Considering Bandit Algorithms With Sampled Graph Side Observations on Smooth Graphs

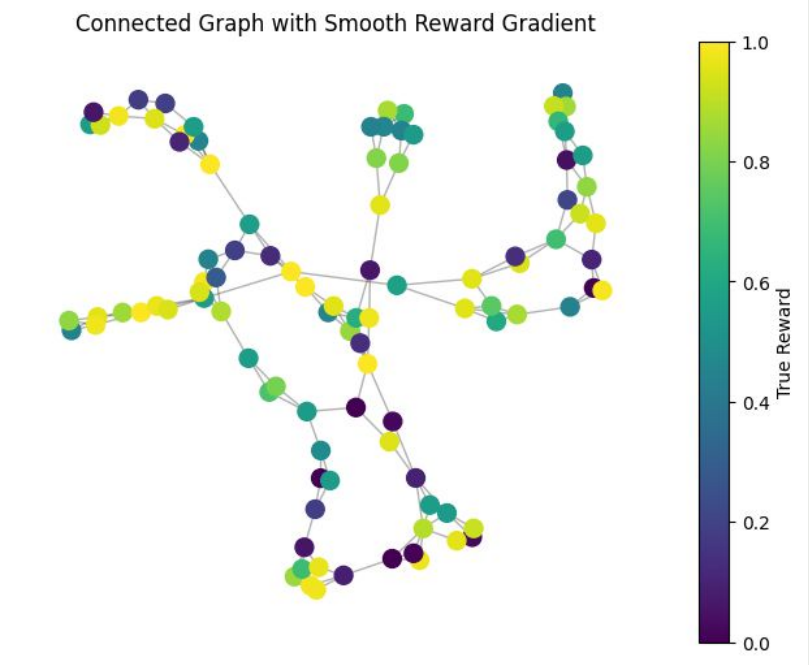
Anup De

Introduction and Key Topics

Graph Bandit Algorithms are are class of online problems in which **actions** can be represented by **nodes.**

The feedback mechanism in this setting allows the learner to gain access to the **reward of the node played**, as well as the **reward of neighboring nodes.**

For this research, I employ the assumption of a **"smooth" graph**, meaning that **neighboring nodes have similar true rewards**.



The Problem Setup

Setting:
Let $G = (V,E)$ be a fixed undirected graph, where each node represents an arm
Each node in V has an expected reward μ between 0 and 1
The reward vector is smooth over the graph such that neighboring nodes have similar expected rewards

Interaction:
At each round, a new graph is sampled from the true instance such that each edge has independent probability p of being included in the sample.
Select a node from the entire graph according to some policy
Observe the noisy reward of the node selected and the noisy rewards of the neighbors of the node selected

Regret:

$$R_T = T \cdot \mu^* - \sum_{t=1}^{T} \mathbb{E}[\mu_{a_t}], \quad \text{where } \mu^* = \max_i \mu_i.$$

Why is This Useful?

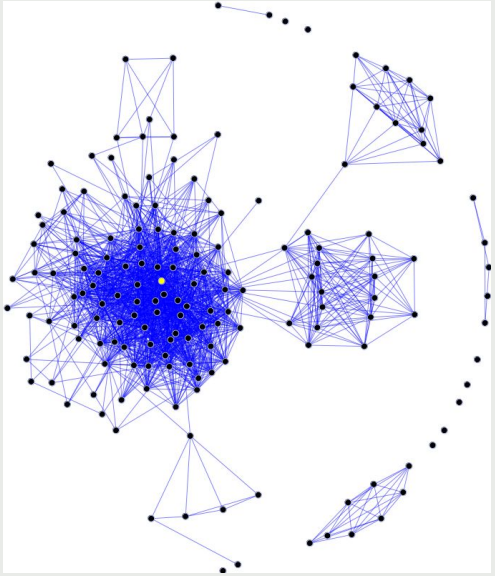
The purpose of this modeling is to increase the applicability of known bandit algorithms. This process will allow them to learn and play on graphs in settings in which they do not have access to full information at each time.

This can also lead to regret bounds that depend on **graph structure and sampling probability**, which may prove to be stronger in context.

I will also look at one method of **reward estimation** in this smooth setting, and this project can serve as an evaluation on that method.

Why is This Useful?

Consider the following scenario:
Edges of a social network are “active” only at certain random times (perhaps when users are active online). It is the task of the algorithm to choose a person in the network to receive an advertising campaign (any person, whether active or not) to earn some reward. ***The algorithm’s ability to estimate is now dependent on the sample provided.***



Plenty of other scenarios model this behavior, such as neurons which stochastically stimulate one another to particles which activate at random times.

Note that in all of these cases, the “smoothness” assumption likely holds. Ad campaigns recommended to people socially connected will have similar reward expectations.

Understanding Smoothness

Let A be the adjacency matrix of our true graph, and let D be the degree matrix of that graph. Define the combinatorial Laplacian matrix as $L = D - A$. I consider the smoothness definition as when the true rewards of neighboring nodes i, j are within some constant S, as in $\sum_{(i,j)\in E}(\mu_i - \mu_j)^2 \leq S$.

$$
\begin{aligned}
\sum_{(i,j)\in E} (\mu_i - \mu_j)^2 &= \sum_{(i,j)\in E} \mu_i^2 + \mu_j^2 - 2\mu_i\mu_j \\
&= \sum_i \mu_i^2 \cdot \deg(i) + \sum_j \mu_j^2 \cdot \deg(j) - 2 \sum_{(i,j)\in E} \mu_i\mu_j \\
&= 2\sum_i \deg(i)\mu_i^2 - 2 \sum_{(i,j)\in E} \mu_i\mu_j \\
&= \sum_i \deg(i)\mu_i^2 - \sum_{(i,j)\in E} \mu_i\mu_j \\
&= \mu^\top D\mu - \mu^\top A\mu \\
&= \mu^\top L\mu \leq S
\end{aligned}
$$

Smoothed Estimator

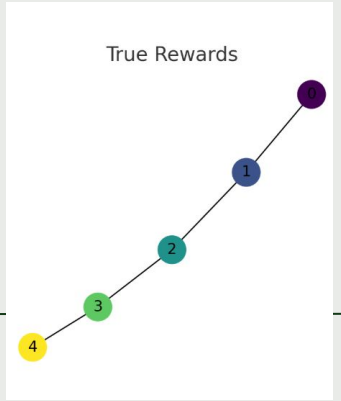
In many ways, this project is just an evaluation on the smooth estimator created below.

(1) $$(I + \lambda L) \cdot \tilde{\mu} = \hat{\mu}$$

For an inputted reward estimate vector ũ, and the Laplacian of the new learned graph, a new reward estimate is computed, based on the regularization parameter lambda.

Take the following example of true rewards for a graph that has so far been learned as 0-1-2-3-4:

- Node 0 → 0.0
- Node 1 → 0.25
- Node 2 → 0.5
- Node 3 → 0.75
- Node 4 → 1.0



Smoothed Estimator

This makes the Laplacian:

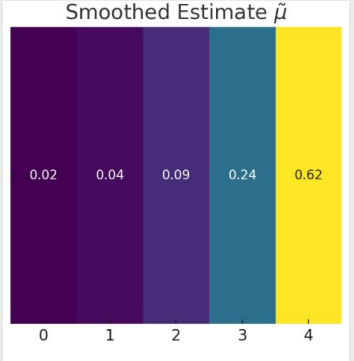
$$L = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Algorithm 3 Smooth Estimator

Input: Learned Graph $G_{learned}$ and current reward estimates

Create Laplacian Matrix, L, for $G_{learned}$
Create Smoothing Matrix, $A = I + \lambda L$
New reward estimate $= A^{-1}$(reward estimate inputted)

return New reward estimate



Our current reward estimate (for sake of simplicity) is the vector [0,0,0,0,1]

Applying lambda = 1, we get the following smooth estimate from the equation 1:

[0.02, 0.04, 0.09, 0.24, 0.62]

Some of those original rewards associated with node 4 have spread through the graph's edges, creating better reward estimates

Epsilon-Greedy Algorithm (Base)

4.2 Epsilon-Greedy with Graph Side Observations Algorithm

Algorithm 1 Epsilon-Greedy with Graph Side Observations

Input: all nodes K, horizon T, sampling probability p, noise standard deviation σ)

Initialize reward estimates $\hat{\mu}_i$, pull counts $N_t(i)$ for all nodes i, and total regret

for each round t in T **do**
  Create $G_t = (V, E_t)$ where E_t includes each edge $(i, j) \in E$ with probability p
  With probability $1 - \epsilon$, $a_t = \arg\max_{i \in K} \hat{\mu}_i$
  Otherwise select a_t as random node $i \in K$
  Receive reward $\mu_{a_t} + N(0, \sigma^2)$
  Update $\hat{\mu}_{a_t}$
  Add $\mu_\star - \mu_{a_t}$ to total regret
  For every neighbor, j. of $a_t \in G_t$, observe $\mu_j + N(0, \sigma^2)$ and update $\hat{\mu}_j$.
end for

return total regret

This epsilon-greedy algorithm does not bother making considerations to the sample G_t and therefore its Regret bound does not depend on p. It simply takes the opportunity to learn more and provide for more accurate estimates of true rewards without incurring regret of suboptimal nodes.

Epsilon-Greedy Algorithm (SE)

Algorithm 1 Epsilon-Greedy with Graph Side Observations - SE

Input: all nodes K, horizon T, sampling probability p, noise standard deviation σ

Initialize reward estimates, pull counts $N_t(i)$ for all nodes i, and total regret

Initialize empty graph $G_{learned}$ with K disconnected nodes

for each round t in T **do**
- Create $G_t = (V, E_t)$ where E_t includes each edge $(i, j) \in E$ with probability p
- Update $G_{learned}$ with edges sampled
- Compute Smooth Estimators, $\hat{\mu}_i$ from reward estimates
- With probability $1 - \epsilon$, $a_t = \arg\max_{i \in K} \hat{\mu}_i$
- Otherwise select a_t as random node $i \in K$
- Receive reward $\mu_{a_t} + N(0, \sigma^2)$
- Update reward estimates and pull counts
- Add $\mu_\star - \mu_{a_t}$ to total regret
- For every $(a_t, j) \in E_t$, observe $\mu_j + N(0, \sigma^2)$, update reward estimates and pull counts.

end for

return total regret

This learner is the same as before, but now leverages the "smooth estimator" function mentioned before to serve as reward estimators.

Lemma For Side Observation Setting

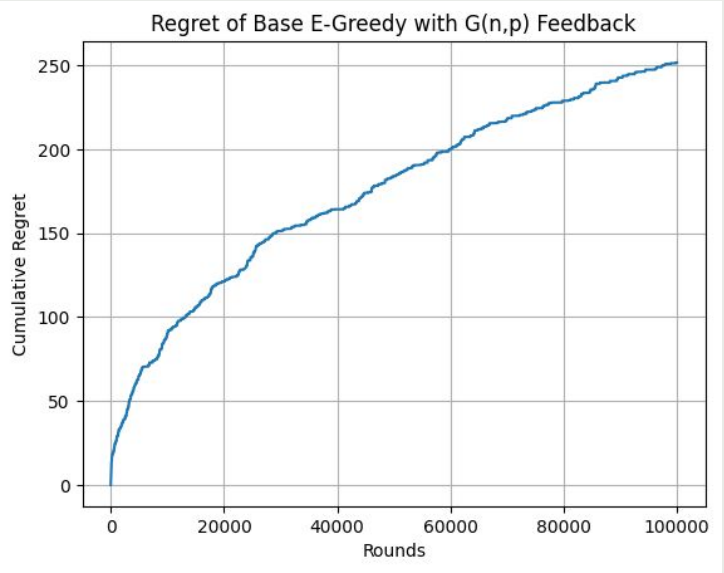
Since the algorithms does not incur regret every time arm *i* is observed, but can instead also be observed from a side observation, we can rewrite this as (for $N_i(t)$ times the arm was actually selected considering $\mathcal{N}(i)$ as the neighbors of *i*)

$$\tilde{N}_i(t) = N_i(t)\left(1 + p \cdot \sum_{j \in \mathcal{N}(i)} \mathbb{E}[\text{times arm j is pulled}(t)]\right)$$

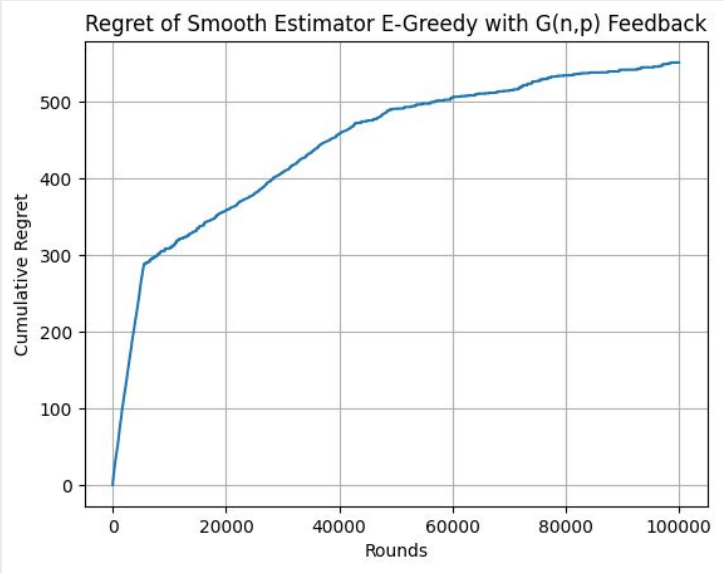
This summation is just the degree of node *i*. Whenever $N_i(t)$ would normally appear in a proof, we can replace by solving for the true $N_i(t)$, now putting regret in terms of *p* and *d*.

Epsilon-Greedy Empirical Results

Without Smooth Estimator:



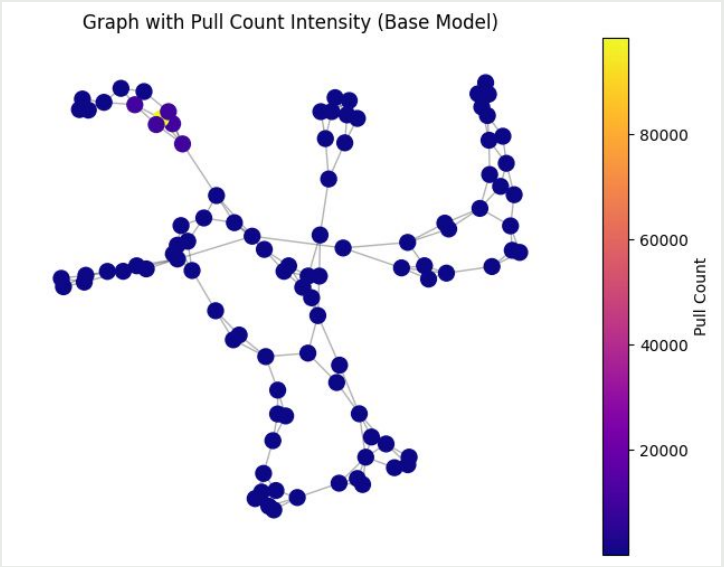
With Smooth Estimator:



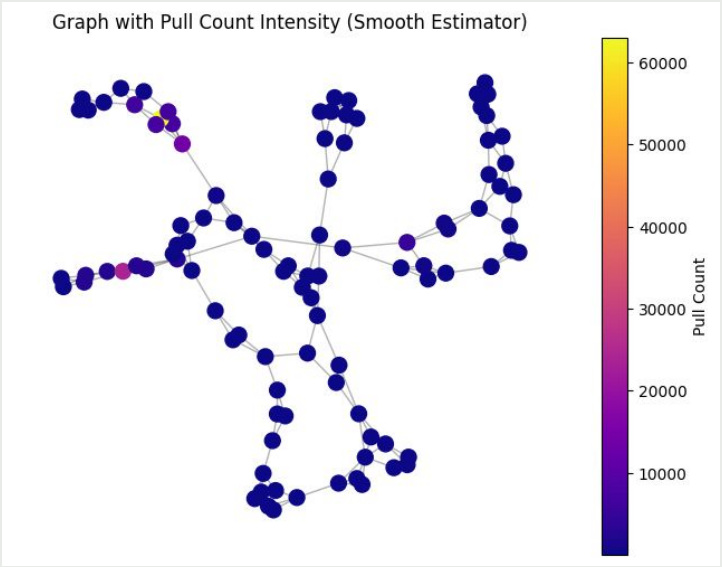
p = 0.1, noise st.dev = 0.1, lambda = 1, 100 nodes, 10^5 rounds

Epsilon-Greedy Empirical Results

Without Smooth Estimator:



With Smooth Estimator:



p = 0.1, noise st.dev = 0.1, lambda = 1, 100 nodes, 10^5 rounds

UCB-Base

Algorithm 2 UCB with Graph Side Observations

Input: all nodes K, horizon T, sampling probability p, noise standard deviation σ

Initialize reward estimates $\hat{\mu}_i$, pull counts $N_t(i)$ for all nodes i, and total regret

for each round t in T **do**
  Create $G_t = (V, E_t)$ where E_t includes each edge $(i, j) \in E$ with probability p
  Compute bonus terms and UCB values for all i
  Select arm $a_t = \arg\max_{i \in K} \mathrm{UCB}_t(i)$
  Receive reward $\mu_{a_t} + N(0, \sigma^2)$
  Update reward estimates and pull counts
  Add $\mu_\star - \mu_{a_t}$ to total regret
  For every $(a_t, j) \in E_t$, observe $\mu_j + N(0, \sigma^2)$, update reward estimates and pull counts.
end for

return total regret

This UCB algorithm does not bother making considerations to the sample G_t and therefore its Regret bound does not depend on p. It simply takes the opportunity to learn more and provide for more accurate estimates of true rewards without incurring regret of suboptimal nodes.

UCB-SE

Algorithm 2 UCB with Graph Side Observations

Input: all nodes K, horizon T, sampling probability p, noise standard deviation σ)

Initialize reward estimates $\hat{\mu}_i$, pull counts $N_t(i)$ for all nodes i, and total regret

Initialize empty graph $G_{learned}$ with K disconnected nodes

for each round t in T **do**
- Create $G_t = (V, E_t)$ where E_t includes each edge $(i, j) \in E$ with probability p
- Update $G_{learned}$ with edges sampled
- Compute Smooth Estimates of $\hat{\mu}_t(i)$
- Compute bonus terms and UCB values for all i
- Select arm $a_t = \arg\max_{i \in K} \text{UCB}_t(i)$
- Receive reward $\mu_{a_t} + N(0, \sigma^2)$
- Update $\hat{\mu}_{a_t}$ and pull counts
- Add $\mu_\star - \mu_{a_t}$ to total regret
- For every neighbor, j. of $a_t \in G_t$, observe $\mu_j + N(0, \sigma^2)$ and update $\hat{\mu}_j$ and pull counts.

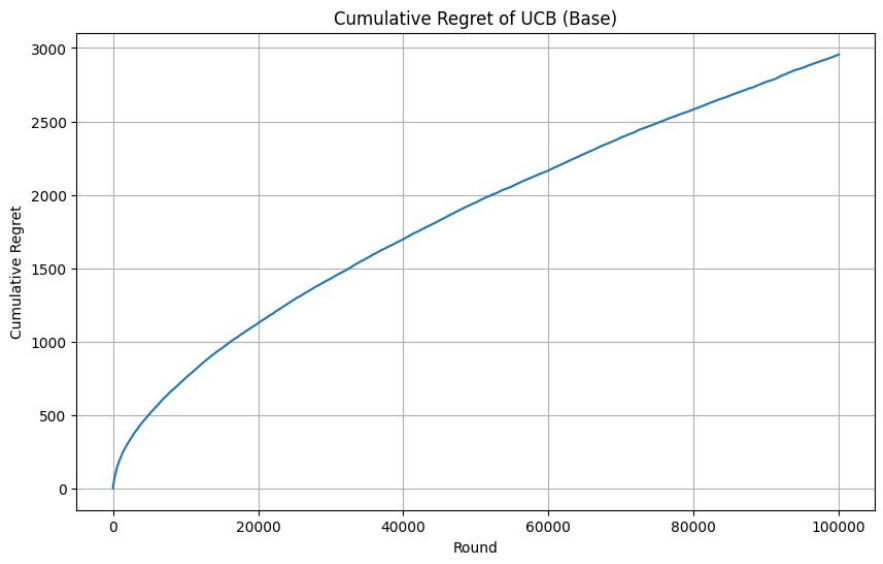
end for

return total regret

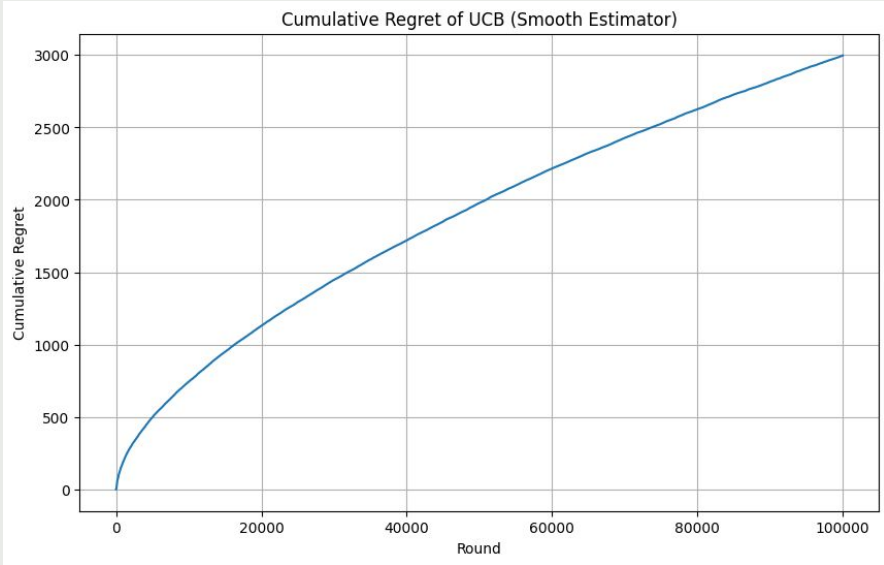
The UCB Algorithm in this setting will also compute an estimate of the rewards seen, but this time I employ the use of a **smoothed estimator** and then perform UCB. The bonus term can still be in terms of the total number of times the node was *observed*. As the per the lemma earlier, we can remove instances in which the node was *observed* and not *played*, but for just the bonus term, the estimate accuracy is dependent on all *observations*.

UCB Empirical Results

Without Smooth Estimator:



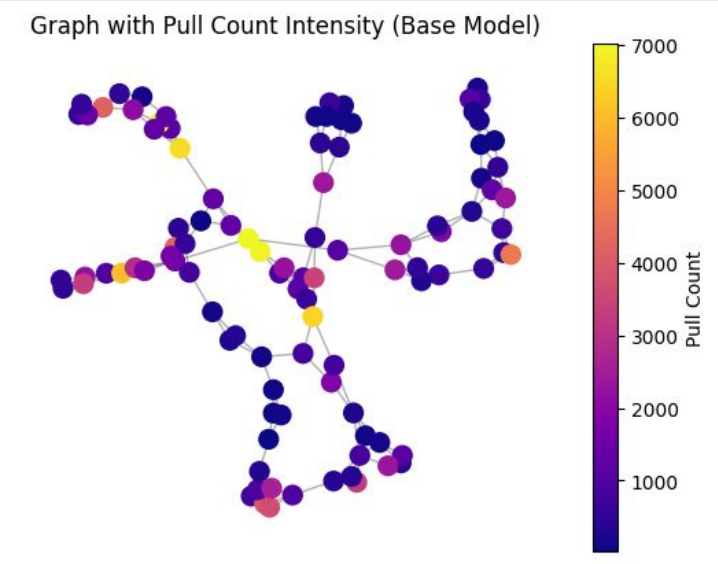
With Smooth Estimator:



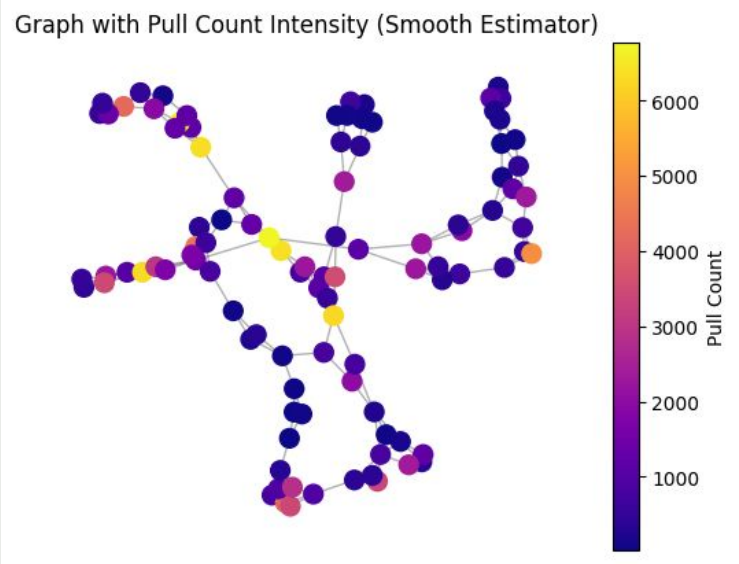
p = 0.1, noise st.dev = 0.1, lambda = 1, 100 nodes, 10^5 rounds

UCB Empirical Results

Without Smooth Estimator:



With Smooth Estimator:



p = 0.1, noise st.dev = 0.1, lambda = 1, 100 nodes, 10^5 rounds

Regret Bound Intuition for Base Models

Starting at the Hoeffding concentration bound, the estimate in the base model setting is within a beta concentration dependent on the number of times the arm has been observed. From side observation lemma, we can solve for the number of times the arm actually incurs regret to be **log(t) / (1 + pd)(delta)**.
The final regret is still in terms of the exploration phase, returning epsilon(T) regret. Even though this is set equal to the smaller arm being actually pulled, term the regret bound is asymptotically the same

$$\hat{\mu}_i - \mu_i \leq 2\sqrt{\frac{2K \log t}{\tilde{N}_i(t)}}$$

This implies that arm i will be chosen at time t only if:

$$\mu_\star - \mu_i \leq 2\sqrt{\frac{2K \log t}{\tilde{N}_i(t)}}$$

$$\Delta_i \leq 2\sqrt{\frac{2K \log t}{\tilde{N}_i(t)}}$$

$$\implies \tilde{N}_i(t) \leq \frac{\log(t)}{\Delta_i^2}$$

Regret Bound Intuition for SE Models

Under Smoothed Estimator Models, the concentration bound is different. It will be in terms of the variance of the estimator and the bias of the estimator. The estimator is biased because it purposefully shifts rewards onto neighbors.
The variance is the same concentration that was from the base models from Hoeffding (which worked on unbiased estimators).
The bias at time *t*, is dependent on the current Laplacian of the current known graph and the true mean rewards vector that has been going out. From Cauchy-Schwarz the following intuition is created for the bias:

$$\|\mu - \hat{\mu}\|_2 \leq \lambda \|L\mu\|_2 \leq \lambda \sqrt{\lambda \cdot \mu^\top L \mu} \leq \sqrt{\lambda} S$$

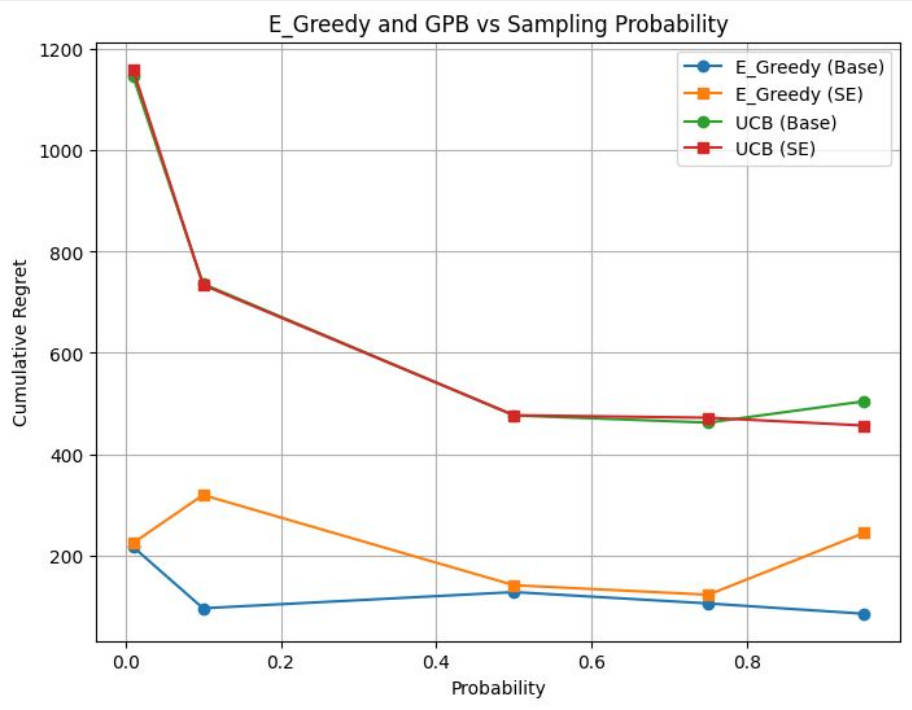
Following the same lemma as before, the number of times arm *i* is observed is:

$$\tilde{N}_i(t) \leq \frac{2\log(K/\delta)}{\left(\frac{\Delta_i}{2} - \sqrt{S\lambda}\right)^2}$$

The regret-contributing pulls are bounded by:

$$\implies N_i(t) \leq \frac{2\log(K/\delta)}{\left(\frac{\Delta_i}{2} - \sqrt{S\lambda}\right)^2 (1 + p d_i)}$$

Varying Probability



Probability seems to have an impact on the UCB modeling. Sampling probability seems to be in the regret, such that the edges of the graph get learned faster, the true rewards are also learned faster. This likely means the smooth estimator needs constant sampling to be accurate.

T = 10^4

Further Research

Smooth setting makes Epsilon-Greedy very valuable
- Why?

The Smooth Estimator for Epsilon-Greedy did worse than its base model
- Even in the smooth setting, the mean realized rewards is valuable.
- Perhaps this was because of the imperfections in the smooth setting
- Perhaps this was because of small variance, making leveraging neighborhoods less rewarding

The Smooth Estimator did not have an effect on UCB
- Possibly means the bonus term was the dominating factor in picking a node rather than the estimator

There may be no need in the Smooth Estimator to “smoothen” out learned rewards, as side observations functions in this capacity.

References

Laplacian Regularized Graph Bandits: Algorithms and Theoretical Analysis (2019); Yang et.al.; https://arxiv.org/pdf/1907.05632

Introduction to Multi-armed Bandits Lecture (2020); Panageas; https://panageas.github.io/_pages/L09_LectureNotes.pdf

Spectral Bandits for Smooth Graph Functions (2014); Valkos et.al.; https://proceedings.mlr.press/v32/valko14.pdf

Dynamic clustering of contextual multi-armed bandits (2014); Nguyen and Lauw; https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?referer=&httpsredir=1&article=3328&context=sis_research