Assignment-1 Report

- G.M.Ritesh Kumar
- 160070048
- 03/09/2019

IMPLEMENTATION:

I have used python3 for implementation of the bandit algorithms
Common Assumption: Tiebreaks are resolved by selecting an arm with min index everywhere

Round-robin:
Implemented using a for loop

E-greedy:
For a given e value I have generated a Bernoulli with p as e (binomial with n,p = 1,e)
If e is 1 then pull a random arm (np.choice)
else: pull the arm with max mean
Assumption**:** For the mean, if an arm isn't pulled yet then I have assigned it's mean to be 1

UCB:
Round-robin for starting n rounds where n is the number of arms and
Ucb from n to the horizon

KL-UCB:
Round-robin for starting n rounds where n is the number of arms and
KL-Ucb from n to the horizon

For calculation fo the KL-UCB value I have done this
Let, KL = KL(p||q) = pln(p/q) + (1-p)ln((1-p)/(1-q))

Note that KL divergence is convex and solving KL’ = 0 gives p as the minima
We need to find the max value of q in [p,1] for which the below inequality satisfies

KL < ln(t) + 3*ln(ln(t)) / u
KL< C

Can be converted to KL - C <=0
As the minima is p => KL is increasing from p to 1 and the value of KL at 1 is +infinity

So, if the KL at p is less than C then the solution would be the value of q which satisfies

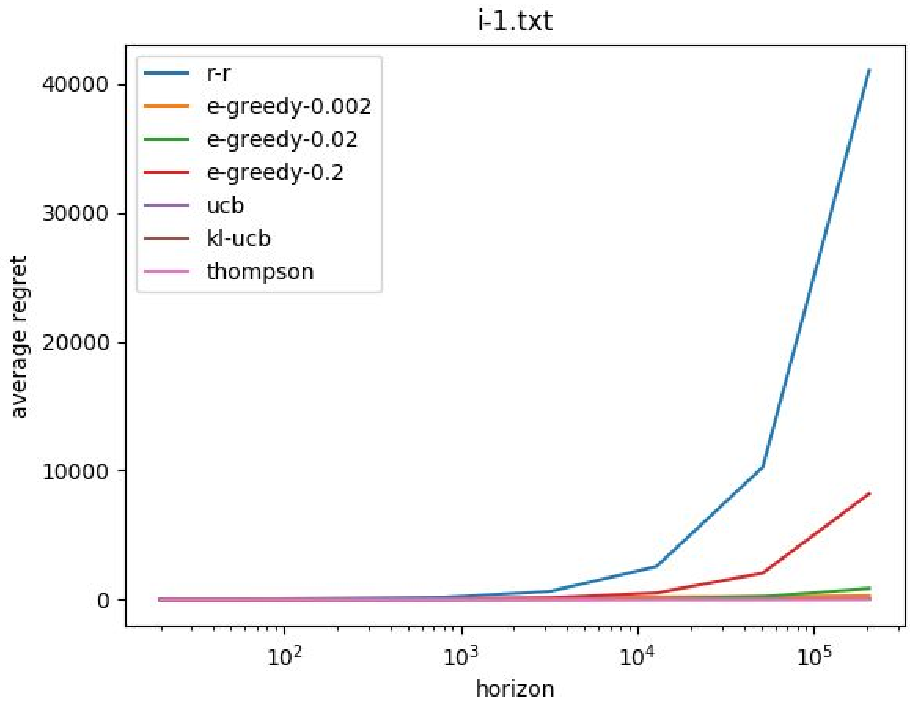
KL-C = 0 (Since it’s increasing from p to 1), I have used scipy.optimize.bisect method to solve this

Thompson:
I have used the numpy.random.beta function for sampling

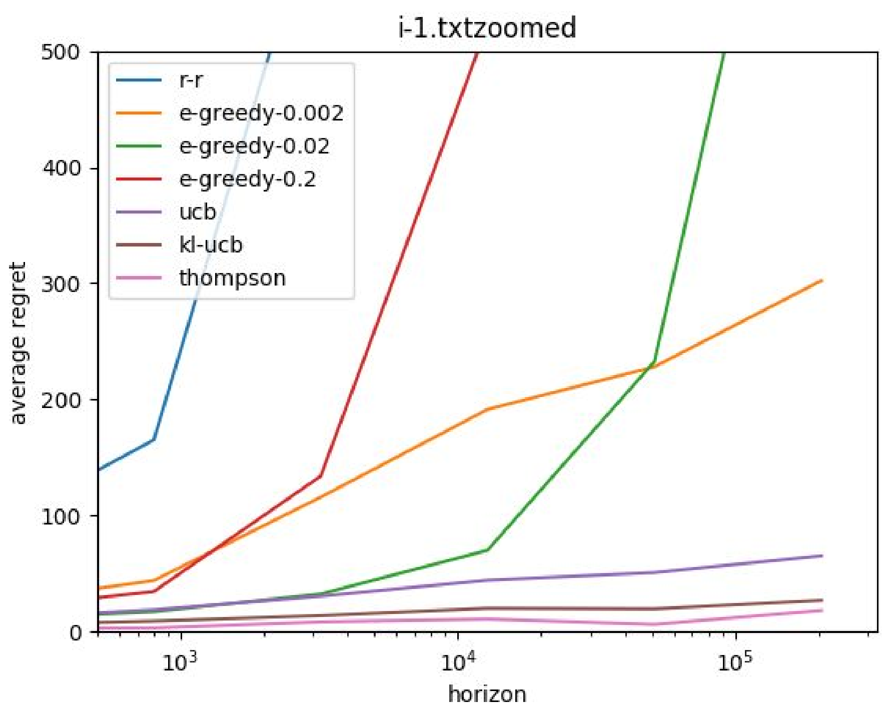
PLOTS:

For instance 1:

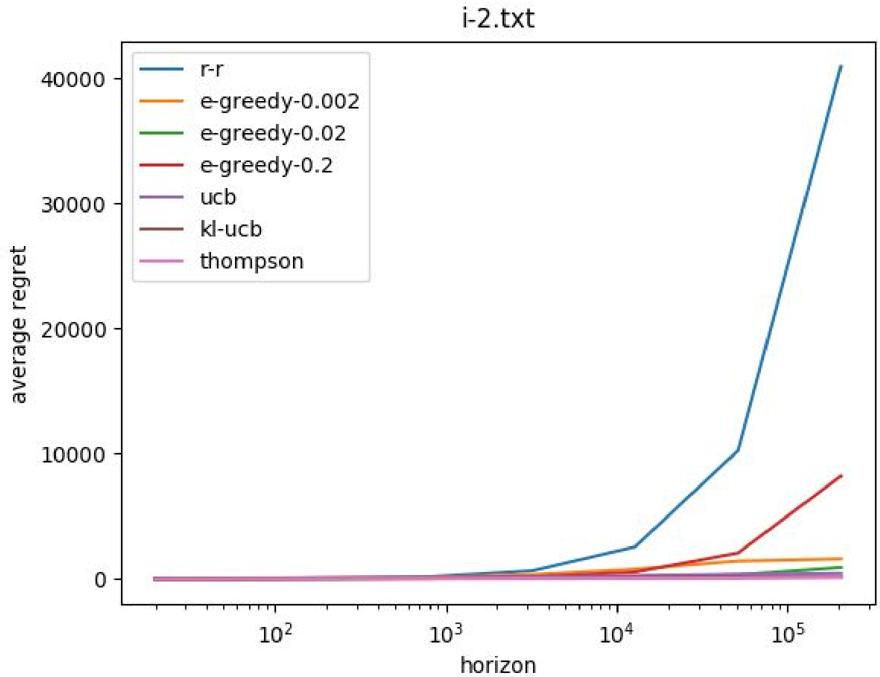
Plot till horizon is



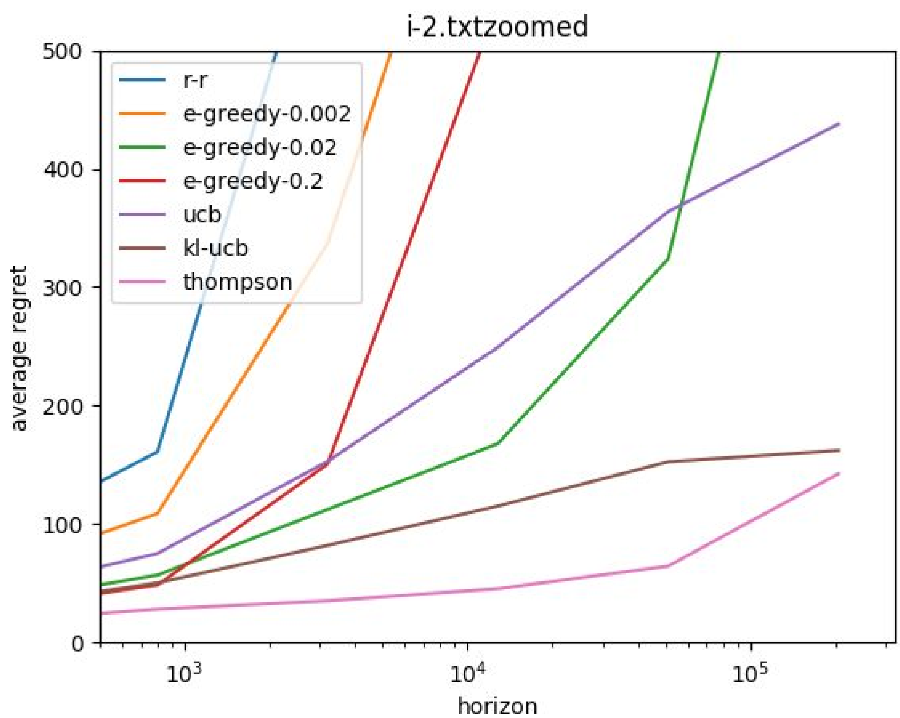
Zoomed view of the above plot is here:



For instance 2:
plot till horizon is

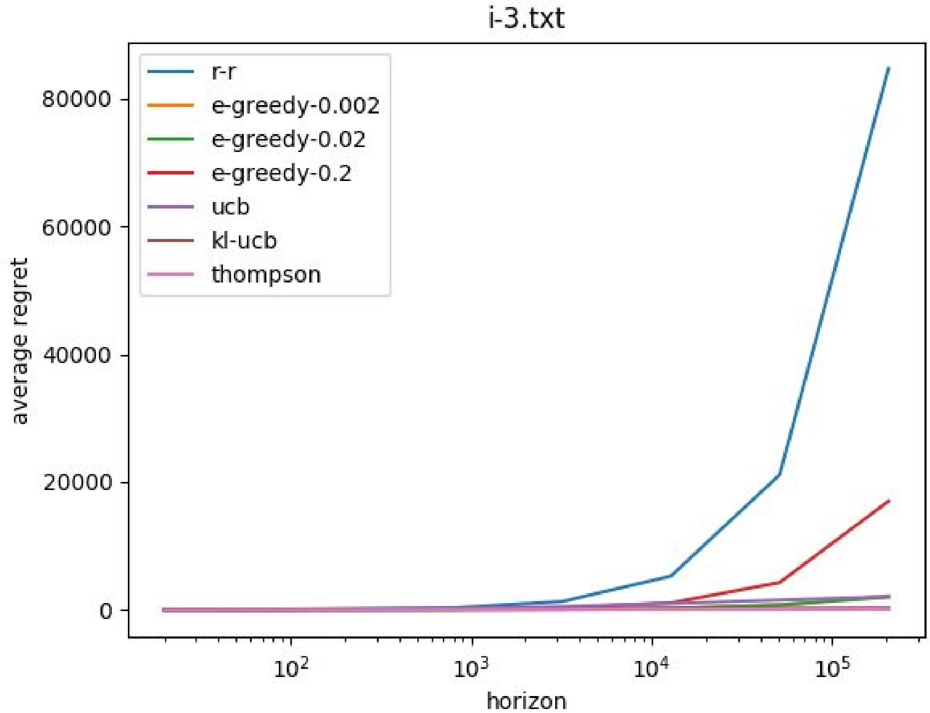


Zoomed view of the above plot is

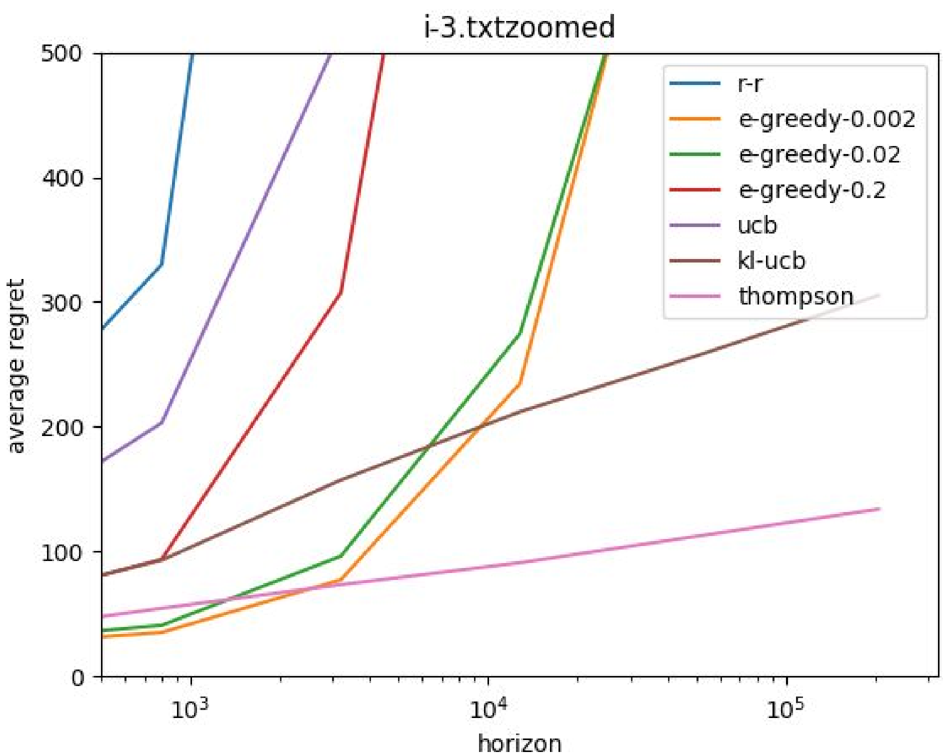


For instance 3

Plot till the horizon is



Zoomed view of the above plot is



OBSERVATIONS:

- For all the instances round-robin has the highest regret So, it's performance is the worst of all
- As we can see the increase in Round-robin is exponential (hence, it has exponential regret)
- Performance of epsilon greedy in case of 1st instance is eg-0.002 better than 0.02 better than 0.2. This is because as we have only 2 bandits, in this case, we don't need much exploration so, eg-0.002 is the best among all.
- 0.2 epsilon greedy is not good for all the instances this is because of the higher value of epsilon which leads to over exploration
- For instance 2,3 we can see that the performance of 0.02 is better than 0.002 this is because 0.002 is a lower value and it is like no exploration at all in case of higher number bandits
- In all the cases we get the regret of UCB > KL-UCB > Thompson sampling which is expected and if we notice clearly, these regrets are linear, since, we have plotted the results on a logarithmic scale so, the regrets are logarithmic.