CS747 Programming Assignment 1 Report

Uzair Sayed, 17D070032

1 Task 1

1. Epsilon Greedy

   A uniform random variable is sampled at the beginning of each step. If the value sampled is less than ϵ, we pull a random arm (exploration). Otherwise, we pull the arm which we expect to give the maximum reward in that iteration. Ties are broken randomly. At t=0, the expected reward is taken to be 0 for all arms and the epsilon greedy algorithm starts without any initial round robin exploration.

2. UCB

   The number of pulls is first incremented to one for all the arms by performing a round robin exploration. After this the arm with the maximum UCB value is pulled. Ties are broken randomly. The UCB formula is as shown below and is in accordance with what was derived in class:

```
class sampler(bernoulliArms):
    def ucb(self):
                pulls = self.armpulls * 1.0
                uta = np.ones_like(pulls)
                uta[:] *= ( ((2 * np.log(self.totalPulls))) / pulls[:] )**0.5
                ucb = self.Pavg + uta
                arm = argmax(ucb)
                return self.pull(arm)
```

3. KL-UCB

   A round robin exploration is done to ensure all arms have been pulled once. After this the arm with the maximum KL-UCB value is pulled. Here, KL mean the KL divergence. The KL-UCB matrix is calculated keeping c=3, and is looped until a precision of less than 10e-6 is achieved. Binary Search is used to perform updation. The KL-UCB formula is as shown below and is in accordance with what was derived in class:

```
class sampler(bernoulliArms):
    def kl(p,q):
                if(p == 0):
                        return (1-p)*np.log((1-p)/(1-q))
                if p==1:
                        return p*np.log(p/q)
                return p*np.log(p/q) + (1-p)*np.log((1-p)/(1-q))
    def klUCB(self, c = 3, precision = 1e-06):
                klucb = np.zeros(self.k)
                t = self.totalPulls
                logt_term = np.log(t) + c*np.log(np.log(t))
                for i in range(self.k):
                        p = self.Pavg[i]
                        RHS = logt_term / self.armpulls[i]
                        if(p == 1 or RHS < 0):
```

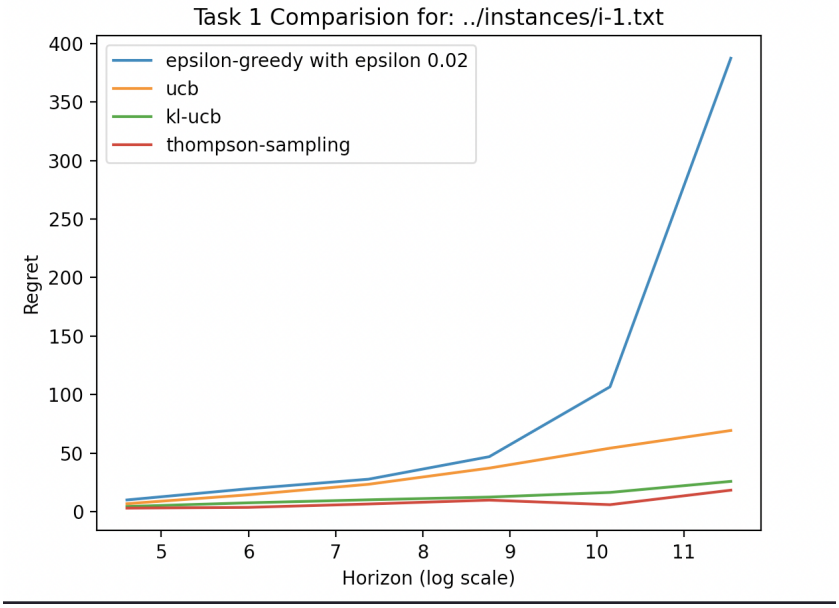
```
                    klucb[i] = p
                    continue
                lb, ub = p, 1.0
                q = (ub + p)/2.0
                LHS = kl(p,q)
                while(not isclose(LHS , RHS, precision)):
                    if(LHS > RHS): ub = q
                    elif(LHS < RHS):lb = q
                    q = (ub + lb)/2.0
                    LHS = kl(p,q)
                klucb[i] = q
        arm = argmax(klucb)
        return self.pull(arm)
```

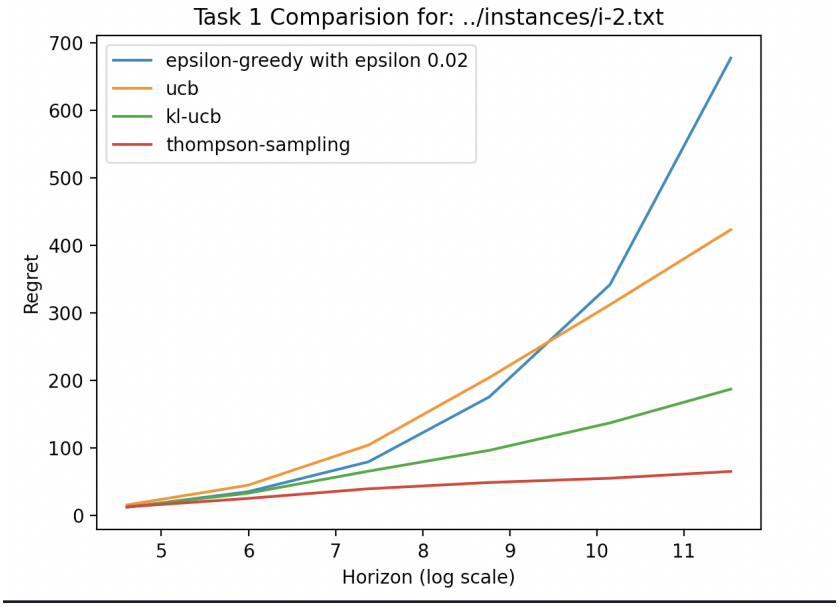
4. Thompson Sampling

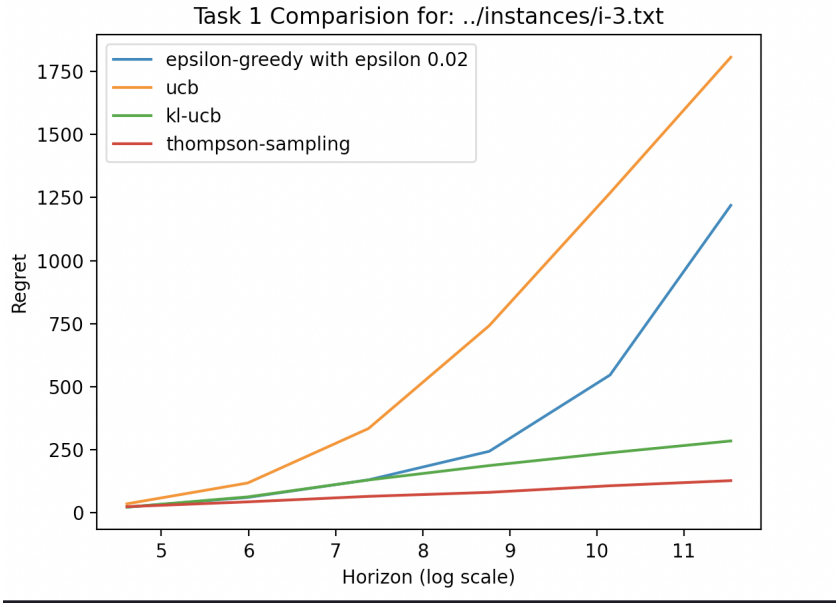
   We simply sample a Beta-distribution for each arm. The parameters for the beta distribution are defined as: α = sum of rewards + 1, and β = number of arm pulls - the sum of rewards + 1. Np.argmax is used to find the selected arm.

5. The Plots

   Note that the horizon is plotted in log scale. The values of the Horizons that were considered are [100, 400, 1600, 6400, 25600, 102400].







The plots for KL-UCB and Thompson sampling look linear with respect to the logarithmic 'Horizon axis'. This seems to be consistent with their logarithmic E(regret). Moreover, we can observe that Thompson sampling performs better than kl-UCB in all instances.
The plots corresponding to the epsilon greedy algorithm are exponential for all the instances(with respect to log(horizon)). Thus, confirming its linear regret.

2 Task 2

1. The Hint

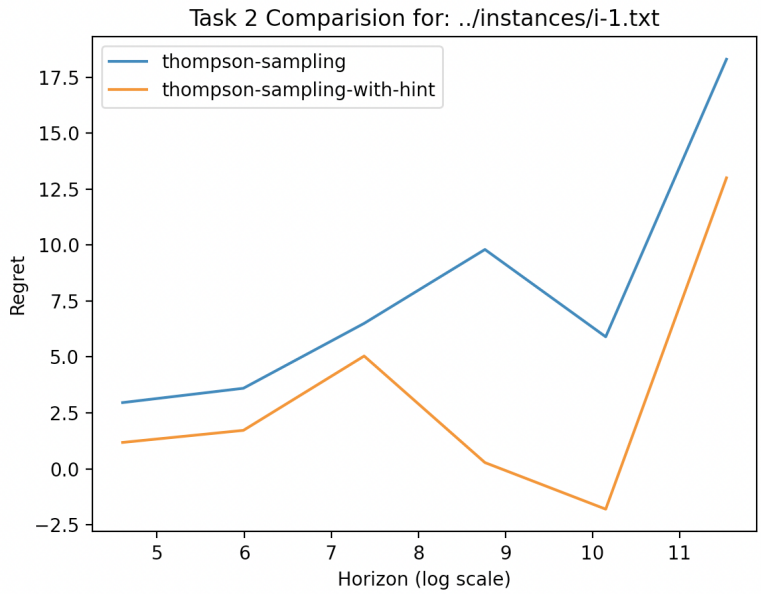
   The hint is a random permutation of the arms in the instance. We just use a single integer value (P*) for each instance.

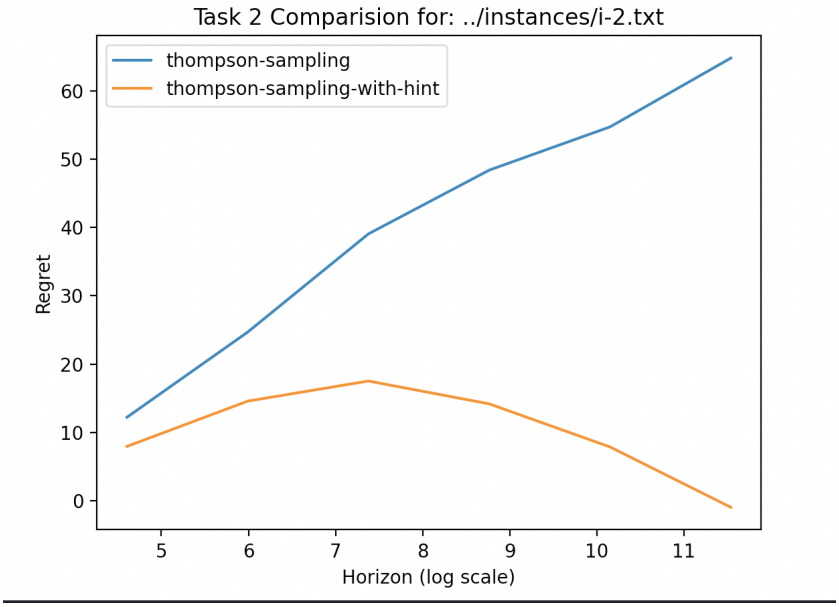
2. Thompson Sampling with Hint

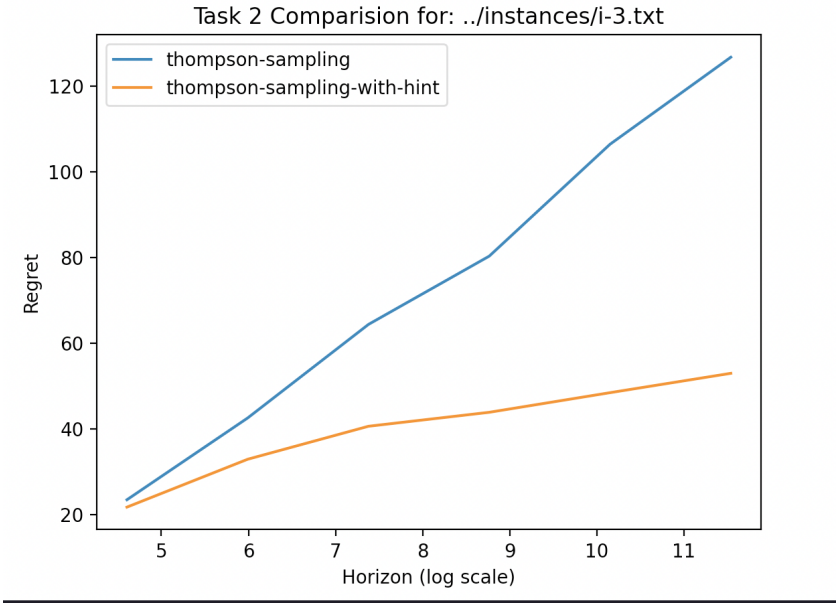
Just as with Thompson Sampling, we sample from the beta distribution with the exact same parameters as we showed earlier. However instead of simply choosing the arm that has the maximum beta value, we include a bias to the Bayesian prior estimate.
Since we know the best arm's mean, we can use the KL divergence to measure how different is an arms distribution from the 'ideal' one. KL divergence, scaled, acts as our bias.

```
class sampler(bernoulliArms):
        def hintedThompson(self):
                hint_ls = self.optimalArm()
                s = self.Psum
                f = self.armpulls - s
                beta = np.random.beta(s+1, f+1)
                for a in range(self.k):
                        beta[a] += np.exp(-self.armpulls[a]*kl(self.Pavg[a], hint_ls))
                arm = np.argmax(beta)
                return self.pull(arm)
```

3. The Plots







The algorithm- 'Thomas sampling with hint' clearly does better than Thomas Sampling in all the given instances and horizons. The algorithm gives negative regret in some cases.

3 Task 3

Instance	ϵ=0.001	ϵ=0.02	ϵ=0.999
i-1	434.02	387.48	20496.3
i-2	2479.86	677.78	20458.94
i-3	3717.5	1219.2	42275.22

With $\epsilon = 0.001$, we are exploring less than when compared to $\epsilon = 0.02$, and with $\epsilon = 0.999$, we are exploiting less.