a big enough sample size to concentrate such large-variance data around the expectation.

Thus, we performed another experiment with 200 seeds, whose results are below. The results of that experiment show that, at large horizons our algorithm performs clearly better. at small horizons, the difference is barely $1 - 2$ either way; thus, it is hard to conclude anything. Note that the negative regret is again an artifact of the high variance data here.

Next, we look at the data from instance 2. It clearly shows that our algorithm is good at large horizons.

In instances 2 and 3, we can see the trend clearly even at small horizons, which is not true for instance 1. This is because there are a lot of arms with close true means; thus, there are enough "opportunities" for regular Thompson sampling to go wrong. Here, our algorithm clearly shines through as the better alternative. We can also see the gap between the algorithms widening with increasing horizon.

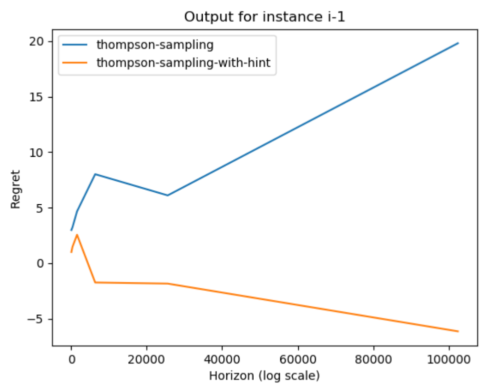

Figure 5: T2, Instance-1 Results; 0-199 seeds averaged

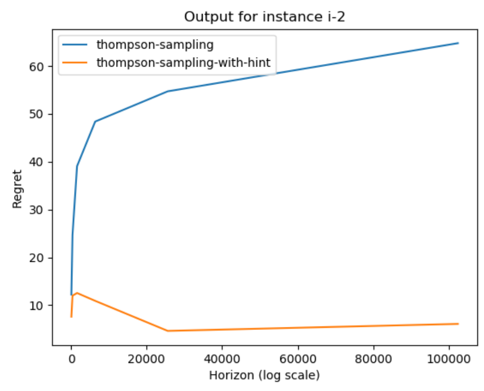

Figure 6: T2, Instance-2 Results

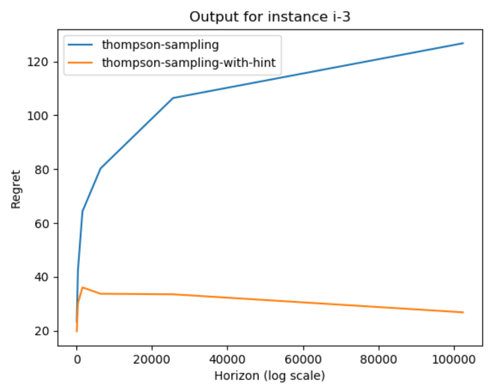

Figure 7: T2, Instance-3 Results

3 Values of Epsilon - T3

Yes, it is possible for all 3 instances to find such ϵ values.

- Instance 1: $\epsilon_1 = 0.004, \epsilon_2 = 0.005, \epsilon_3 = 0.006$
- Instance 2: $\epsilon_1 = 0.005, \epsilon_2 = 0.007, \epsilon_3 = 0.01$
- Instance 3: $\epsilon_1 = 0.005, \epsilon_2 = 0.01, \epsilon_3 = 0.02$