Prateek Garg
20D070060
prateekg@iitb.ac.in

Programming Assignment 1
CS747: Foundations of Intelligent and Learning Agents, Autumn 2023

2023-09-10

Task 1

The implementations of UCB, KL-UCB, and Thompson Sampling Algorithms.

1.a UCB Algorithm

State Variables: $counts[i]$, $values[i]$ $\forall i$, $total\_counts$

- $counts[i] := c_i$ denotes the number of times, arm i has been pulled.
- $values[i] := v_i$ denotes the empirical average of reward observed from a particular arm.
- $total\_counts := t$ denotes the total number of pulls from the algorithm.

Pull Step: returns an arm to be played

1. Increment t by 1.
2. for i in $[1, .., .n]$, return i if $c_i == 0$
   - Important for well defined ucb as well as c_i is 0, so it means it should be explored first
3. Calculate ucb for each arm i using $ucb_i = v_i + \sqrt{\frac{2\log(t)}{c_i}}$
4. return $i = \arg\max_i ucb_i$

Reward Step: takes $reward := r$, arm index i

1. Increment c_i by 1
2. update v_i using (new) $v_i = \frac{c_i - 1}{c_i} v_i + r/c_i$
   - We could have just save the cumulative rewards, but it might lead to overflow

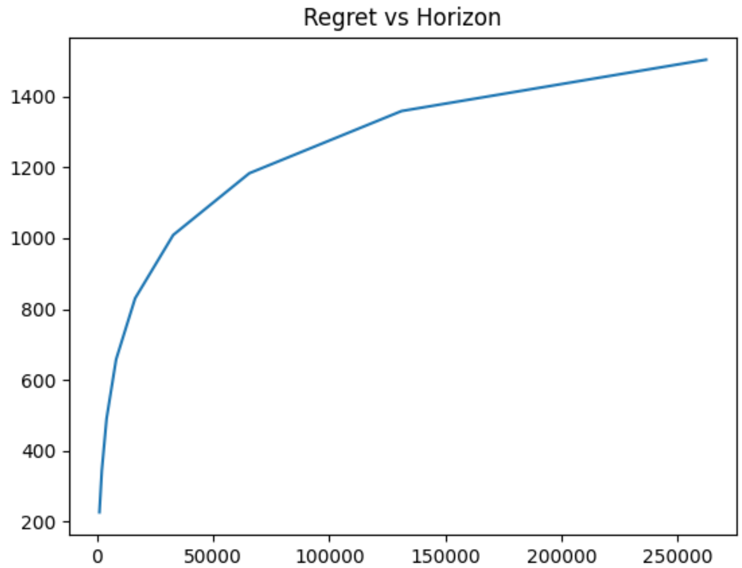


Figure 1: Regret accumulated by UCB Algorithms

1.b KL-UCB Algorithm

Implementation of KL-UCB is same as UCB algorithm, except value of $ucb[i]$ is different and calculated numerically. Here we illustrate the subroutine to calculate $ucb[i]$ for KL-UCB algorithm.

```
def kl_bern(p,q):
    eps = 0.00000001 # For numerical stability
    p[p==0] += eps
    p[p==1] -= eps
    q[q==0] += eps
    q[q==1] -= eps
    return p*np.log(p/q) + (1-p)*np.log((1-p)/(1-q))

def rhs(count,t, c=0):
    te = math.log(t)
    if te == 0: te = 0.00000001 # For numerical stability
    return (te + c*math.log(te))/count

def get_ucb_kl(counts,values,total_counts,prec=0.00000001,c=0, max_iter=30):
    num_arms = len(counts)
    rhss = rhs(counts,total_counts,c)    # calculating rhs for each arm
    q = np.zeros(num_arms)               # placeholder for solutions
    l = np.zeros(num_arms)
    l+= values                           # Lower estimate of the solutions
    u = np.ones(num_arms)                # Upper estimate of the solutions

    for _ in range(max_iter):
        q = (u+l)/2               # Candidate Solutions
        kls = kl_bern(values,q)   # KL-div for each arm
        NEG_MASK = kls < rhss     # indices where lower estimate needs to be updated
        l[NEG_MASK] = q[NEG_MASK]                                # updating lower estimate
        u[np.logical_not(NEG_MASK)] = q[np.logical_not(NEG_MASK)] # updating upper estimate
        NOT_UPDATE_MASK = abs(u-l) < prec    # checking if every value is within required precision
        if all(NOT_UPDATE_MASK) : break      # Terminate the loop
    return q
```

- kl_bern function returns kl-divergence between two Bernoulli distributions parameterised by p and q, if p and q are vectors, it does so parallely.
- rhs function returns rhs value for each arm i
- get_ucb_kl returns the ucb value calculated using Newton Raphson method.
  - Has two parameter: prec(precision) and max_iter for Newton Raphson

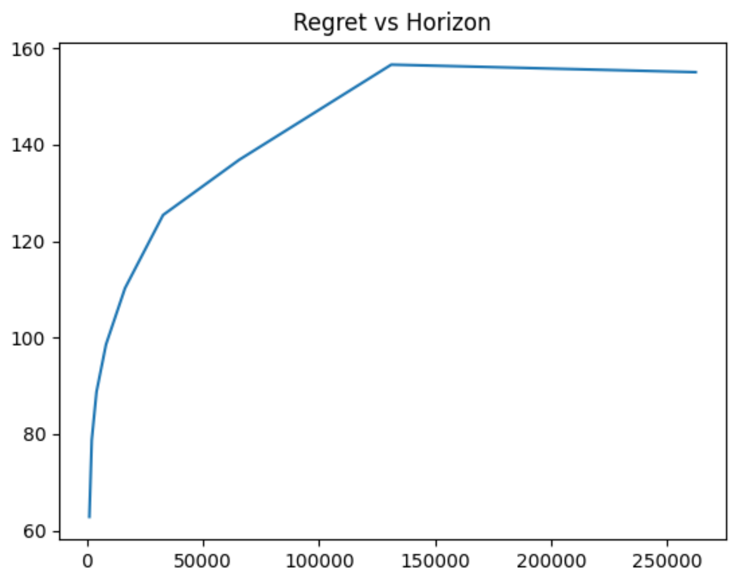


Figure 2: Regret accumulated by KL-UCB Algorithm

1.c Thompson Sampling Algorithm

State Variables: $success[i], failures[i]$

- $success[i] := s_i$ denotes the number of pulls on i^{th} arm gave reward 1

- $failures[i] := f_i$ denotes the number of pulls on i^{th} arm gave reward 0

Pull Step: returns the arm to be played

1. $\mu_i \sim \beta(s_i + 1, f_i + 1) \forall i$
   (a) done parallely using numpy library function
2. return $\arg\max_i \mu_i$

Reward Step: takes $reward := r$, arm index i

1. if $r == 0 : f_i = f_i + 1$
2. if $r == 1 : s_i = s_i + 1$

Figure 3: Regret accumulated by Thompson Sampling Algorithm

Task 2

2.a Differences between means of arms and UCB Algorithm

Regret of UCB algorithm is,

$$R_T = O\left(\sum_{a:p_a \neq p^*} \frac{1}{p^* - p_a} \log(T)\right)$$

for instance $[p1, p2], p_1 = p^*$, it reduces to,

$$R_T = O\left(\frac{1}{p_1 - p_2} \log(T)\right)$$

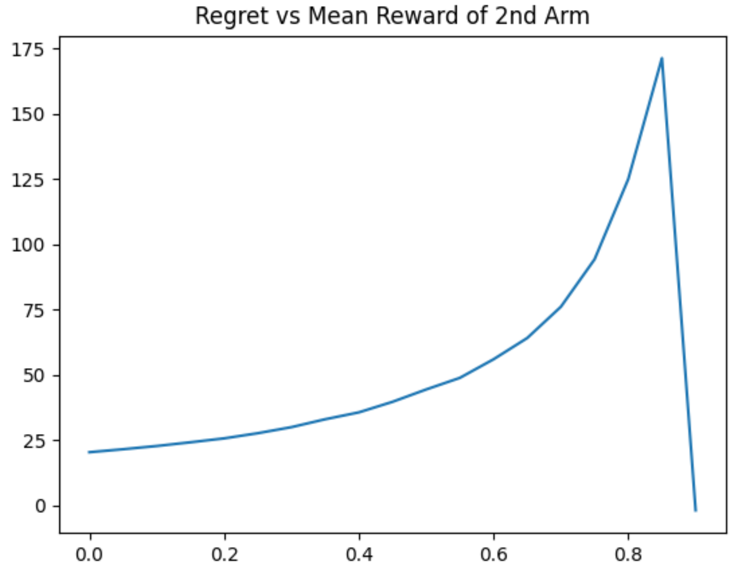
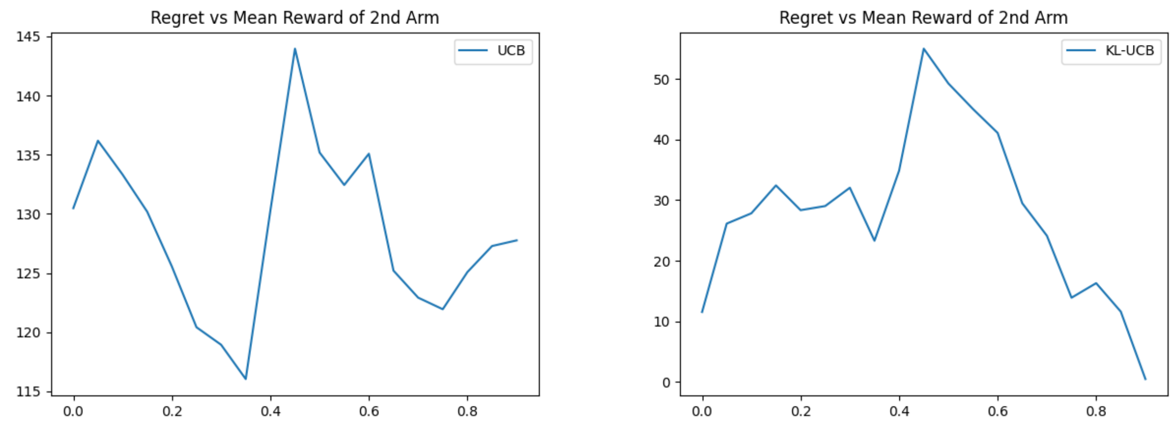


Figure 4: Regret accumulated by UCB Algorithm on instance $[p_1, p_2]$, where $p_1 = 0.9$,p_2 (on x-axis varies 0 to 0.9 (both inclusive) in steps of 0.05 over the horizon of 300000)

Fixing $T = 30000$, we plot the regret of UCB algorithm and indeed we see that the regret increases as p_2 increases but plummets to 0 when $p_1 = p_2 = 0.9$, since now both arms are the optimal arms.

2.b UCB and KL-UCB Algorithms vs difference of means of arms



(a) UCB

(b) KL-UCB

Figure 5: Regret accumulated by two algorithms on instance $[p_1, p_2]$, where $p_1 - p_2 = 0.1$ and p_2(on x-axis varies 0 to 0.9 (both inclusive) in steps of 0.05 over the horizon of 300000)

We see that the regret of UCB algorithm is *almost* constant($130 \pm 11\%$), which is in agreement with the regret bound seen in 2.a as $p_1 - p_2 = 0.1$ is held constant.
Regret of KL-UCB algorithm is,

$$R_T = O\left(\sum_{a:p_a \neq p^*} \frac{p^* - p_a}{KL(p_a, p^*)} \log(T)\right)$$

for instance $[p1, p2], p_1 = p^*$, it reduces to,

$$R_T = O\left(\frac{p_1 - p_2}{KL(p_2, p_1)} \log(T)\right)$$

We see that the regret of KL-UCB algorithm varies a *lot*$(25 \pm 120\%)$, since we have $KL(p_1, p_2)$ term in denominator which changes with every value of p_2 as shown in 6

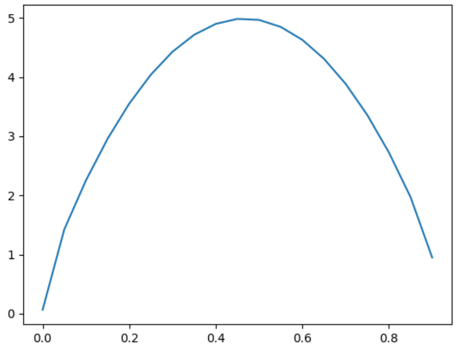


Figure 6: $\frac{p_1 - p_2}{KL(p_2, p_1)}$ as function of p_2, $p_1 - p_2 = 0.1$

This also explains why we observe a peak around $p_2 = 0.45$

Figure 7: Comparison of regret accumulated by two algorithms on instance $[p_1, p_2]$, where $p_1 - p_2 = 0.1$ and p_2(on x-axis varies 0 to 0.9 (both inclusive) in steps of 0.05 over the horizon of 300000)

It is also observed that regret accumulated by KL-UCB is far lesser than the UCB, which is also expected because KL-UCB has tighter bound on regret.

Task 3

After experimenting with almost every algorithm, we found that Thompson Sampling is easiest to adapt and gives lower regrets than any other algorithm. So the goal is to design a Thompson style algorithm to Faulty Bandit case.
Let E be the event that fault occurs, and we $f = P(E)$, $P(r|E) = 0.5$

3.a Modifying the Posterior Update

Posterior in the reward step of Thompson Sampling is,

$$P(x_i|r) = \frac{P(r|x_i) \cdot P(x_i)}{\mathbb{E}_{x_i \sim P(x_i)}\left[P(r|x_i)\right]}$$

$x_i \sim \beta(s_i+1, f_i+1)$, $r|x_i \sim Bern(x_i) \implies P(r=1|x_i) = x_i$ which results in

$$x_i|r \sim \beta(s_i+1+r, f_i+1+1-r)$$

In case of the faulty arm, $P(r|x_i) = 0.5\cdot f + x_i\cdot(1-f)$, we get

$$\begin{aligned} P(x_i|r) &= \frac{P(r|x_i)}{\mathbb{E}_{x_i\sim P(x_i)}\left[P(r|x_i)\right]}\cdot P(x_i) \\ &= \frac{0.5\cdot f + x_i\cdot(1-f)}{0.5\cdot f + \mathbb{E}_{x_i\sim P(x_i)}\left[P(r|x_i,\neg E)\right]\cdot(1-f)}\cdot P(x_i) \end{aligned}$$

assuming $x_i \sim \beta(\alpha_i,\beta_i) \implies \mathbb{E}_{x_i\sim P(x_i)}\left[P(r=1|x_i,\neg E)\right] = \frac{\alpha}{\alpha+\beta}$,

$$\begin{aligned} P(x_i|r=1) &= \frac{0.5\cdot f + x_i\cdot(1-f)}{0.5\cdot f + \frac{\alpha_i}{\alpha_i+\beta_i}\cdot(1-f)}\cdot P(x_i) \\ &= \frac{0.5\cdot f}{0.5\cdot f + \frac{\alpha_i}{\alpha_i+\beta_i}\cdot(1-f)}\cdot P(x_i) \\ &+ \frac{\frac{\alpha_i}{\alpha_i+\beta_i}\cdot(1-f)}{0.5\cdot f + \frac{\alpha_i}{\alpha_i+\beta_i}\cdot(1-f)}\cdot\left(\frac{\alpha_i+\beta_i}{\alpha_i}\cdot x_i\cdot P(x_i)\right) \end{aligned}$$

Looking closely we see that this is a mixture of Beta Distributions,

$$P(x_i|r=1) = \frac{0.5\cdot f}{0.5\cdot f + \frac{\alpha_i}{\alpha_i+\beta_i}\cdot(1-f)}\cdot f_{\alpha_i,\beta_i}(x_i) + \frac{\frac{\alpha_i}{\alpha_i+\beta_i}\cdot(1-f)}{0.5\cdot f + \frac{\alpha_i}{\alpha_i+\beta_i}\cdot(1-f)}\cdot\left(f_{\alpha_i+1,\beta_i}(x_i)\right)$$

where $f_{\alpha,\beta}(x_i) = P(x_i)$, if $x_i \sim \beta(\alpha_i,\beta_i)$ similarly, can be shown that,

$$P(x_i|r=0) = \frac{0.5\cdot f}{0.5\cdot f + \frac{\beta_i}{\alpha_i+\beta_i}\cdot(1-f)}\cdot f_{\alpha_i,\beta_i}(x_i) + \frac{\frac{\beta_i}{\alpha_i+\beta_i}\cdot(1-f)}{0.5\cdot f + \frac{\beta_i}{\alpha_i+\beta_i}\cdot(1-f)}\cdot\left(f_{\alpha_i,\beta_i+1}(x_i)\right)$$

Combining the two we get,

$$P(x_i|r) = \frac{0.5\cdot f}{0.5\cdot f + \frac{\beta_i*(1-r)+\alpha_i*r}{\alpha_i+\beta_i}\cdot(1-f)}\cdot f_{\alpha_i,\beta_i}(x_i) + \frac{\frac{\beta_i*(1-r)+\alpha_i*r}{\alpha_i+\beta_i}\cdot(1-f)}{0.5\cdot f + \frac{\beta_i*(1-r)+\alpha_i*r}{\alpha_i+\beta_i}\cdot(1-f)}\cdot\left(f_{\alpha_i+r,\beta_i+1-r}(x_i)\right)$$

Looking even more closely, we get that,

$$P(x_i|r) = P(E|r)*f_{\alpha_i,\beta_i}(x_i) + P(\neg E|r)*f_{\alpha_i+r,\beta_i+1-r}(x_i)$$

In general it will be intractable to use this update rule, since the next update will result in 3 components and so on. So on t^{th} time we will have 2^t components, thus memory requirement will be $O(2^T)$ where T is horizon, whereas normal Thompson Sampling has $O(1)$ memory requirement.
To tackle this, we observe that this posterior can be interpreted as "Do not update the belief is event E occurs, but update otherwise" and indeed if we were to use the exact posterior for the sampling step, we would be sampling over these t events($E_1, E_2, ..., E_t$) first, choose the corresponding component and then sample from it. So, in our approach, we sample one mixture component and discard the other,

$$e \sim Bern\left(\frac{0.5\cdot f}{0.5\cdot f + \frac{\beta_i*(1-r)+\alpha_i*r}{\alpha_i+\beta_i}\cdot(1-f)}\right)$$

$$P(x_i|r) = \begin{cases} f_{\alpha_i,\beta_i}(x_i), & \text{if } e=1 \\ f_{\alpha_i+r,\beta_i+1-r}(x_i), & \text{otherwise} \end{cases}$$

which would be equivalent to re-using the sample of $E_1, E_2, ..., E_t$ from previous timesteps.
It is worth noting that we take a different prior of $x_i = 0.5\cdot f + (1-f)\cdot y_i, y_i \sim \beta(\alpha_i,\beta_i)$ a similar expression is obtained for the posterior update rule. Here x_i should be thought of as the actually *observed* mean of i^{th} arm and y_i is corresponding *true* mean. This reduces the support of x from $[0,1]$ to $[f*(0.5), 1-0.5*f]$
Also, for this very reason, if we use Thompson Sampling without using this prior information at all, we still achieve very competitive results, since it recovers the observed mean asymptotically.

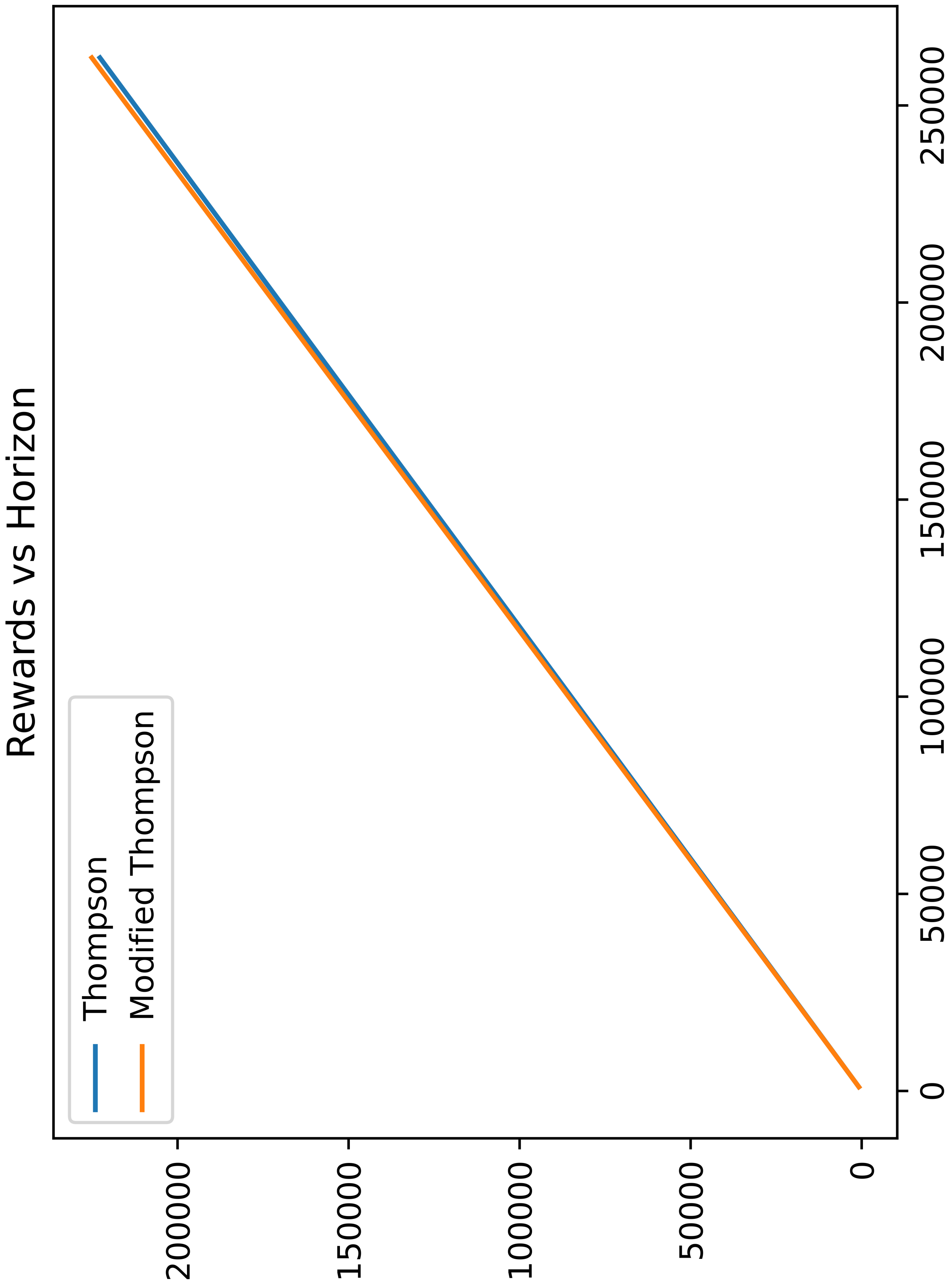


Figure 8: Comparison of rewards accumulated by Thompson and Modified Thompson Sampling, Modified Thomspon marginally performs better.

Task 4

Unlike the case in task 3 where we don't observe if the fault occured or not, here we observe the set which was used, so we can update the *beliefs* accordingly. Let $x_{i,a}$ denote the mean of arm i from set a. Then the expected reward when arm i is pulled is = $\mathbb{E}_a[x_{i,a}]$ which in this case reduces to $\frac{x_{i,0}+x_{i,1}}{2}$.
State Variables: $success[i,a], failures[i,a]$

- $success[i,a] := s_{i,a}$ denotes the number of pulls on i^{th} arm of set a gave reward 1
- $failures[i,a] := f_{i,a}$ denotes the number of pulls on i^{th} arm of set a gave reward 0

Pull Step: returns the arm to be played

1. $\mu_{i,a} \sim \beta(s_{i,a}+1, f_{i,a}+1) \forall i, a$
   (a) done parallely using numpy library function
2. return $\arg\max_i \frac{\mu_{i,0}+\mu_{i,1}}{2}$

Reward Step: takes $reward := r$, arm index i, set pulled a

1. if $r == 0 : f_{i,a} = f_{i,a} + 1$
2. if $r == 1 : s_{i,a} = s_{i,a} + 1$

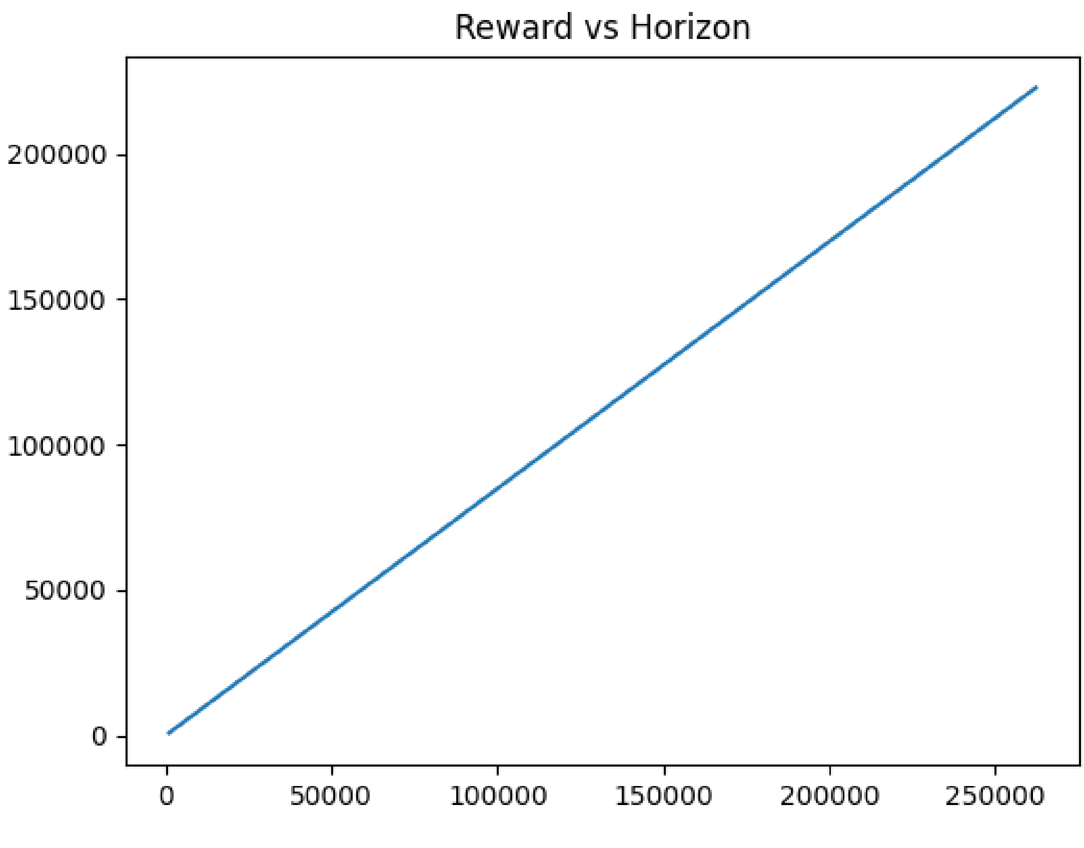


Figure 9: Reward accumulated by our Algorithm