CS747 – Assignment 1 Report

Bandit algorithms

Mahesh Abnave (203059010)

1. Algorithms

- Implemented following three algorithms:
  1. Epsilon greedy
  2. UCB
  3. KL-UCB
  4. Thompson sampling
  5. Thompson sampling with hint

2. Code

- There are py files corresponding to each algorithm.
- Bandit_instace.py emulates pulling arm once initialized with weights.
- For number of horizons equal to number of arms, round robin approach is followed to pull each arm once.

3. Task 1 – Epsilon greedy vs UCB vs KL-UCB vs Thompson sampling

3.1. Plots

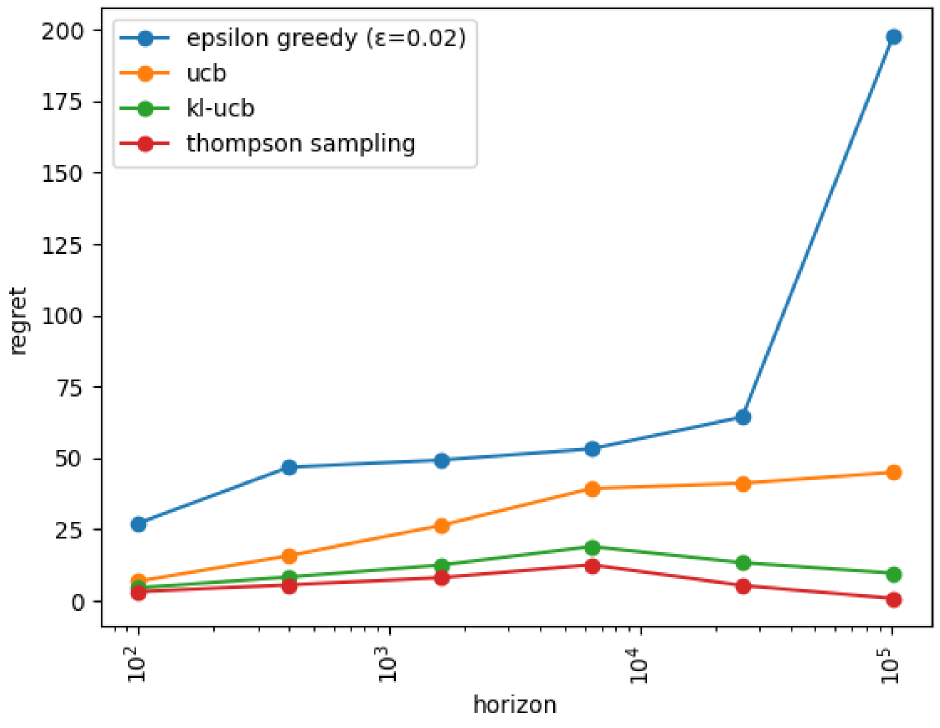

Fig. Evaluating regrets of vairous algorithms for instance 1

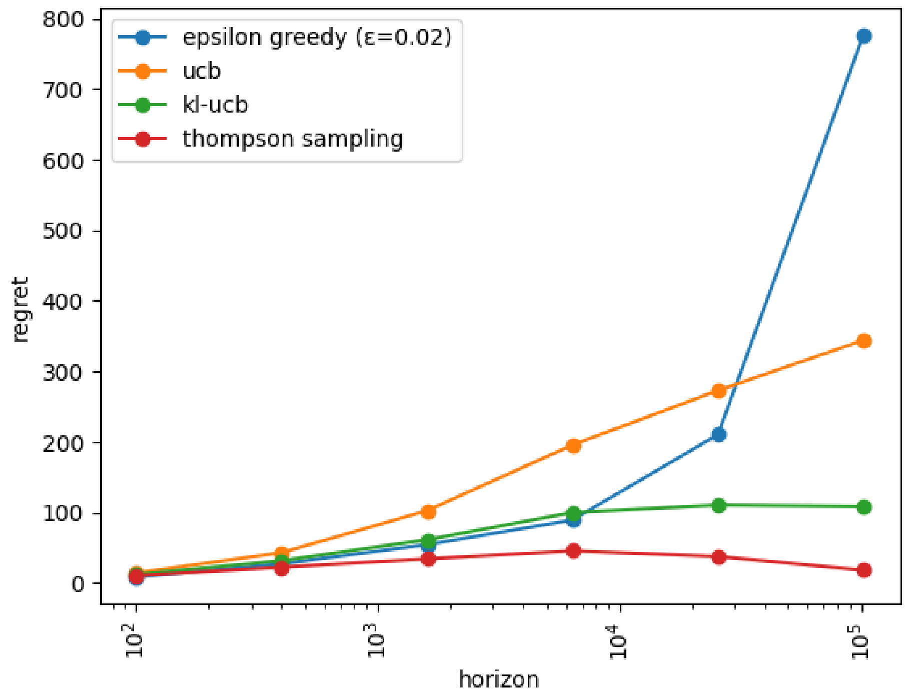

Fig. Evaluating regrets of vairous algorithms for instance 2

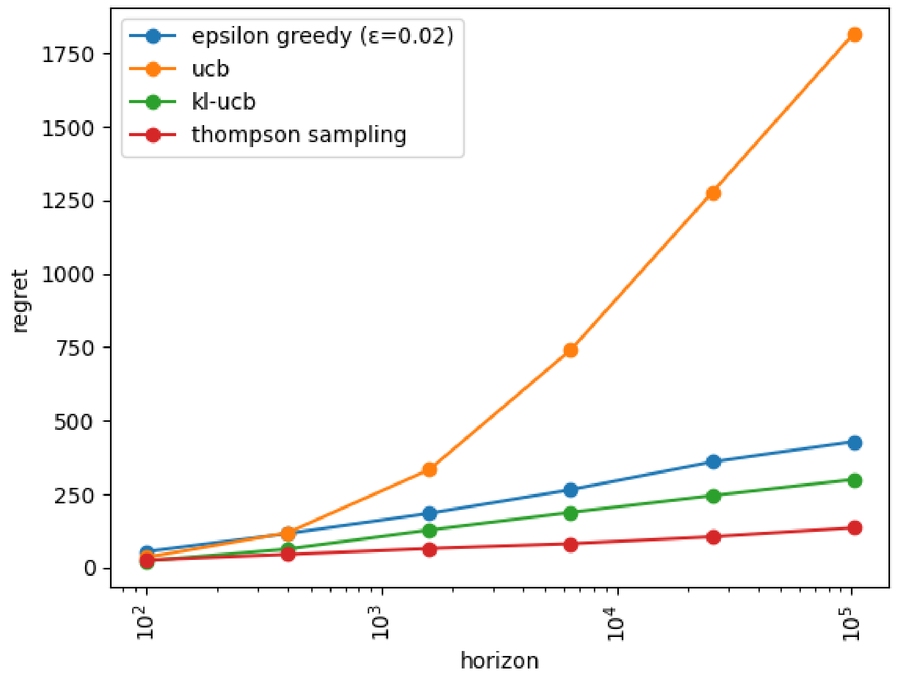

Fig. Evaluating regrets of vairous algorithms for instance 3

3.2. Observations

- All algorithms gives sublinear regret.
- Thompson sampling performs best in all bandit instances.
- KL-UCB behaves second best in all bandit instances.

- UCB seems to take time to converge to true mean as number of arms increases. (As for two arm bandits, it performs better than epsilon greedy. For five arm bandit, it initially performs poorer than epsilon greedy, but eventually ends up performing better than it.)

4. Task 2 - Thompson sampling vs Thompson sampling with hint

4.1. Plots

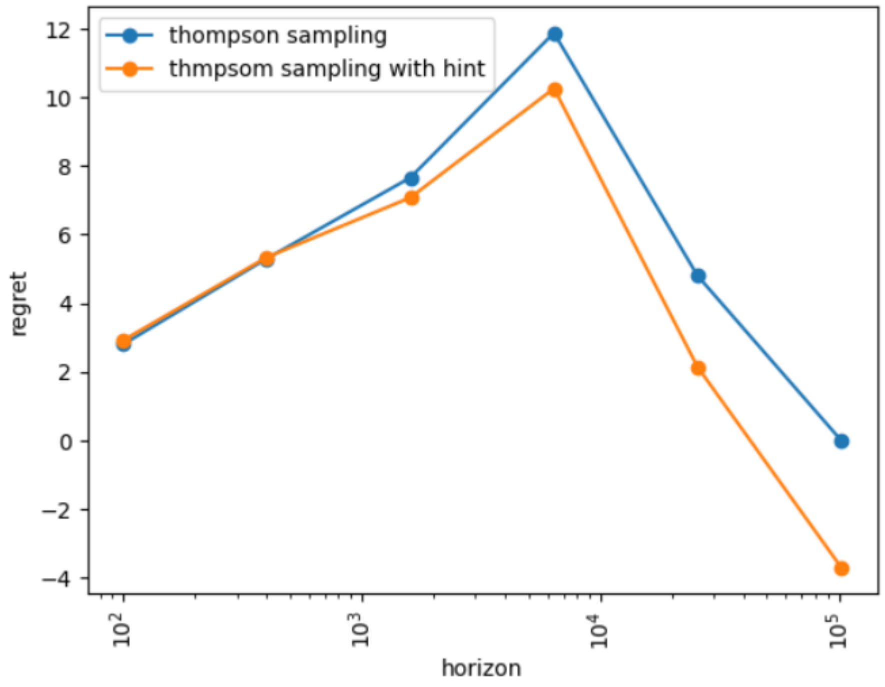

Fig. Evaluating regrets of vairous algorithms for instance 1

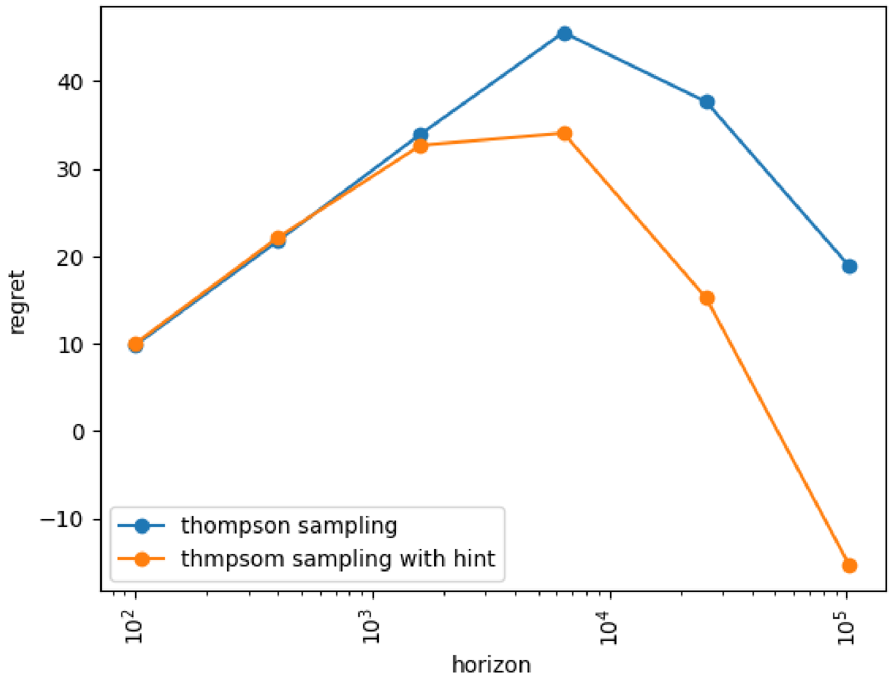

Fig. Evaluating regrets of vairous algorithms for instance 2

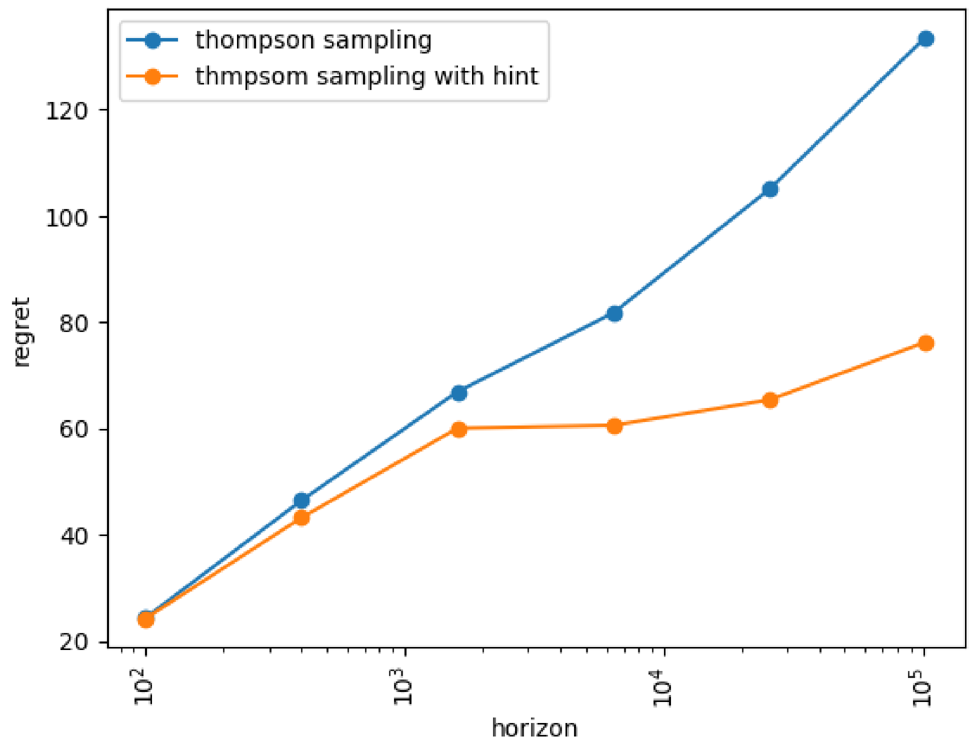

Fig. Evaluating regrets of vairous algorithms for instance 3

4.2. Idea of Thompson Sampling with hint algorithm

The idea is simple. For initial iteration, we startup with normal Thompson sampling algorithm. And once we find optimal arm, we keep exploiting it. For finding optimal arm, we perform additional tasks in each iteration. Say for first iteration, we sample according to normal Thompson sampling algorithm. Then we check which arm has empirical mean (EM) closest to max true mean (MTM). We “note down” index of this arm. In addition, we increment counter which tells how many “last consecutive” times, an arm is found to have EM closest to MTM. So if in second iteration, we find that the same arm has EM closest to MTM, we increment this counter. Now, say counter is 100 and we find new arm to have EM closest to MTM. Then we note down index of this new possibly-optimal arm and reset counter to 1. We assume arm is optimal once its found to have EM closest to MTM for (threshold) 1000 consecutive iterations. Then onwards we simply exploit this possibly-optimal arm (that is always pull this arm instead of one as per naïve Thompson sampling). Once arm is found to be optimal, we still keep checking if its EM falls below EM of other arms. And if this happens, we reset the whole process again.

4.3. Pseudocode for Thompson sampling with hint algorithm

(# green text is comment)

```
1. Initialise:
   last_arm_with_em_closest_to_max_tm = -1
   no_of_times_arm_has_em_closest_to_max_tm = 0 #for last "consecutive" iterations t
   optimal_arm_determined = False
   max_true_mean = max(true_means)
2. For t = 1 to no_of_arms: #initially pull each arm once
     call pull_arm_and_update_values(arm_to_pull = t)
3. For t > no_of_arms:
     if NOT optimal_arm_determined:
        beta_per_arm = sample all arms from beta distribution
        arm_to_pull = argmax(beta_per_arm)
     else:
        arm_to_pull = last_arm_with_em_closest_to_max_tm
   call pull_arm_and_update_values(arm_to_pull)
4. function pull_arm_and_update_values(arm_to_pull)
     number_of_pulls(arm_to_pull)++
     if reward == 1:
        number_of_successes[arm_to_pull]++
     else:
        number_of_failures[arm_to_pull]++
     empirical_means[arm_to_pull] = #EM formula
     if NOT optimal_arm_determined:
        arm_with_em_closest_to_max_tm = #arm which has its EM closest to max TM
        if last_arm_with_em_closest_to_max_tm == arm_with_em_closest_to_max_tm:
           # arm with its EM closest to max TM is same as that in last iteration t
           no_of_times_arm_has_em_closest_to_max_tm++
        else:
           # arm with its EM closest to max TM is different than that in last iteration t
           last_arm_with_em_closest_to_max_tm = arm_with_em_closest_to_max_tm
           no_of_times_arm_has_em_closest_to_max_tm = 1
        if no_of_times_arm_has_em_closest_to_max_tm == 1000:
           # if arm is found to have EM closest to max TM for last 1000 "consecutive"
           # iterations, assume it's the optimal arm
           optimal_arm_determined = True
     else: # that is if optimal_arm_determined = True
         # check if the arm which we assumed to be optimal is really optimal
         arm_with_em_closest_to_max_tm = #arm which has its EM closest to max TM
         if last_arm_with_em_closest_to_max_tm == arm_with_em_closest_to_max_tm:
            # arm which we assumed to be optimal turned out to be sub-optimal, reset
            # everything
            optimal_arm_determined = False
            last_arm_with_em_closest_to_max_tm = arm_with_em_closest_to_max_tm
            no_of_times_arm_has_em_closest_to_max_tm = 1
```

4.4. Observations

- The Thompson sampling with hint algorithm formulated seem to perform better on all instances.
- Due to threshold 1000, it performs similar to (or “slightly” better than) naïve Thompson sampling. But after t=1000, it starts performing significantly better than naïve Thompson sampling, because of exploiting of optimal arm.

5. Task 3 - Experiments on epsilon-greedy

5.1. Observations

For $\epsilon = 0.002$, $\epsilon = 0.02$ and $\epsilon = 0.2$, it was found that epsilon greedy algorithm with $\epsilon = 0.02$ performs better than the other two with significant margin except for instance 2. For instance 2, it performs “slightely” poorer than $\epsilon = 0.002$, that too for higher horizon. For instance 2, for not so bigger horizon ($< 10^5$), $\epsilon = 0.02$ still performs better than the other two.

5.2. Plots

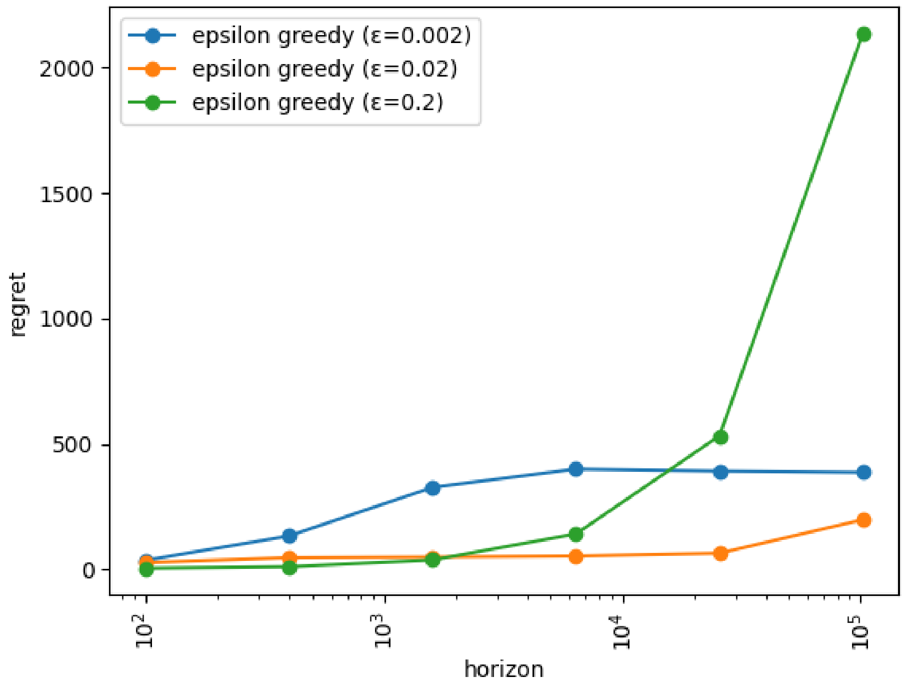

Fig. Evaluating regrets of various algorithms for instance 1

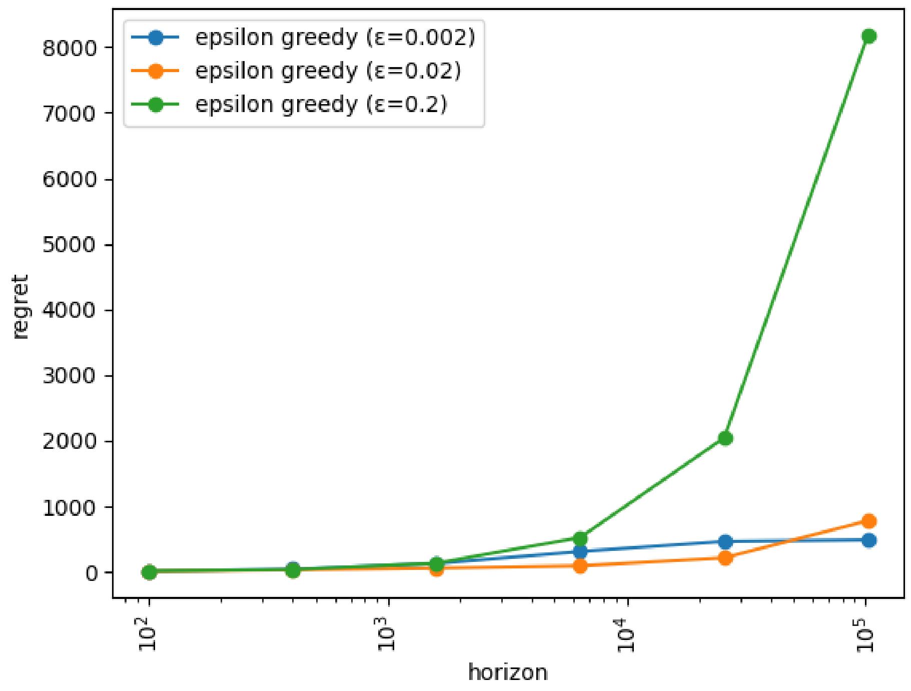

Fig. Evaluating regrets of various algorithms for instance 2

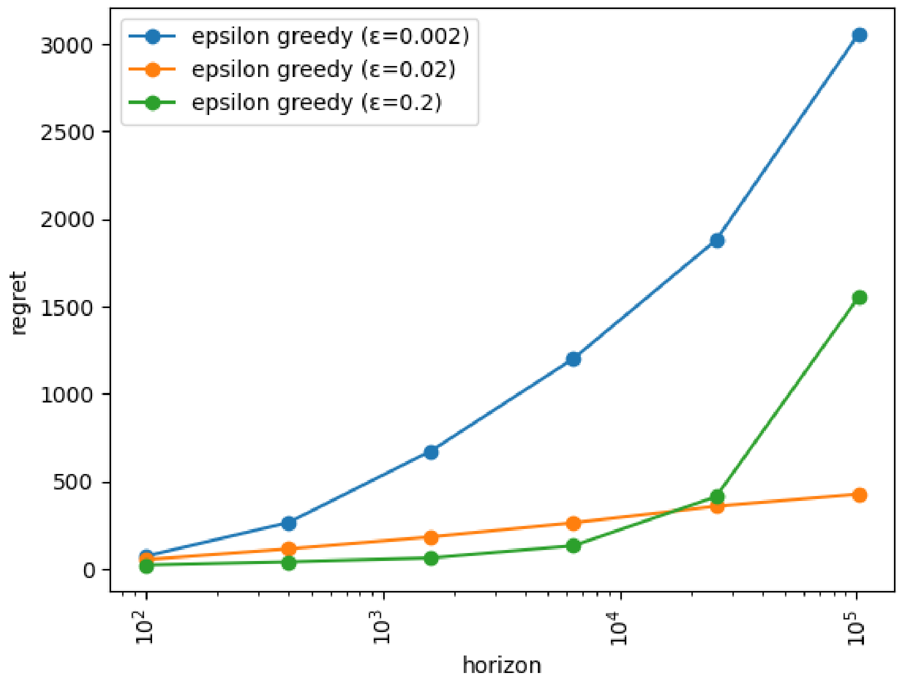

Fig. Evaluating regrets of various algorithms for instance 3