CS 747: Programming Assignment 1

Multi Armed Bandit

Neel Aryan Gupta
180050067

Assumptions

1. Wherever we need to find argmax over a quantity, ties are broken randomly, with each candidate having equal probability of being picked.
2. Both UCB as well as KL-UCB algorithms are initialised with their "scores" as infinity. As a consequence of assumption 1, each arm in both these algorithms will be pulled at least once, in a random order, thereby initializing the algorithms.
3. The choice of constant *c* is fixed to *2.0* for the algorithm *ucb-t1*.
4. The choice of constant *c* is fixed to *3.0* for the algorithm *kl-ucb-t1*.
5. *epsilon-greedy-t1* has been implemented as an epsilon-G3 algorithm, as discussed in lectures.
6. Every point on any of the graphs, be it REGRET or HIGHS-REGRET, is an average done over 50 seeds ranging from 0 to 49 inclusive.

Usage

```
usage: python3 bandit.py [-h] --instance in [--algorithm al] [--randomSeed rs]
[--epsilon ep] [--scale c] [--threshold th] [--horizon hz]

optional arguments:
  -h, --help        show this help message and exit
  --instance in     Path to the instance file (default: None)
  --algorithm al    Algorithm to use (default: epsilon-greedy-t1)
  --randomSeed rs   Number to set as seed for RNG (default: 42)
  --epsilon ep      Epsilon for epsilon greedy (default: 0.02)
  --scale c         Scale factor for exploration bonus of UCB (default: 2)
  --threshold th    Threshold for Task 4 (default: 0)
  --horizon hz      Total number of time steps (default: 10000)
```

Task 1

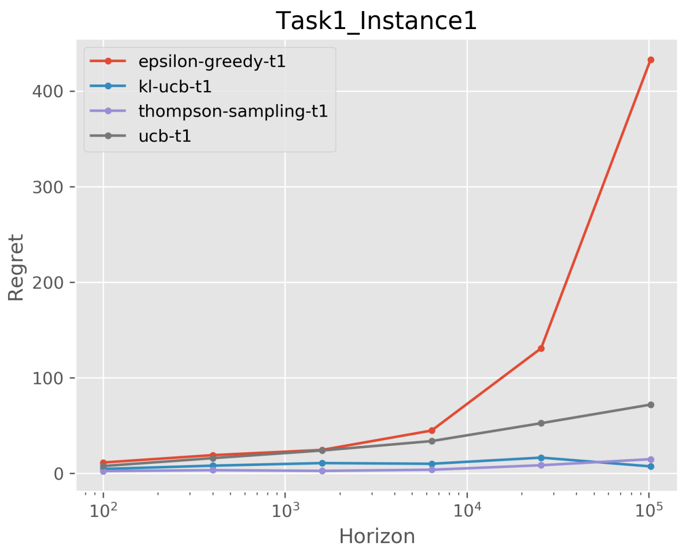
In this task, there are three bandit instances, consisting of 2, 5, and 25 arms respectively. This task studies the effect of the number of arms on each of the four algorithms.

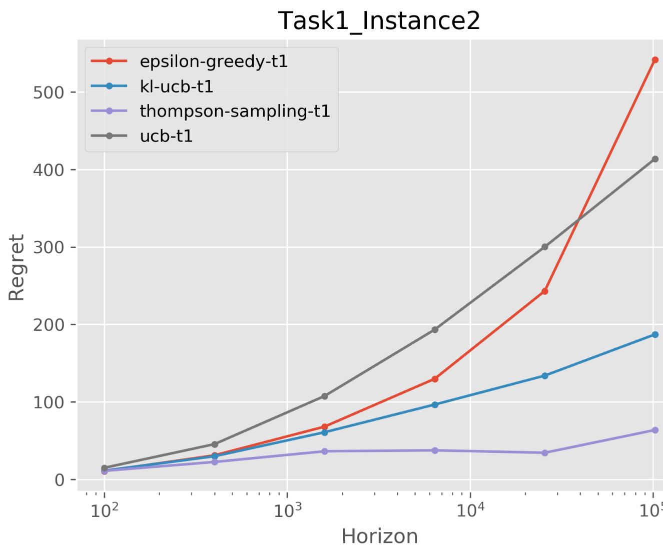
Observations:

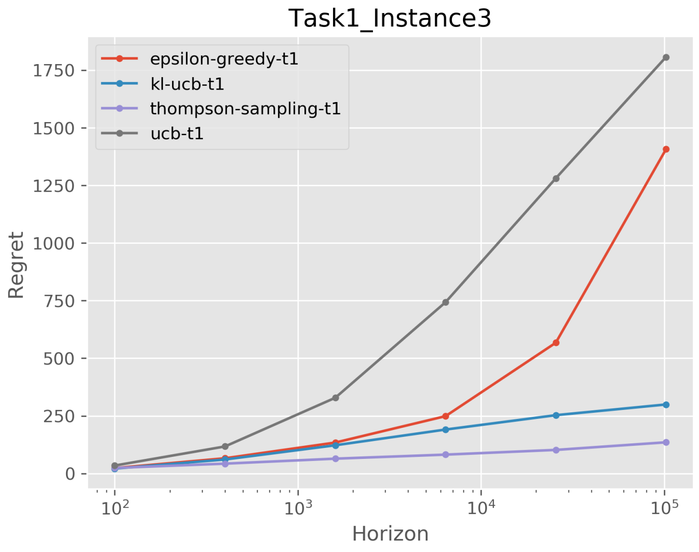
1. In each of the three graphs, all algorithms except *epsilon-greedy-t1* start off with a slightly exponential curvature, and eventually take on roughly a linear scale (logarithmic to actual horizon) with respect to the

log scale horizon. The *epsilon-greedy-t1* algorithm clearly follows the trend of that of an exponential curve, suggesting that the regret is linear with respect to the horizon scale.

2. The regret values for instance 2 are higher than all other instances, because of the fact that the true probabilities associated with each arm in instance are in closer proximities as compared to the other instances. As a result, all the algorithms struggle to judge the optimality of the arm (on a relative scale).
3. The *ucb-t1* algorithm does achieve sublinear regret, however it is not the optimal algorithm, which is observable from the graphs and is consistent with theory as well.
4. The *ucb-t1* algorithm eventually catches up with *epsilon-greedy-t1* algorithm, and will cross the same in instance 3 on extrapolation. However this trend of "catching up" of the *ucb-t1* algorithm shows that this algorithm takes longer time to explore before actually moving to the exploiting phase.
5. The rough comparison of the algorithms follow the order - *epsilon-greedy-t1*, *ucb-t1, kl-ucb-t1*, *thompson-samping-t1* - in the order of decreasing regret.





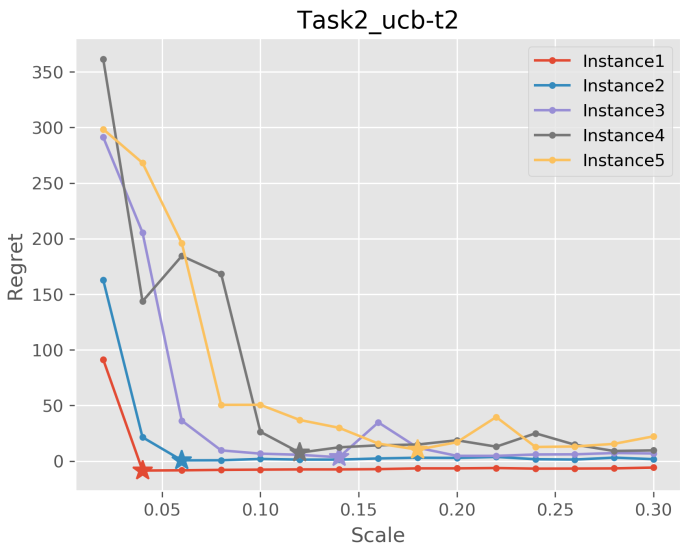


Task 2

In this task, there are five bandit instances for each of which we need to tune the scale parameter of the UCB algorithm i.e. analyse the effect of exploration on UCB algorithm's performance.

Observations:

1. Barring a few outliers, all the lines follow a decreasing trend with increasing scale implying that the algorithm needs to explore upto a certain point to appropriately judge an exploitable arm.
2. Beyond a certain scale value, the regret becomes constant as the algorithm's exploitation overpowers its exploration because of its correct choice of an arm.
3. The optimal values for c for each instance are marked as a "star" in the plot and their values are as follows - 0.04, 0.06, 0.14, 0.12, 0.18 for the instances 1 through 5 respectively.



Task 3

In this task, the generalization of bernoulli as a finite support probability distribution is introduced. Observing from the previous tasks, Thompson Sampling algorithm achieves the lowest regret consistently, and moreover is a theoretically optimal algorithm. Therefore, it is only natural to try to adapt this algorithm to a finite support system. The intuition behind the design of adapted Thompson sampling comes from its derivation using the Bayes Rule, done in the lectures.

$$Belief_{m+1}(w) = \frac{Belief_m(w).P(e_{m+1}|w)}{\sum_{w' \in W} Belief_m(w').P(e_{m+1}|w')}$$

Note, we replaced the $P(e_{m+1}| w)$ term with x, when the reward was 1 and with $1 - x$ when the reward was 0. Adapting this to the finite support system, we will now multiply by p_i whenever we get the ith reward from the support, where p_i is the probability of the i^{th} support. Now the iterating over various time steps we will get the updated belief as follows:

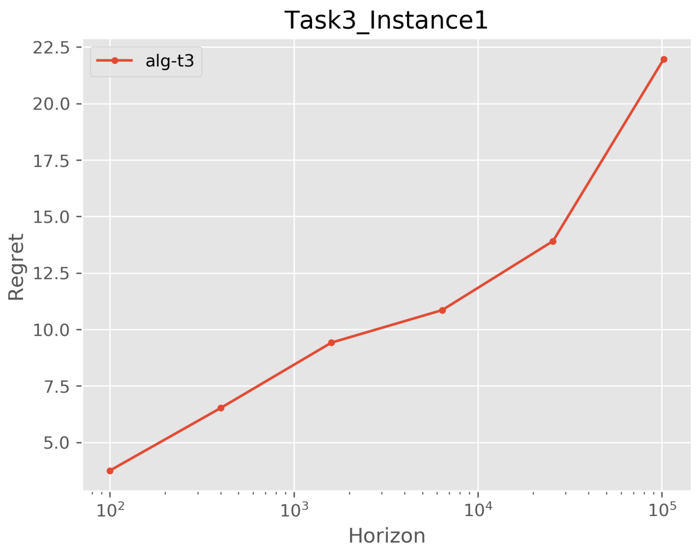
$$Belief_m(x) = \frac{\prod^k p_i^{s_i}}{N} = Dirichlet(s_1 + 1, s_2 + 1, \ldots, s_k + 1)$$

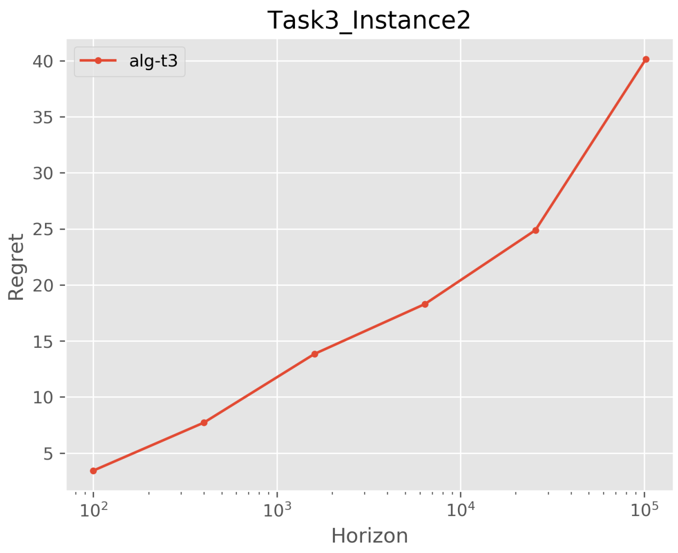
$$where \sum^{k} p_i = 1$$

Note that this has now transformed from Beta distribution to a multivariate Beta distribution with parameters $\widehat{p}$, also known as the Dirichlet distribution. Therefore we now sample k-dimensional samples from the Dirichlet distribution for each arm, take their dot product (or weighted average) with the supports, and pull the arm which has the maximum such dot product.

Observations:

1. This algorithm achieves very low regrets and is also linear with respect to the log scale horizon.
2. The values of regrets are slightly higher for instance 2 as compared to 1, because one of the arms in instance 2 has a low expected cumulative mean reward of 0.39 which harms the algorithm during its exploration phase.





Task 4

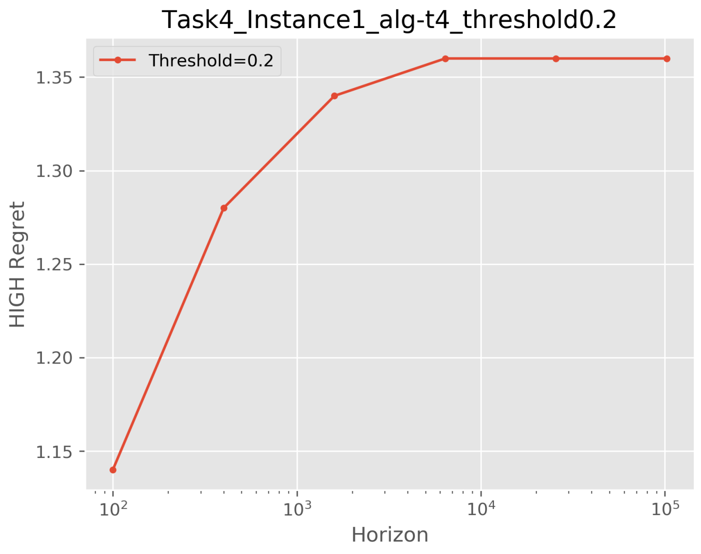
This task involves the maximization of thresholded rewards (called HIGHS) from a finite support bandit. This problem can be easily morphed into that of a bernoulli bandit. For all intents and purposes, the algorithms can assume that the actual probability associated with each arm for sampling a HIGH reward is a bernoulli parameter p which is the sum of the probabilities for which the support is greater than the threshold, for each arm. Therefore, Thompson Sampling algorithm can be applied as is, in this case, where we just change the definition of success i.e. s_1 which is incremented only if the reward received from the bandit is greater than the threshold and is otherwise, counter as a failure, and finally we sample from the Beta distribution with the parameters $(s_i + 1, f_i + 1)$ as per the above definition.

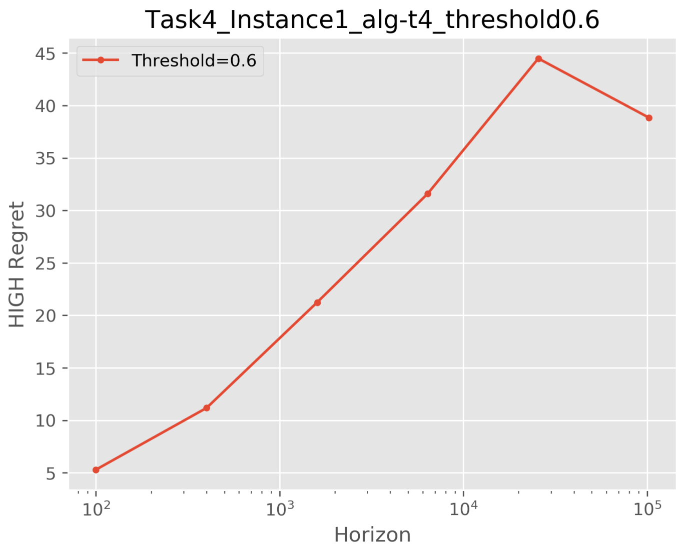
Observations:

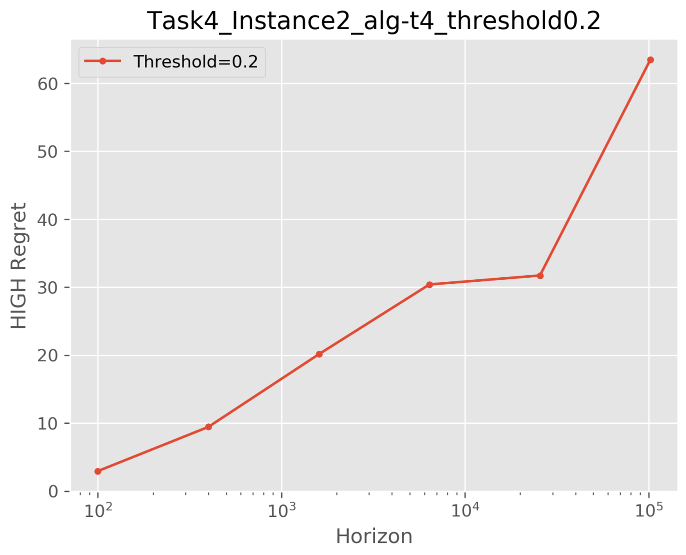
1. Instance 1 for threshold of 0.2, has two arms (out of three) which will output a HIGH reward with probability 1.0, making the overall regret of the algorithm lower as well. The algorithm quickly picks up on

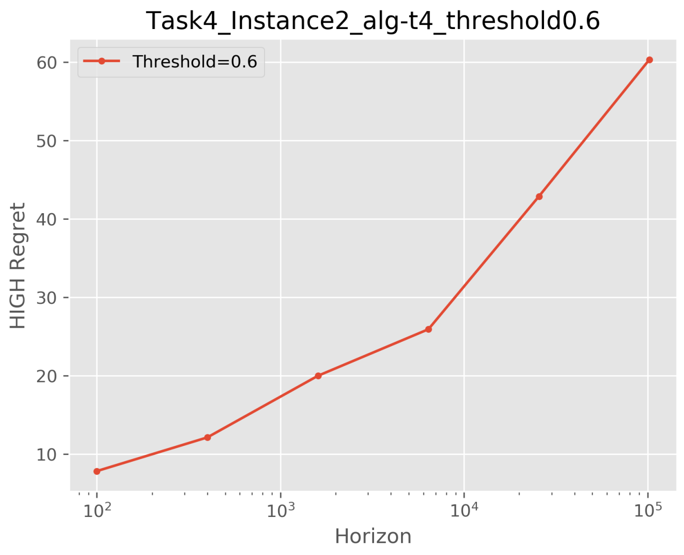
the optimal arm and incurs at most a penalty of 2, even for a horizon as large as 102400. This is the only case where the graph isn't linear and converges very quickly to the optimal regret.

2. For all cases except the one mentioned in point 1, each graph is a straight, roughly linear, line, implying that the algorithm achieves asymptotic logarithmic regret with respect to horizon.
3. For instance 2, the algorithm achieves comparable regret because of a tradeoff between increased threshold (lower HIGH probabilities) and the closely spaced probabilities. For threshold 0.2, the HIGH probabilities (0.85, 0.9, 0.81) are in close proximity and make it harder for the algorithm to judge the optimality of any arm, whereas for threshold 0.6, the HIGH probabilities (0.3, 0.53, 0.25) show a clear optimal arm which is easier to distinguish as compared to the aforementioned case.


Task4_Instance1_alg-t4_threshold0.2
Threshold=0.2
HIGH Regret
Horizon
10^2
10^3
10^4
10^5


Task4_Instance1_alg-t4_threshold0.6
Threshold=0.6
HIGH Regret
Horizon
10^2
10^3
10^4
10^5


Task4_Instance2_alg-t4_threshold0.2
Threshold=0.2
HIGH Regret
Horizon
10^2
10^3
10^4
10^5


Task4_Instance2_alg-t4_threshold0.6
Threshold=0.6
HIGH Regret
Horizon
10^2
10^3
10^4
10^5