Report Lab 1

190050118 - Sudhansh

Task 1

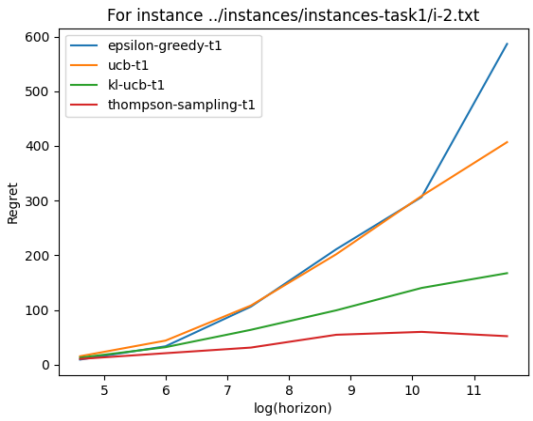
Here is the list of assumptions for this task

- I chose the Epsilon Greedy 3 algorithm for the first task.
- In case of ties, all algorithms use lowest index policy for tie breaking
- In UCB and KL-UCB, each arm is pulled once before using the bounds suggested by the algorithm.
- For task 1, the scale factor in UCB is set to 2, and the factor in the KL-UCB expression is set to 3.
- In the output, scale and threshold are printed as 0.

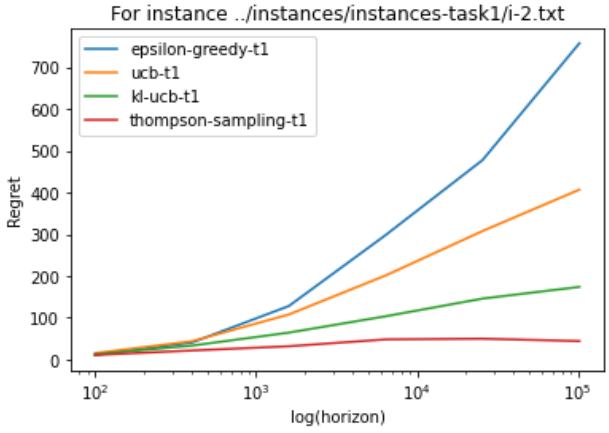
Implementation details

All the implementations are present in the `` `_impl.py` `` file.

- Each function is documented clearly in the code itself.
- The epsilon greedy algorithm is implemented in the `epsilon_greedy_t1` function.
- I used binary search to determine the KL_UCB bound. This algorithm has the highest execution time.
- The code is straightforward, and implements the algorithms exactly as discussed in class.
- You can run `task1.py` to generate the complete data for this task.
- **An Important implementation aspect**. I initialised the initial empirical means to **1** instead of **0**. I noticed that this gave better results in comparison. For example, consider the plots for instance 2 -



Initialisation = 1

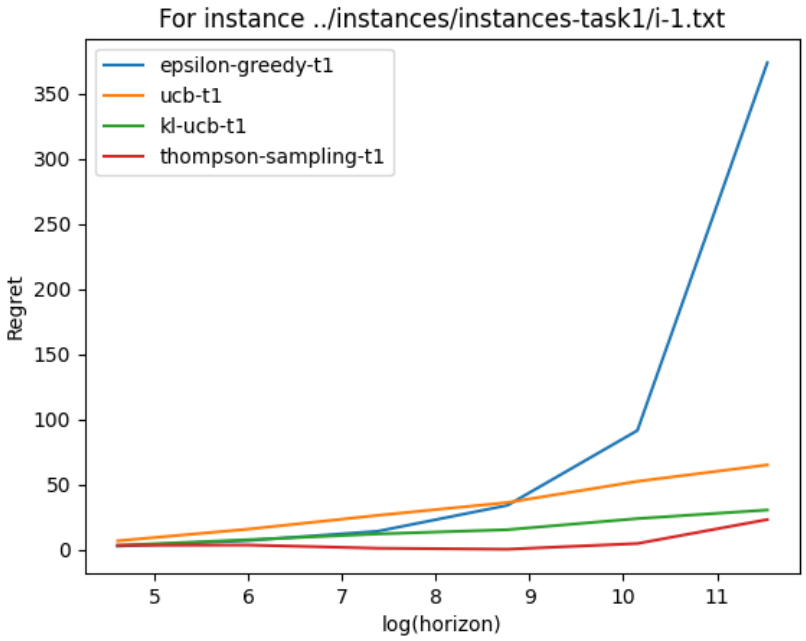


Initialisation = 0

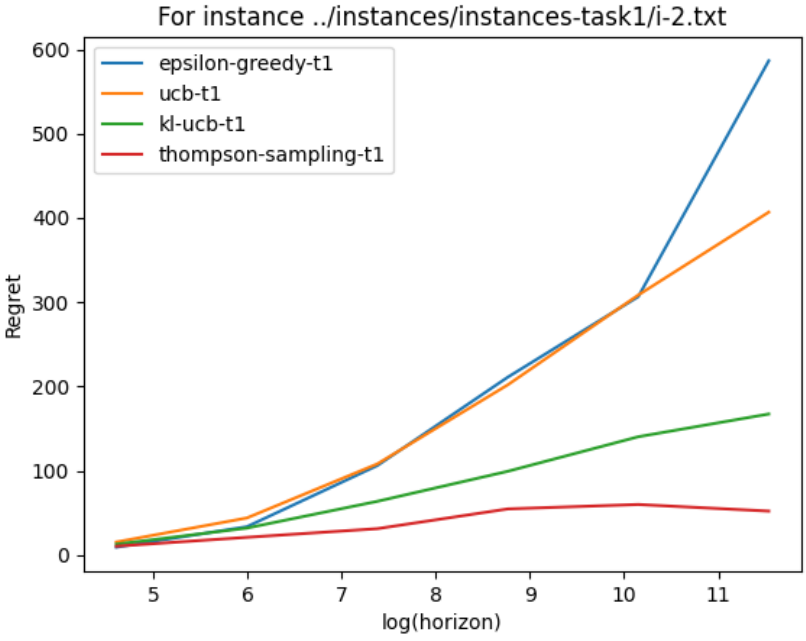
As we can see, initialising with empirical means as 1 yields lower regret.

Observations: The regret increases with horizon as expected. KL-UCB and Thompson have the lowest regret as expected. Thompson performs slightly better than KL-UCB due to implementation constraints. As expected, epsilon greedy has the highest regret.

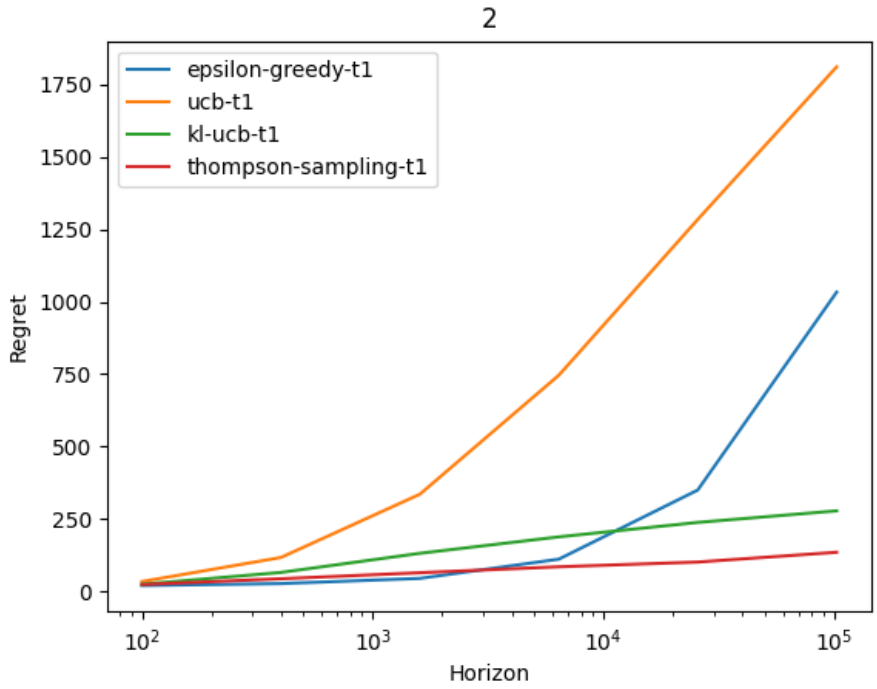
Here are the plots for this task:



Regret plots for instance 1 Task 1



Regret Values for 2nd instance of Task 1



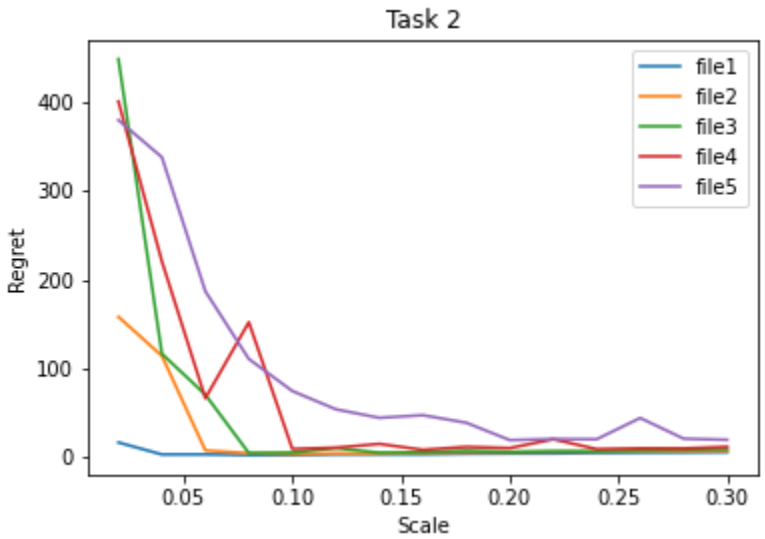
Regret Values for 3rd instance of Task 1

Also, as mentioned in the problem statement, the regrets are sometimes negative in some seeds. It may happen because our rewards all turn out to be 1's but the Tp* value in the regret expression is not as high as our reward.

This task took the highest time to run. After multiple optimizations, I could get the execution time down to 3 hours :')

Task 2

The aim of this task was to determine how the regret of UCB algorithm changes when the scale factor changes. Here is the plot with all the instances.



The output has HIGHS and threshold set to 0. Primarily, we can observe that the regret seems to be decreasing as the scale increases. One argument by intuition is that when the scale is too low, then $ucb_a^t \approx \hat{p}_a^t$. This is nothing but the epsilon greedy algorithm with epsilon 0! No wonder the regret is high for low scale values.

A mathematical argument for the same is as follows. Without writing down the whole proof, I'll refer to the step 4.3 of the sub-linearity proof of UCB algorithm done in the class. In particular we have,

$$B \leq e^{-4ln(t)} + exp(-2y(\sqrt{\frac{c^* ln(t)}{y}})^2$$

where c is the scale factor, y is the number of pulls of an arbitrary arm a, and t is the horizon. Now, notice that the RHS of the inequality decreases as "c" increases! This means that the higher the value of c, the tighter bound we have for the ucb score. In general, tighter bounds transcribe to better algorithms because we have more certainty of the true parameters of the instance.

We can also see that file 1 has almost flat regret compared to others. This is because file 1 has only 2 bandit arms, and since the horizon (10000) is very large, the regret almost doesn't change with the scale parameter. We can also see that the regret tapers off as scale increases.

Also, the values of c which gave lowest regret for each bandit instance are as follows.

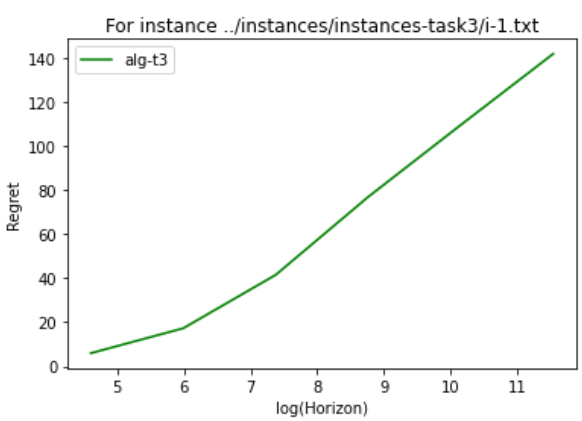
File/ Instance	**Scale Value**
1	0.08
2	0.1
3	0.08
4	0.16
5	0.2

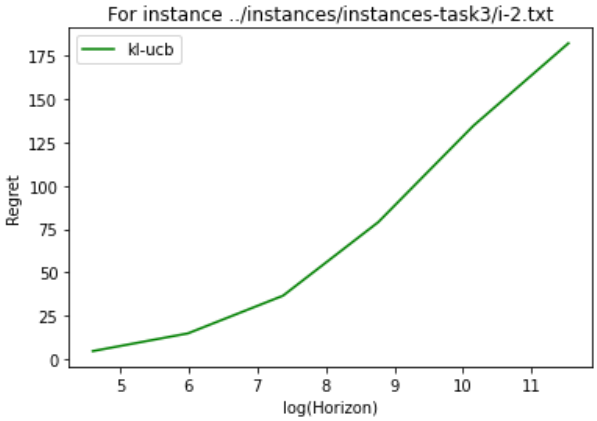
Task 3

The intuition for devising an algorithm for this task to understand “p” as the expected reward in the regret definition of our usual bandit instances. In the normal bandit instances, we try to identify the arm with the maximum expected reward, i.e., “p”. Now, essentially we do the same thing and modify the calculation of “p” accordingly.

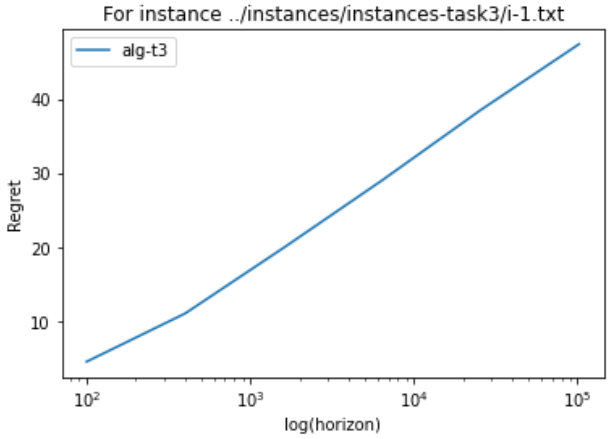
Another way to think about it is, we considered Bernoulli distribution for the arms until now. The arms are just modified to follow another distribution. This does not change how our algorithm acts on the instance. We still try to find the arm with the maximum expected reward and pull it.

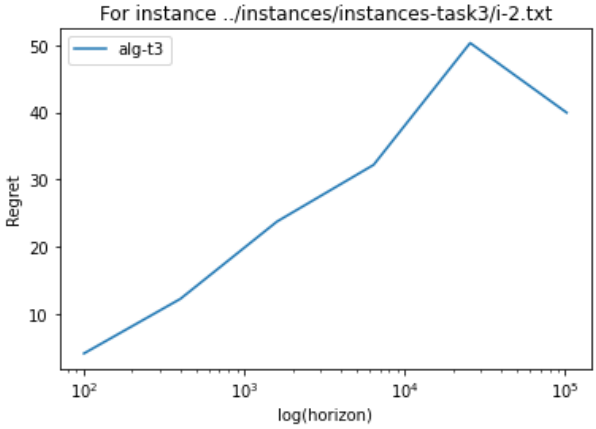
I initially tried implementing this task using KL-UCB. Here are the corresponding plots.





Then, I modified the algorithm to use Thompson Sampling method. The parameter “a” in the distribution is set to the total score from each arm, and the parameter “b” is set to the total score subtracted from the total number of tries of each arm. This gave much lower regret as can be seen in the plots.





However, there is more to this problem. The intuition behind Thompson Sampling for Bernoulli arms was that Beta distribution was a conjugate prior of Bernoulli. Now, similarly this problem is

an extension of the Bernoulli case but with a **Categorical distribution** for the likelihood. The conjugate prior for this distribution is the **Dirichlet Distribution**. We have

$$P(p) \propto \prod_{i=1}^{n} p_i^{s_i}$$

$$P(rew \mid p) \propto p_{rew}$$

Where, s_i is the total number of rewards from outcome i. Therefore, we can devise the following algorithm

- For every arm a, sample P_a^t using the following Dirichlet distribution

$$P_a^t \sim Dirichlet(1 + u_a^t)$$

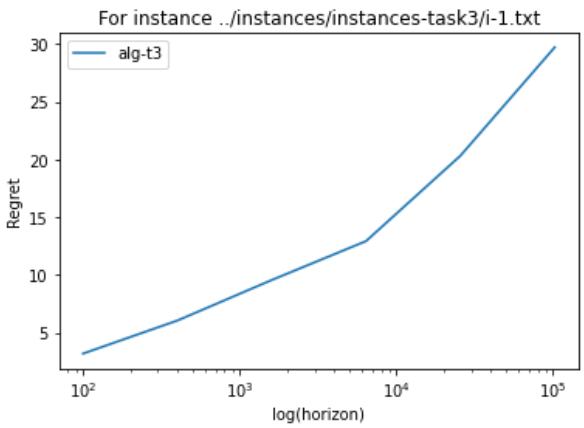
  where u_a^t is the vector containing the number of rewards from each outcome of the arm.

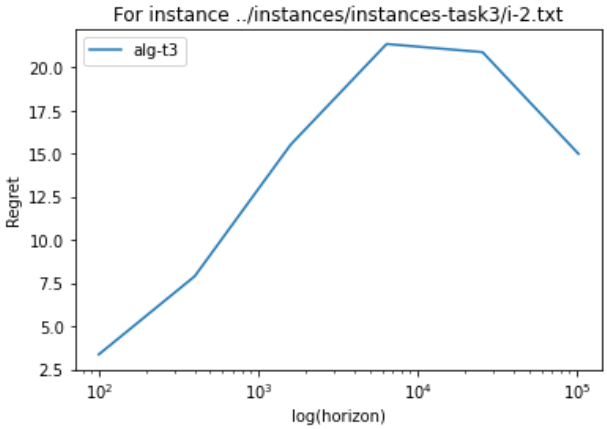
- We then compute the following quantity

$$q_a^t = < P_a^t, Rew >$$

  where Rew represents the set of possible rewards.
- We then sample the arm with the maximum q.

Here are the plots corresponding to this algorithm.





Task 4

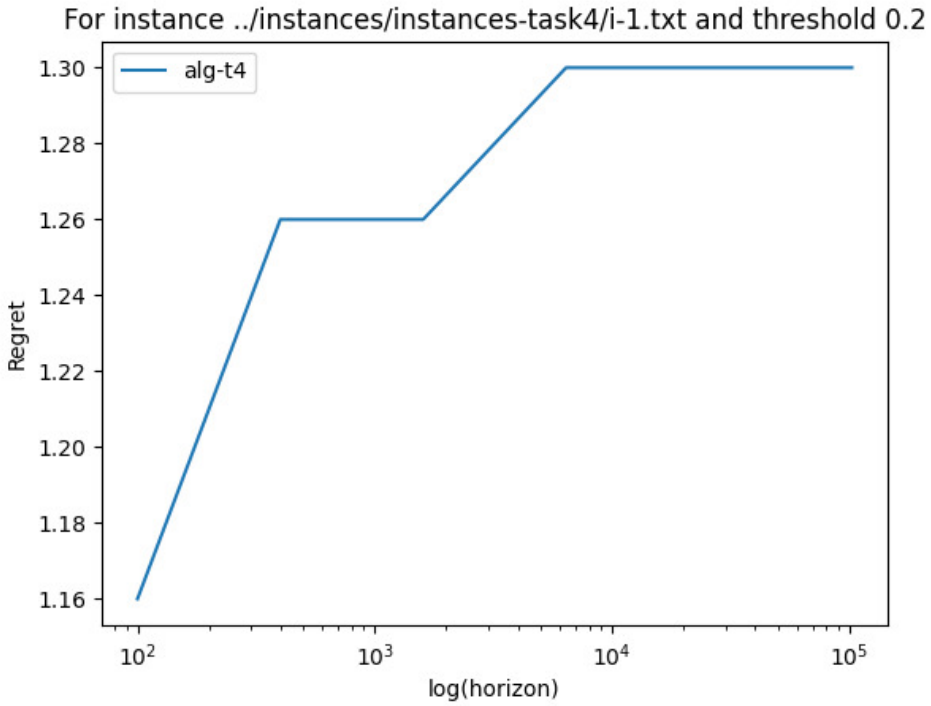
The main point to observe here is that we are trying to maximise the number of HIGHS. First, let us show this problem is equivalent to maximising the number of 1's from a binary bandit instance.

- The rewards are binary here
- The rewards from each arm can be categorised into High and Low based on the threshold. Based on this, we can calculate the probability of getting a HIGH from the arm.

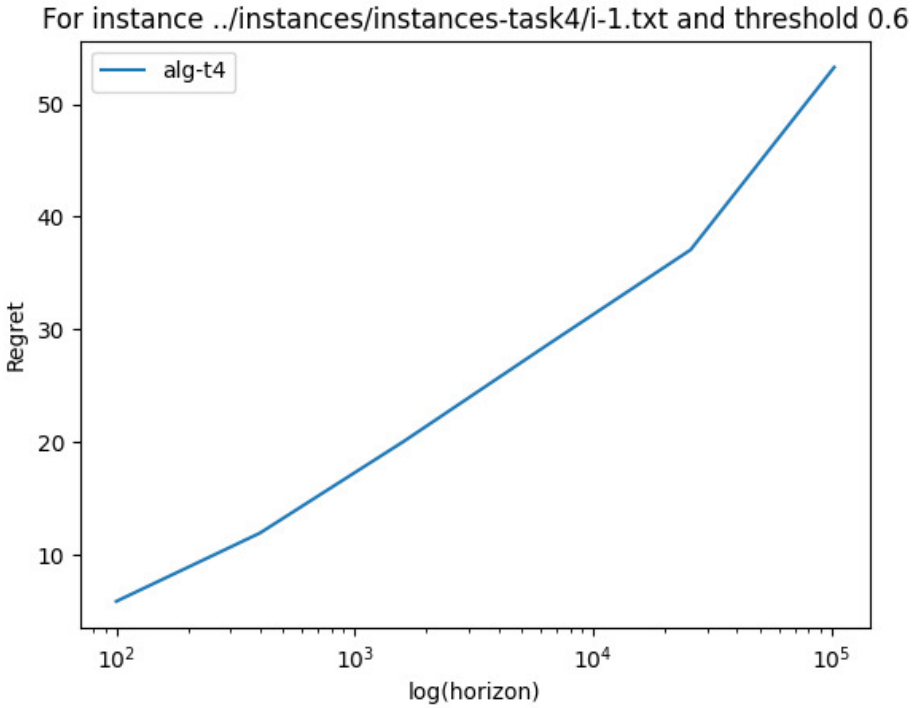
After doing this, we essentially have a binary bandit instance. Now, suppose we know the true probabilities of each arm. Technically, we can obtain the maximum number of HIGHS on **average** by pulling the arm with the maximum probability. Since we can't do this, we can resort to the algorithms we learnt in class. The algorithms we learned in class try to minimise the regret. This is essentially maximising the average reward! Why? Because, Regret = Tp* - Expected Reward. For any given instance, Tp* is constant. Therefore, we can use any of the algorithms from Task 1 for this task.
I chose the Thompson Sampling algorithm because it gave the lowest regret in task 1. Also, it has a much faster execution time!

To describe the algorithm concretely. For each instance, we segregate the rewards into HIGHS and LOWS and make them 1's and 0's respectively. Then, we convert each arm instance into a Bernoulli distribution whose probability is calculated by summing up the probabilities of the rewards which have a value higher than the threshold.

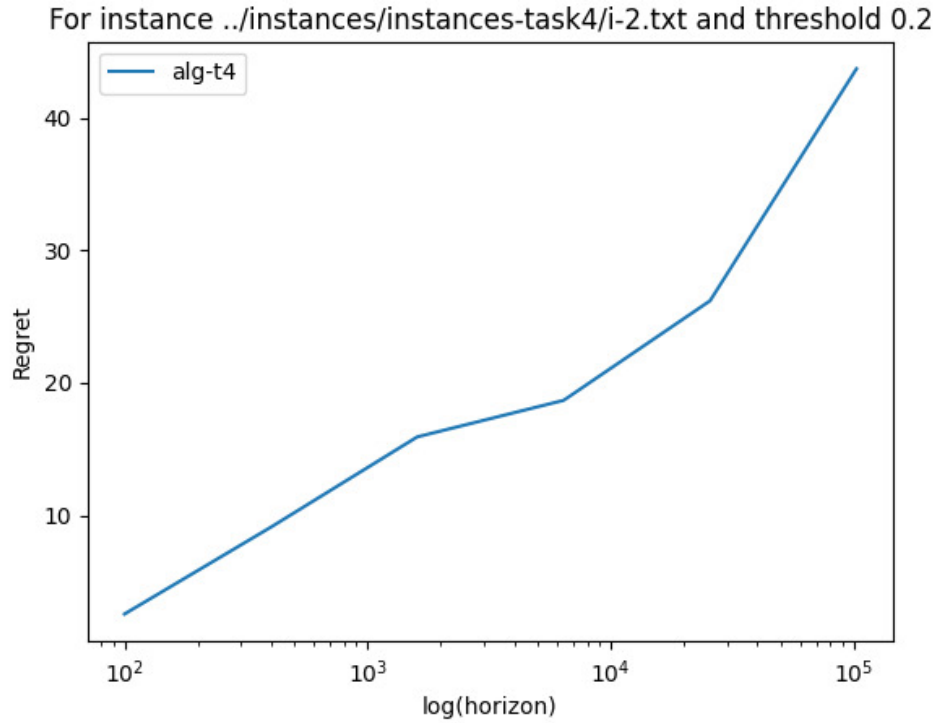
Here are the plots. On the y-axis we have the “HIGH-Regret” As mentioned in the problem statement.



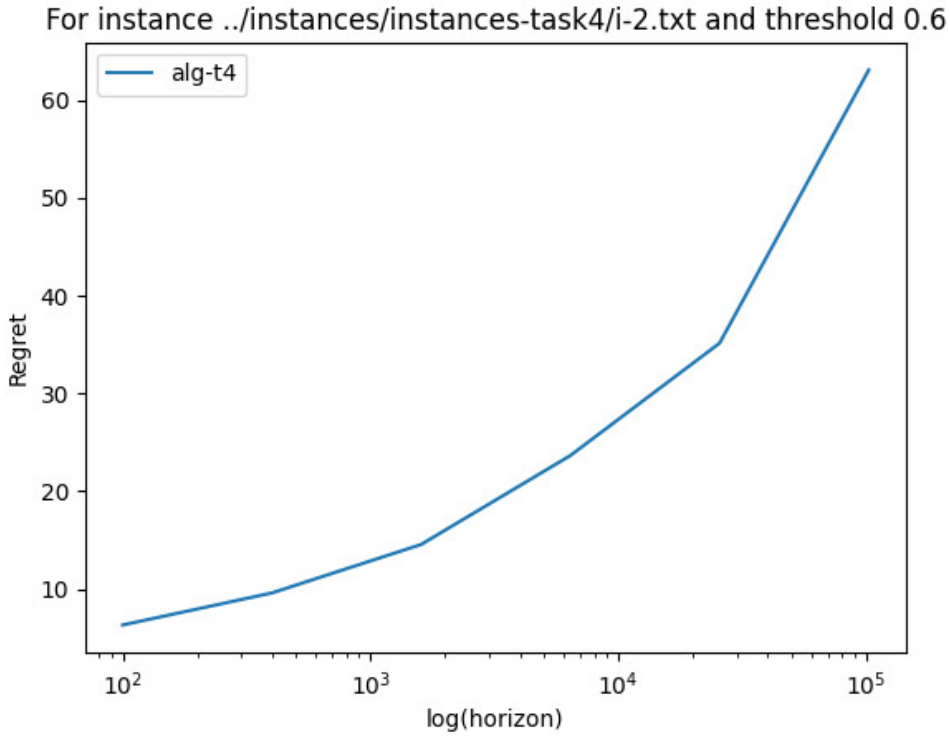
High Regrets for Instance 1 and Threshold 0.2



High Regrets for Instance 1 and Threshold 0.6



High Regrets for Instance 2 and Threshold 0.2



High Regrets for Instance 2 and Threshold 0.2

References are present in References.txt