CS 747 ASSIGNMENT – 1 REPORT

Samarjeet Sahoo
150100017
4th Year
CSE B.Tech

Questions
What patterns do they display?
Do your results validate your intuition about the effect of epsilon?
Did you notice anything unexpected?
Do you have any ideas to further improve performance?
The report should mention what changes you made (if any) to account for non-bernoulli rewards.
It should also include other relevant details about your implementation of the algorithms (for example, if you tuned any parameters).

The Patterns displayed:

The general observations which hold in most cases are:
1. For larger values of epsilon(e.g. 0.5,0.4), the epsilon greedy has the maximum value of expected cumulative regret. Apart from epsilon greedy, we have the following in the decreasing order of expected cumulative regret- UCB, KL-UCB, Thompson Sampling.
2. The least regret is achieved by thompson sampling or by epsilon greedy with epsilon=0.1 always.
3. The UCB, KL-UCB and Thompson Sampling curves have a gradually decreasing slope of the curve. That is, there is a decreasing rate of increase in cumulative regret. For epsilon-greedy, the curves are nearly linear.
4. For smaller values of epsilon, we see that epsilon-greedy performs really well, even better than Thompson, but eventually, Thompson sampling will overtake epsilon greedy algorithm, because of their rate of change of cumulative regret values.
5. For instance histogram, the rate of slope change is lesser for UCB, KL-UCB, Thompson Sampling. So, the curves look almost linear for horizon=1000, but in horizon=10,000, we can see the curves of the aforementioned 3 algorithms to have a decreasing rate of increase

Effect of epsilon:
In all cases, the expected cumulative regret is higher for higher values of epsilon. We see, epsilon(0.5) >epsilon(0.4)>epsilon(0.3)>epsilon(0.2)>epsilon(0.1) in terms of expected cumulative regret. This is intuitive, because the agents with higher values of epsilon tend to explore more and hence accumulate more regret. The ones with

lower epsilon exploit more and hence accumulate lesser regret because they pull the best arm more.

Unexpected Observations:
As far as I could observe there were no unexpected observations.

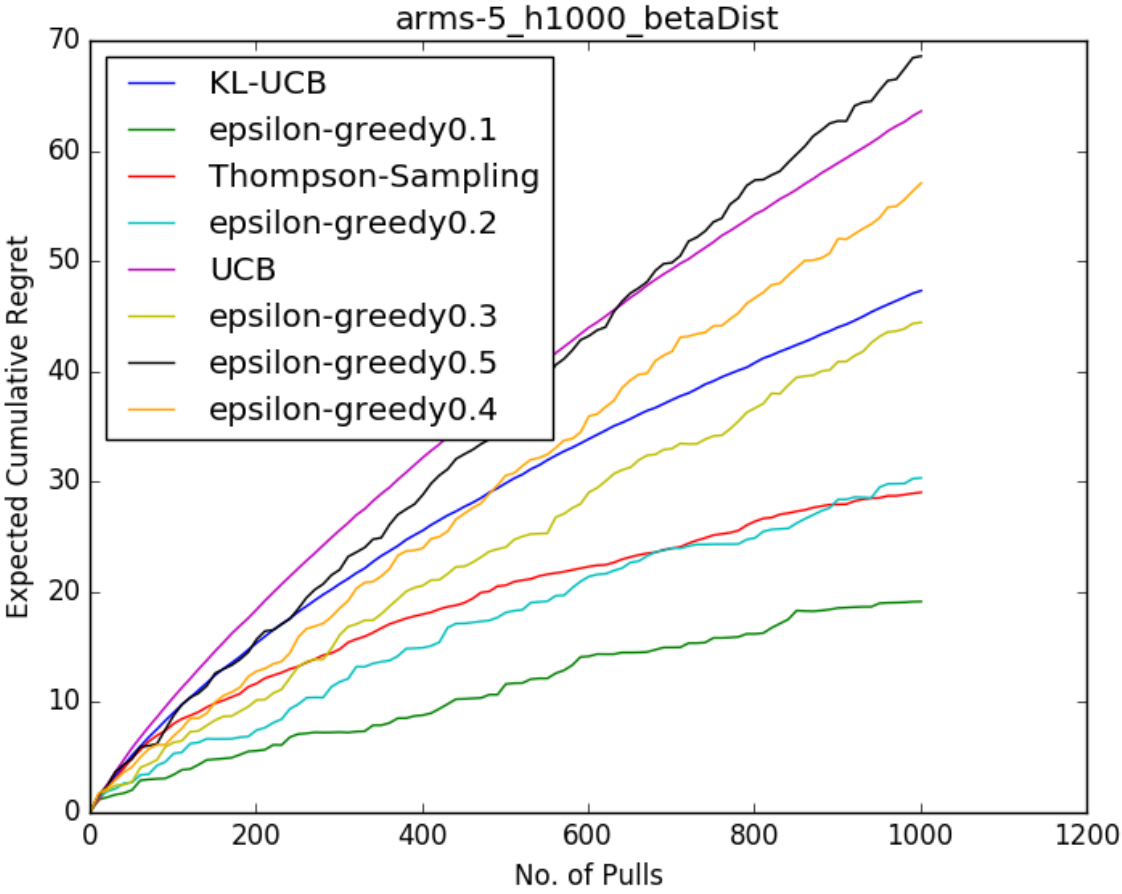
Ideas to improve performance:
None that I could think of.

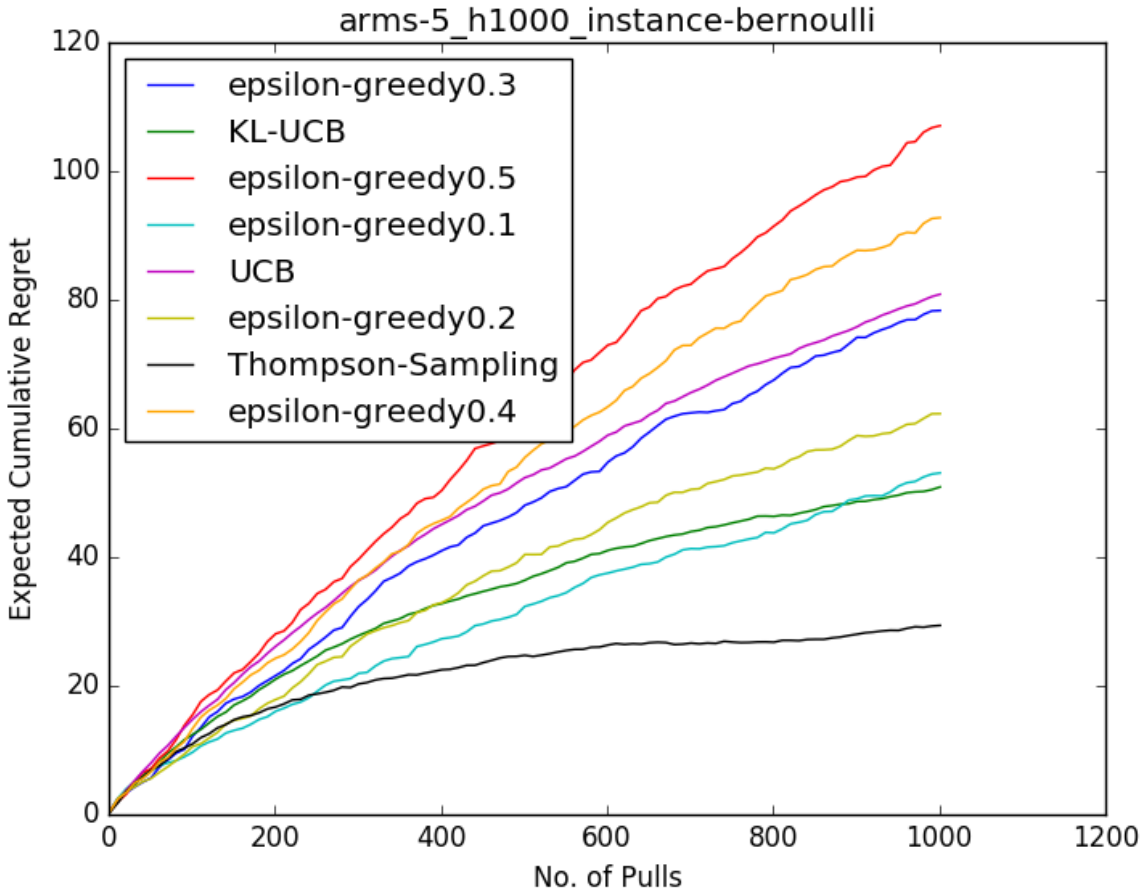
Changes to account for non-bernoulli rewards:
In ***epsilon greedy, UCB and KL-UCB*** we make use of the average reward so far for each arm. So, there wasn't any significant change in these cases. We just add the rewards(even if they are non bernoulli) and divide them by the number of pulls for each arm to maintain the average reward per arm. This is the same method we would use even for bernoulli rewards.
For ***Thompson sampling*** in case of bernoulli rewards, we maintained alpha and beta as alpha = (1 + # of successes) and beta = (1 + # of failures), where reward = 1 means success and reward = 0 means failure.
So, to handle non-bernoulli rewards the changes I made are:
alpha = (1 + total reward for the arm) and beta = (1 + (number of pulls for the arm – total reward for the arm))
This is intuitive, because the case of reward = 1 and reward = 0 translates exactly as for bernoulli case. That is, in case of reward = 1 (success), we add 1 to the alpha and 1-1 = 0 to the beta to update the parameters. And similarly, for case of reward = 0(failure), we add 0 to alpha and 1-0 = 1 to beta to update the parameters. So, intuitively this makes sense. Also, in case of reward = 0.5, we see both alpha and beta increase by equal amount which should be the case.

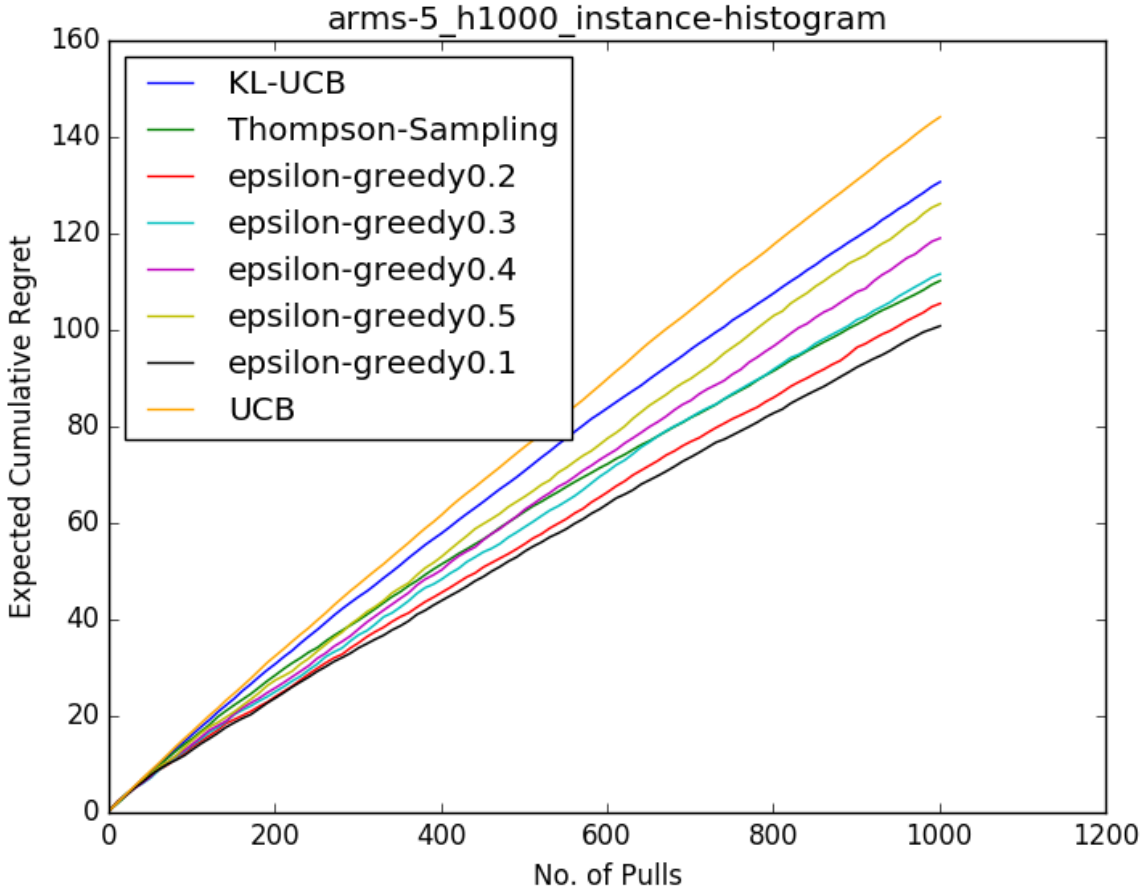
Other relevant implementation details:
The only relevant implementation detail worth mentioning is how I found the q value (max value in (p,1) which satisfies the inequality condition taught in class)
So, if I solve for the equality of the equation, then I have found the max q value satisfying the equation. For solving the equation, I used the Newton Raphson method primarily because the function under consideration is guaranteed to be strictly convex and monotonically increasing in (p,1) and hence would converge. Also, Newton Raphson algorithm takes mostly lesser number of steps to converge compared to other naive algorithms like Binary Search and Linear Search. Also, the initialisation of q while solving for q was taken to be q=0.99999(as close to 1 as possible as at q=1 it never satisfies the inequality, i.e. the value of the function I have considered is greater than 0). We can't have q=1 as then the function blows up.
Handling the corner cases -> If p=1, the range reduces to (1,1), thus the q value is returned as 1. For p=0, we get a direct expression which we can solve in 1 step for the q value satisfying the equality, and hence I have directly solved the equation without any iterative steps.
For other algorithms I used the standard gsl library functions as and when required.

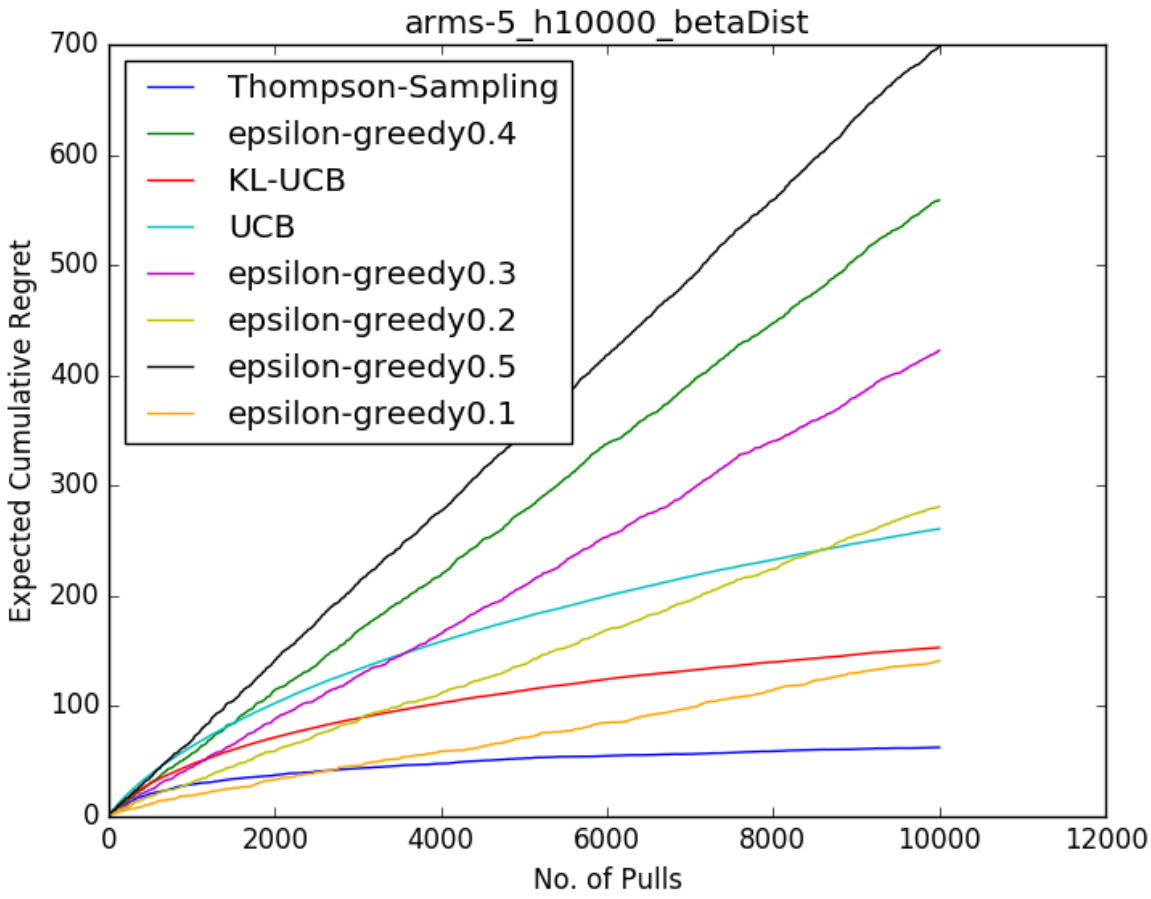
In the rest of the report are attached the graphs for the different combination of algortihms, data instances, horizons.

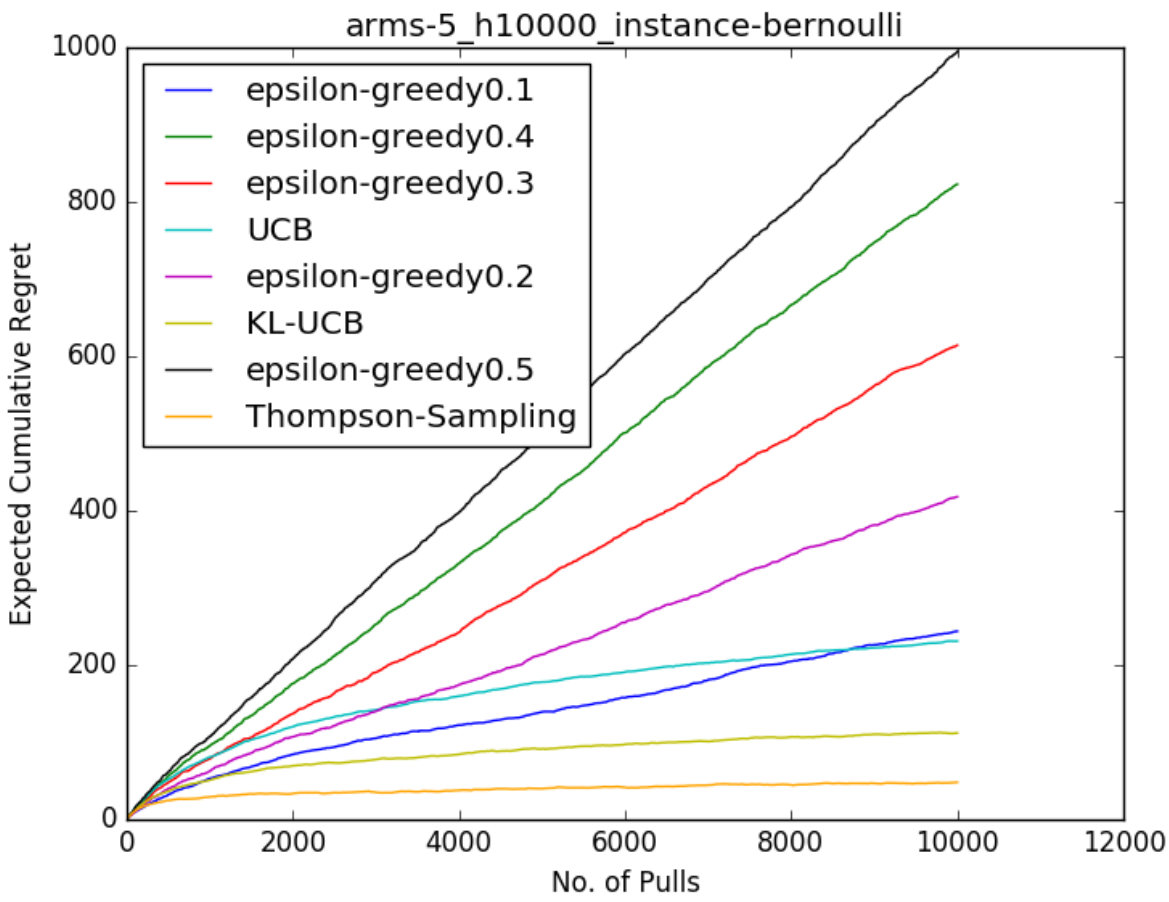
The graph names are of the form –
arms-<number of arms>_h<horizon>_<distribution of the data instance>

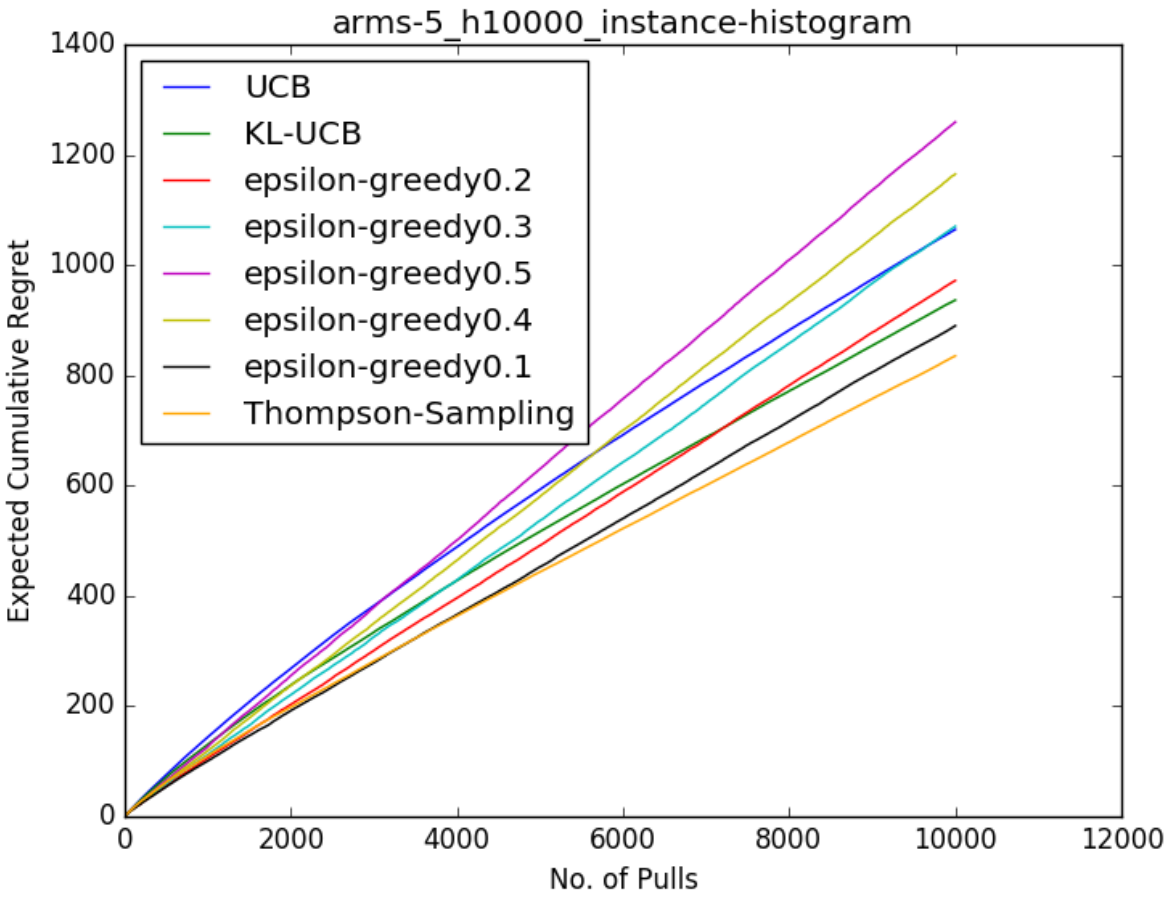


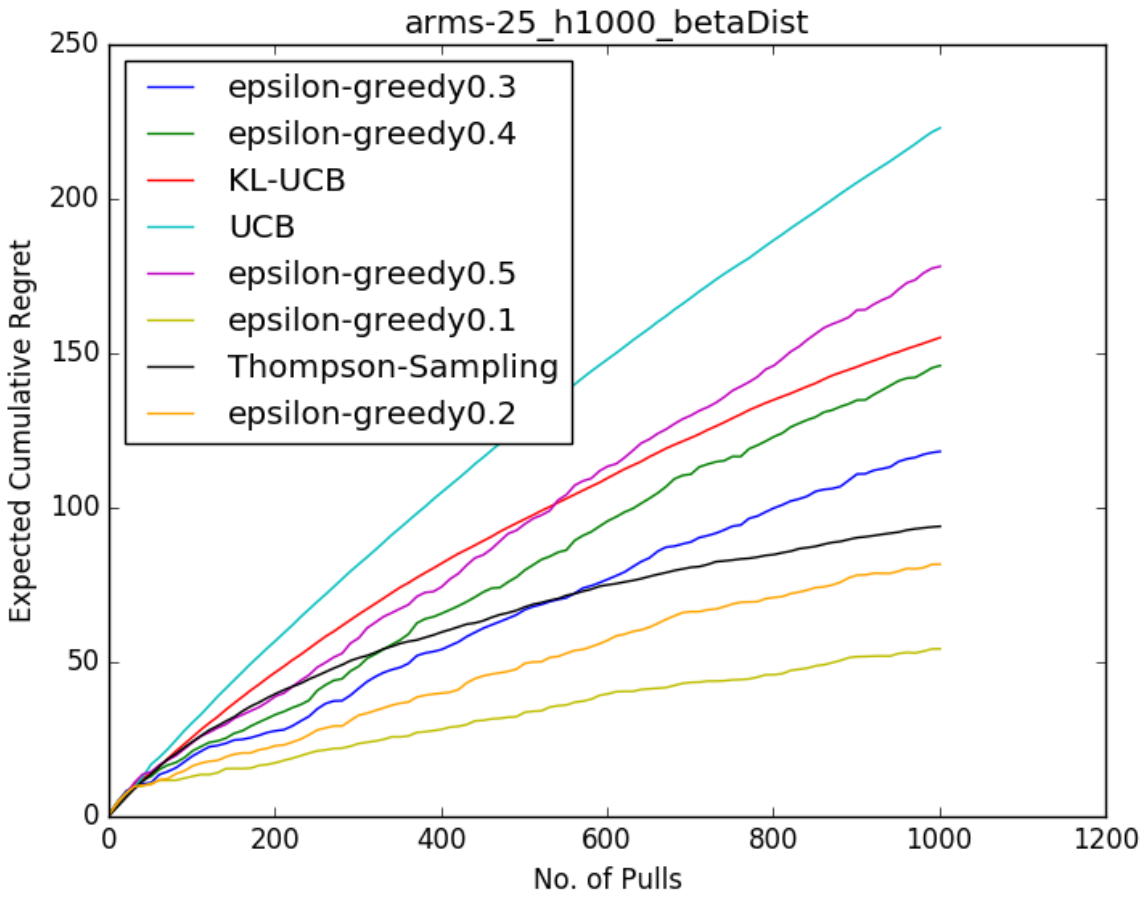

arms-5_h1000_instance-bernoulli
epsilon-greedy0.3
KL-UCB
epsilon-greedy0.5
epsilon-greedy0.1
UCB
epsilon-greedy0.2
Thompson-Sampling
epsilon-greedy0.4
Expected Cumulative Regret
No. of Pulls
0
20
40
60
80
100
120
0
200
400
600
800
1000
1200

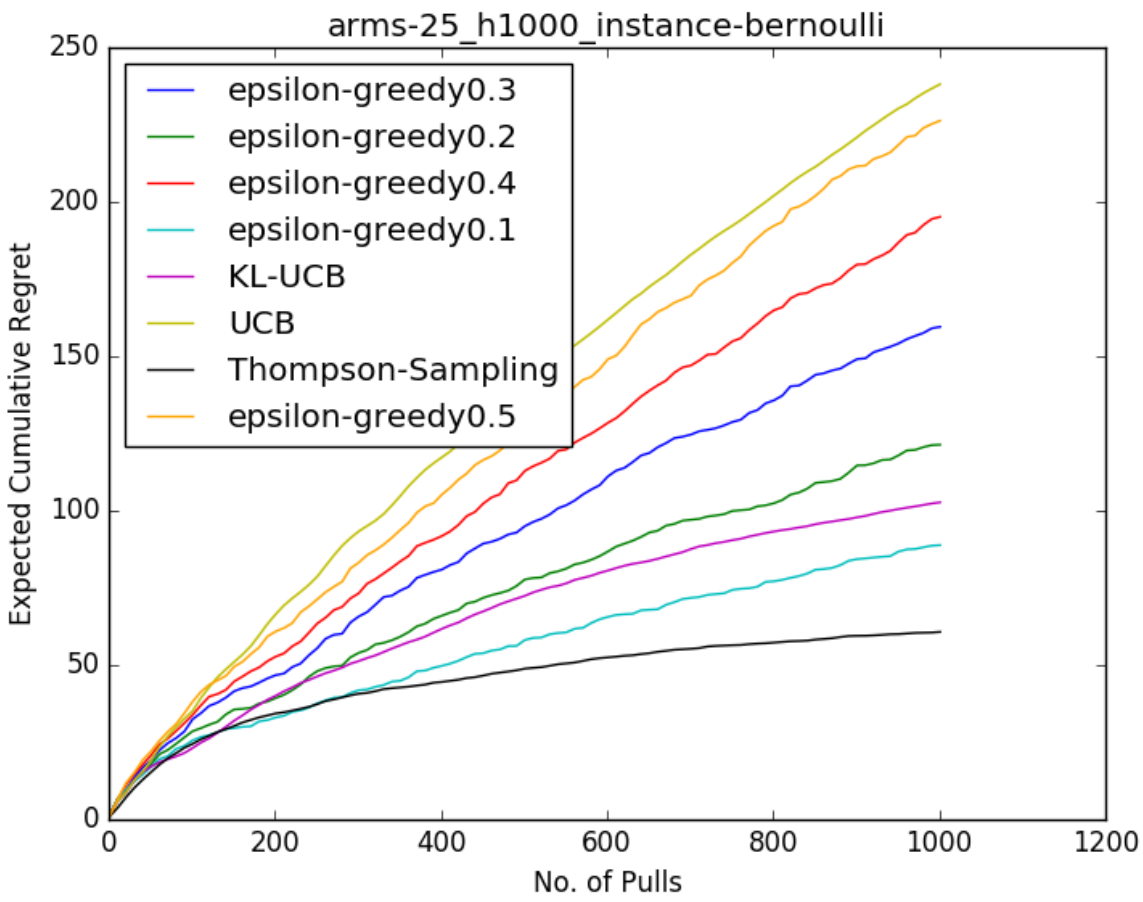

arms-5_h1000_instance-histogram
KL-UCB
Thompson-Sampling
epsilon-greedy0.2
epsilon-greedy0.3
epsilon-greedy0.4
epsilon-greedy0.5
epsilon-greedy0.1
UCB
Expected Cumulative Regret
No. of Pulls
0
20
40
60
80
100
120
140
160
0
200
400
600
800
1000
1200

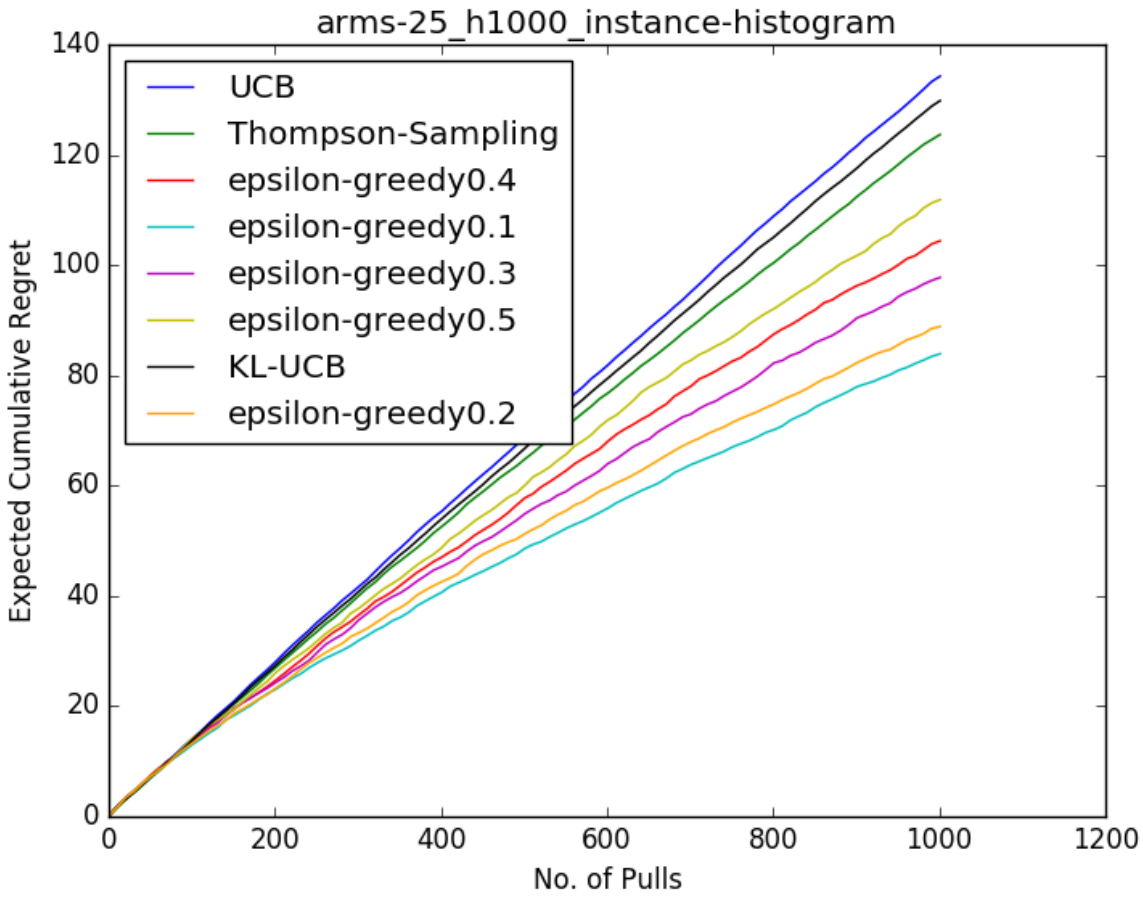

arms-5_h10000_betaDist
Thompson-Sampling
epsilon-greedy0.4
KL-UCB
UCB
epsilon-greedy0.3
epsilon-greedy0.2
epsilon-greedy0.5
epsilon-greedy0.1
Expected Cumulative Regret
No. of Pulls
0
100
200
300
400
500
600
700
0
2000
4000
6000
8000
10000
12000

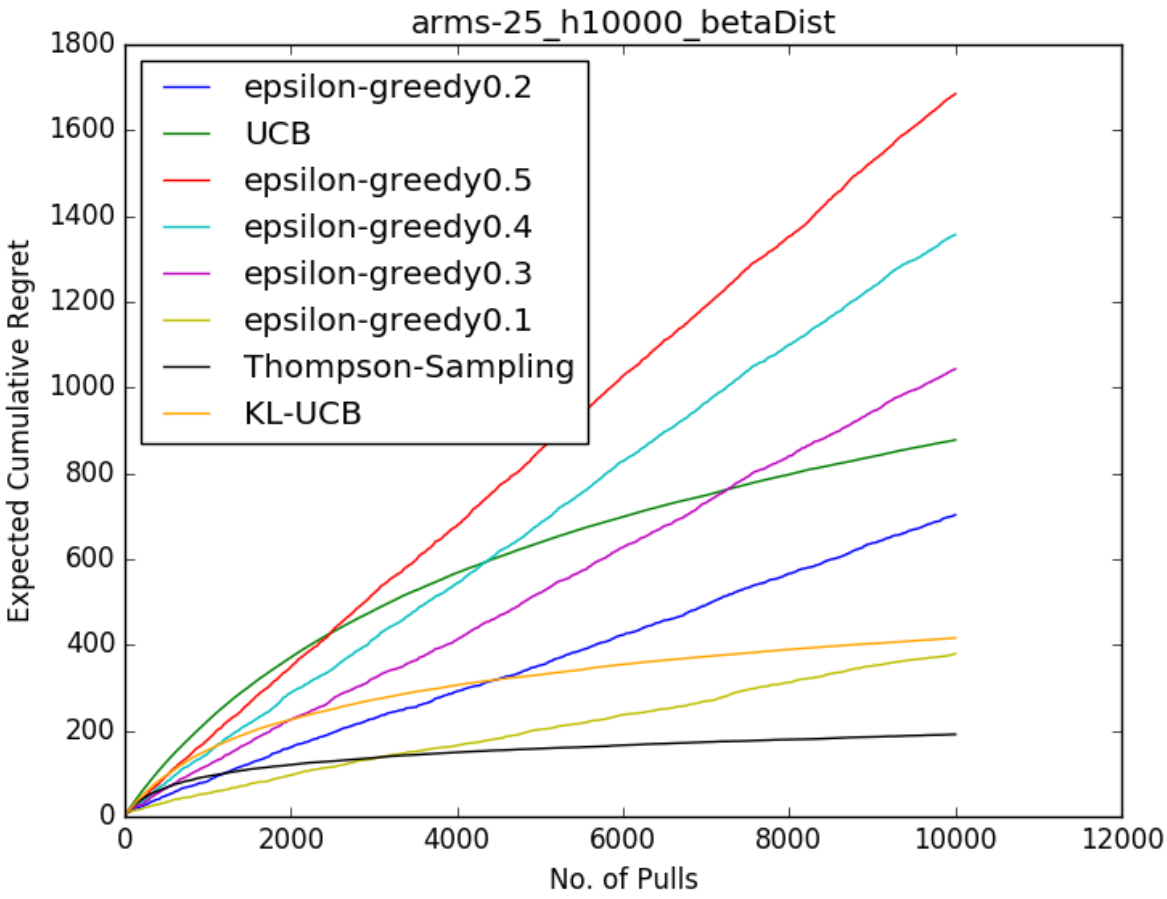

arms-5_h10000_instance-bernoulli
epsilon-greedy0.1
epsilon-greedy0.4
epsilon-greedy0.3
UCB
epsilon-greedy0.2
KL-UCB
epsilon-greedy0.5
Thompson-Sampling
Expected Cumulative Regret
No. of Pulls
0
200
400
600
800
1000
0
2000
4000
6000
8000
10000
12000

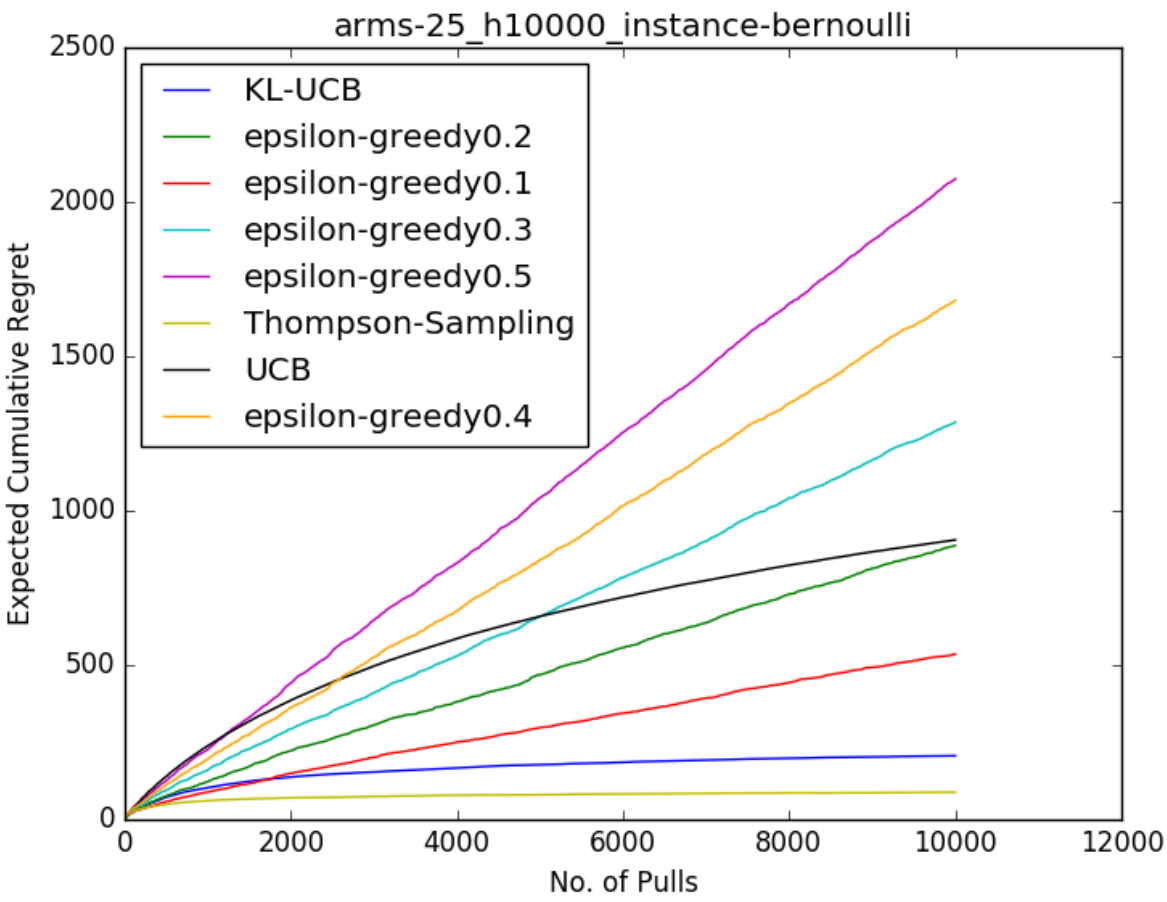

arms-5_h10000_instance-histogram
UCB
KL-UCB
epsilon-greedy0.2
epsilon-greedy0.3
epsilon-greedy0.5
epsilon-greedy0.4
epsilon-greedy0.1
Thompson-Sampling
Expected Cumulative Regret
No. of Pulls
0
200
400
600
800
1000
1200
1400
0
2000
4000
6000
8000
10000
12000

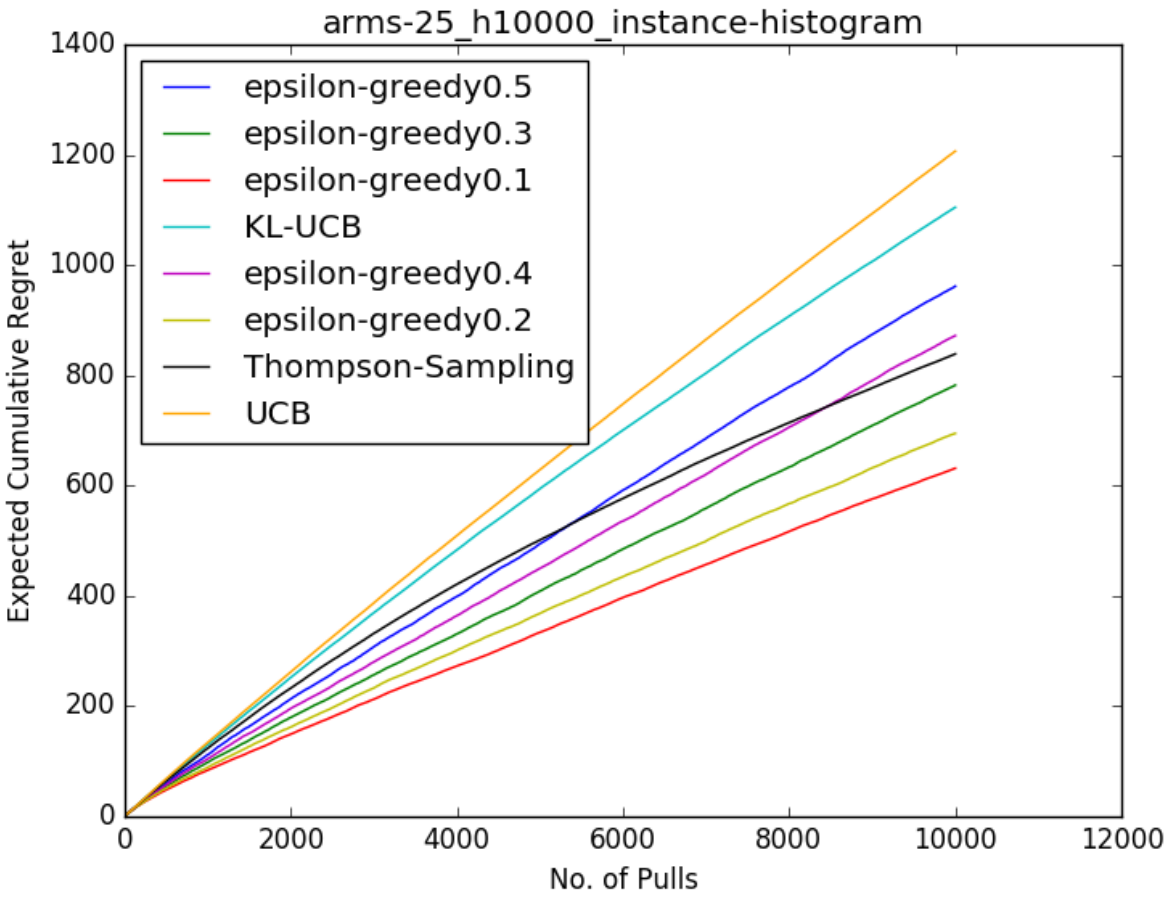

arms-25_h1000_betaDist
epsilon-greedy0.3
epsilon-greedy0.4
KL-UCB
UCB
epsilon-greedy0.5
epsilon-greedy0.1
Thompson-Sampling
epsilon-greedy0.2
Expected Cumulative Regret
No. of Pulls
0
50
100
150
200
250
0
200
400
600
800
1000
1200


arms-25_h1000_instance-bernoulli
epsilon-greedy0.3
epsilon-greedy0.2
epsilon-greedy0.4
epsilon-greedy0.1
KL-UCB
UCB
Thompson-Sampling
epsilon-greedy0.5
Expected Cumulative Regret
No. of Pulls
0
50
100
150
200
250
0
200
400
600
800
1000
1200


arms-25_h1000_instance-histogram
UCB
Thompson-Sampling
epsilon-greedy0.4
epsilon-greedy0.1
epsilon-greedy0.3
epsilon-greedy0.5
KL-UCB
epsilon-greedy0.2
Expected Cumulative Regret
No. of Pulls
0
20
40
60
80
100
120
140
0
200
400
600
800
1000
1200


arms-25_h10000_betaDist
epsilon-greedy0.2
UCB
epsilon-greedy0.5
epsilon-greedy0.4
epsilon-greedy0.3
epsilon-greedy0.1
Thompson-Sampling
KL-UCB
Expected Cumulative Regret
No. of Pulls
0
200
400
600
800
1000
1200
1400
1600
1800
0
2000
4000
6000
8000
10000
12000


arms-25_h10000_instance-bernoulli
KL-UCB
epsilon-greedy0.2
epsilon-greedy0.1
epsilon-greedy0.3
epsilon-greedy0.5
Thompson-Sampling
UCB
epsilon-greedy0.4
Expected Cumulative Regret
No. of Pulls
0
500
1000
1500
2000
2500
0
2000
4000
6000
8000
10000
12000


arms-25_h10000_instance-histogram
epsilon-greedy0.5
epsilon-greedy0.3
epsilon-greedy0.1
KL-UCB
epsilon-greedy0.4
epsilon-greedy0.2
Thompson-Sampling
UCB
Expected Cumulative Regret
No. of Pulls
0
200
400
600
800
1000
1200
1400
0
2000
4000
6000
8000
10000
12000