Considering Bandit Algorithms With Sampled Graph Side Observations on Smooth Graphs

Anup De

1 Introduction

I am considering the known multi-armed bandit model of problems for online learning. The graphical model setup of this problem allows for a standardized way for online learning algorithms to learn information and losses not only about the actions taken by the model (Wang et al., (2020)), but learning based on *graphical side observations*. While this feedback mechanism has continued pathways of study, I am focusing on how the effect of an algorithm's limited knowledge of the graph impacts learning. Under the smooth graph setup, (Valko et al., (2014)), I introduce graph sampling, and then follow with evaluations of known bandit algorithms' performance (Epsilon-Greedy and UCB) under the assumption that they only have the ability to learn and make key estimations from the current sample.

Consider the online learning algorithm that only has access to streaming data, which may represent only a subsection of a larger graph. Consider neuron connections, in which neuron connections fire stochastically. The model must choose exploiting a known node or exploring a potentially highly influential recent sample, and therefore must compute accurate estimates for the true rewards under the graph conditions. Consider a similar setup for a social network, where only certain users are active at a given time, representing the model's current sample of a larger graph.[1] In this instance, a model which may only select one person (to perhaps send its ad campaign or media to), must consider learning about the current sample, or exploiting the graph the algorithm has been building until the current time, while considering these decisions based on current estimates.

From this research, it can be shown that known algorithms can be adapted for the novel graph sampling feedback mechanism, as well as intuition for their adapted regret bounds. It is also seen that this mechanism setup can model true applicable online learning problems, the problem instances where the entirety of the initial setup is not known. This will also setup avenues of research regarding how to evaluate estimates of rewards based on smooth graphs.

2 Background

The first consideration is the smooth graph setup. Problems in this space define a smooth graph function as a function on a graph that returns similar values on neighboring nodes (Valko et al., (2014)). In the bandit case, each node represents a possible action, or selection, that returns some

[1] As opposed to the another potential recommendation engine bandit setup where nodes are nodes are actions (i.e. media, articles, products) to be recommended, and are connected based on similar features, this will consider nodes as individuals in a social network

noisy reward. One of the ways to employ this smooth graph setup is (Wu et al., (2016)) having neighbor similarity reported as edge weights. In this setting, the users are nodes and who each have the ability to present reward based on an action chosen by the algorithm, which is not be the bandit setup I presented above. Other methods of the smooth graph include clustering with the goal of reducing realistic dimensionality of the original graph (Yang et al., (2019)). While both of these is extraordinarily applicable to recommendation engine settings, they do not model the same problem goals as I do. I will consider the graph as smooth, yet connected without considering edge weights and maintaining the bandit problem definition as having nodes as actions. Once I create my definition of smoothness in our graph in section 4 of this paper, I will draw inspiration from the Laplacian found here (Yang et al., (2019) in order to attribute rewards earned from selections onto those of neighbors and create the "smooth estimator". While my feedback mechanism does allow me to directly learn those neighbors rewards, the Laplacian definition allows me to estimate those initially, a need that comes about because of the sampling.

3 Problem Setup

Setting:
Let $G = (V, E)$, $|V| = K$, be a fixed undirected graph where each node represents an arm. Each node $i \in V$ has an unknown expected reward $\mu_i \in [0, 1]$. The reward vector $\mu \in \mathbb{R}^K$ is *smooth* over the graph such that $\mu^\top L \mu \leq S$ where L is the (combinatorial) Laplacian of G, and S is a smoothness constant.

Interaction Process:

At each round $t = 1, \ldots, T$:

1. Sample a subgraph $G_t = (V, E_t) \sim \mathcal{G}(n = K, p)$ by independently including each edge of G with probability p.
2. Select a node $a_t \in V$ using a policy π_t.
3. Observe:
   - The reward $r_t = \mu_{a_t} + \eta_t$, where $\eta_t \sim \mathcal{N}(0, \sigma^2)$.
   - The noisy rewards of all neighbors of a_t in G_t: for each $j \in N_t(a_t)$, observe

$$r_{t,j} = \mu_j + \eta_{t,j}.$$

Goal:

Minimize expected **cumulative regret**:

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

4 The Smooth Setting

4.1 The Setup Conditions

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 $\sum_{(i,j)\in E}(\mu_i - \mu_j)^2 \leq S$.

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

In practice, I use the known method of a smooth graph function being a linear combination of the Laplacian's smallest eigenvectors (Belkin et al., (2004)).

4.2 Smooth Estimation Algorithm

From that above derivation, it is seen that the reward vector can be estimated by the the Laplacian matrix. I add the identity matrix to ensure some non-zero entries and therefore then implement a regularization parameter on the Laplacian part of $I + L$, to parameterize how much trust is placed in the Laplacian. The Laplacian is based on the $G_{learned}$, which is the graph created by all the edges sampled until the current time.

Algorithm 1 Smooth Estimator

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

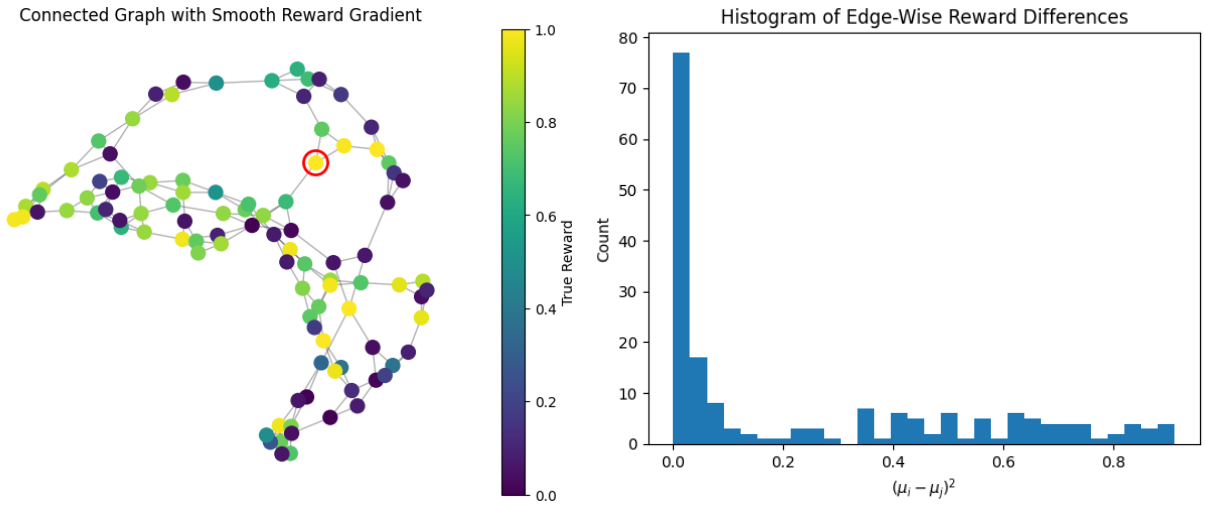
Create Laplacian Matrix, L, for $G_{learned}$
Create Smoothing Matrix, $A = I + \lambda L$
New reward estimate = A^{-1}(reward estimate inputted)

return New reward estimate

5 Empirical Testing

I will proceed with the consideration of the algorithms Epsilon Greedy and UCB, under both baseline graph sampling and feedback mechanism conditions, and under conditions of smooth es-

timators. For the empirical testing, the following truth graph will be used, with lighter nodes representing higher rewards. The subsequent histogram represents the squared reward difference between all i, j pairs in the set of edges. The apparent smoothness conditions are apparent in these models. Parameter $S = 0.5$ holds here and the arm with the highest reward is circled in red.



(a) Graph with True Rewards

(b) Histogram of $(\mu_i - \mu_j)^2 \quad \forall (i, j) \in E$

Figure 1: E-Greedy Base Model

6 Epsilon-Greedy on Smooth Graphs

6.1 Policy Analysis

I will evaluate the known epsilon greedy algorithm on the graphical feedback model, first by estimating true rewards μ_i with the average realized rewards $\hat{\mu}_i$. After creating the sampled graph at round t, the algorithm will select (with probability 1-ϵ) the node with the highest mean rewards (including noise) so far realized by the algorithm's choices. With probability ϵ it shall select any of the K nodes. It still will learn the rewards and update its estimates based on the interaction process outlined above. Global smooth graph G has already been formed with K nodes and E edges. Therefore, the policy of the base epsilon-greedy algorithm will make no reference to the sampled graph, and only use it for extra observations. Then I will model the same algorithm, but this time estimating the rewards using the Smooth Estimate. The purpose of this, to model some learning from the gained sample. Comparing these two models will allow me to evaluate the viability of the smooth estimator.

6.2 Epsilon-Greedy with Graph Side Observations Base Algorithm

Algorithm 2 Epsilon-Greedy with Graph Side Observations - **Base**

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

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
  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

6.3 Epsilon-Greedy with Graph Side Observations Smooth Estimate Algorithm

Algorithm 3 Epsilon-Greedy with Graph Side Observations - **Smooth**

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

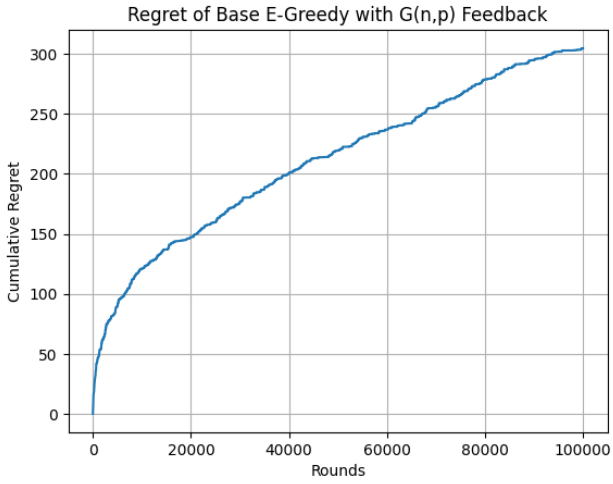
Initialize reward estimates, pull counts $N_t(i)$ for all nodes i, and total regret

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

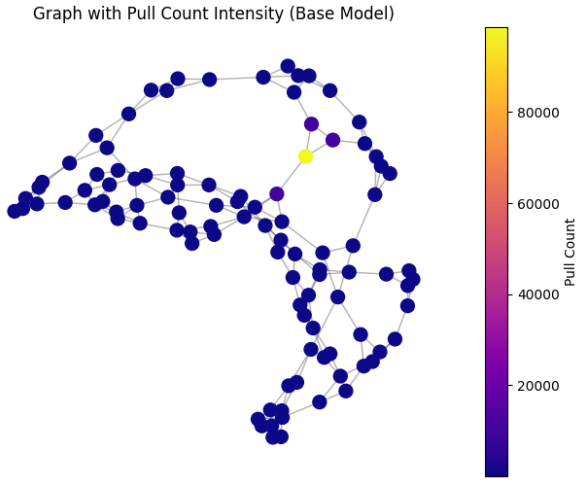
for each round t in T **do**
  Create $G_t = (V, E_t)$ where E_t includes each edge $(i, j) \in E$ with probability p
  Update $G_{learned}$ with edges sampled
  Compute Smooth Estimators, $\hat{\mu}_i$ from reward estimates
  With probability $1 - \epsilon$, $a_t = \arg\max_{i \in K} \hat{\mu}_i$
  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

6.4 Epsilon Greedy - Empirical Test Results[2]

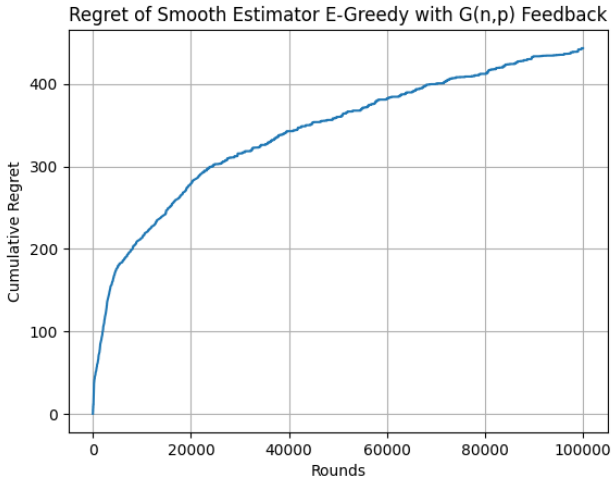


(a) Cumulative Regret

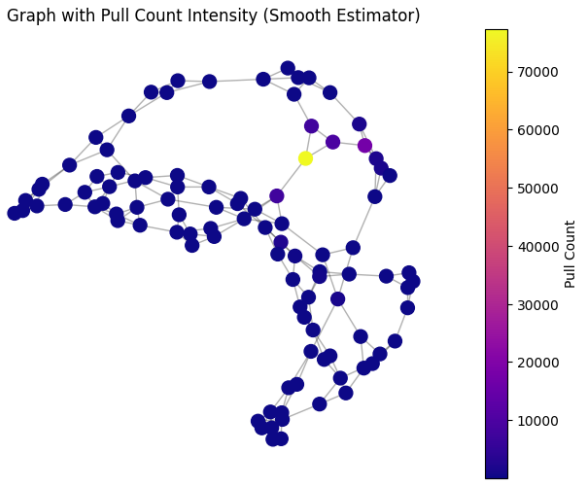


(b) Heat Map of Played Nodes

Figure 2: E-Greedy Base Model



(a) Cumulative Regret



(b) Heat Map of Played Nodes

Figure 3: E-Greedy Smooth Model

Experimental analysis shows that the achieved regret is worse for the smoothed estimator version and learning seems to occur at similar rates at later timesteps. This is motivated by the pull intensity of nodes, which shows that both algorithms were fooled by nearby nodes (highlighting the smooth conditions). The Smooth Estimator version was fooled with higher intensity by these neighboring nodes and did not get to pull the true best arm with the same intensity.

[2]Under the following parameter conditions ($T = 10^5, p = 0.1, \sigma = 0.1, \lambda = 0.1, K = 100$)

7 UCB on Smooth Graphs

7.1 Policy Analysis

The policy of UCB will be in the baseline setting to calculate reward estimates by taking averages and then adding standard UCB bonus terms. Like its epsilon-greedy counterpart, it does not have much care for the sample at the current time and it does not factor into decision making, but only reveals side observations. The modified UCB for the smooth estimate will utilize the same smooth estimate function and then add the UCB standard bonus term. The bonus term will be in terms of the $\tilde{N}_i(t)$, or the number of times the node was observed (increasing confidence in the estimate).

7.2 UCB with Graph Side Base Observations

Algorithm 4 UCB with Graph Side Observations - **Base**

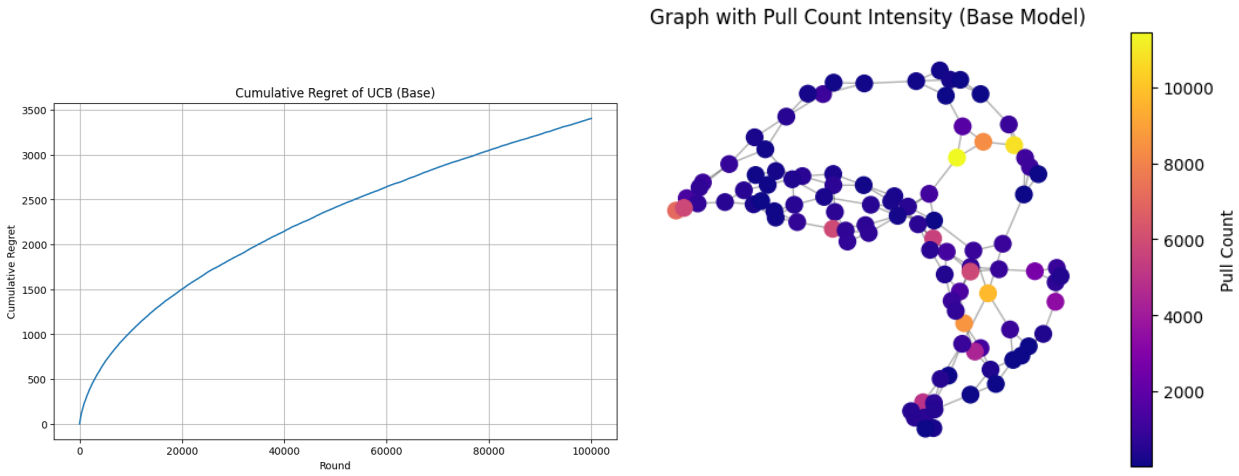
Input: all nodes K, horizon T, sampling probability p, noise standard deviation σ
Initialize reward estimates $\hat{\mu}_i$, pull counts $N_t(i)$ for all nodes i, and total regret
for each round t in T **do**
  Create $G_t = (V, E_t)$ where E_t includes each edge $(i, j) \in E$ with probability p
  Compute bonus terms and UCB values for all i
  Select arm $a_t = \arg\max_{i \in K} \text{UCB}_t(i)$
  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

7.3 UCB with Graph Side Smooth Estimate Observations - Smooth

Algorithm 5 UCB with Graph Side Observations

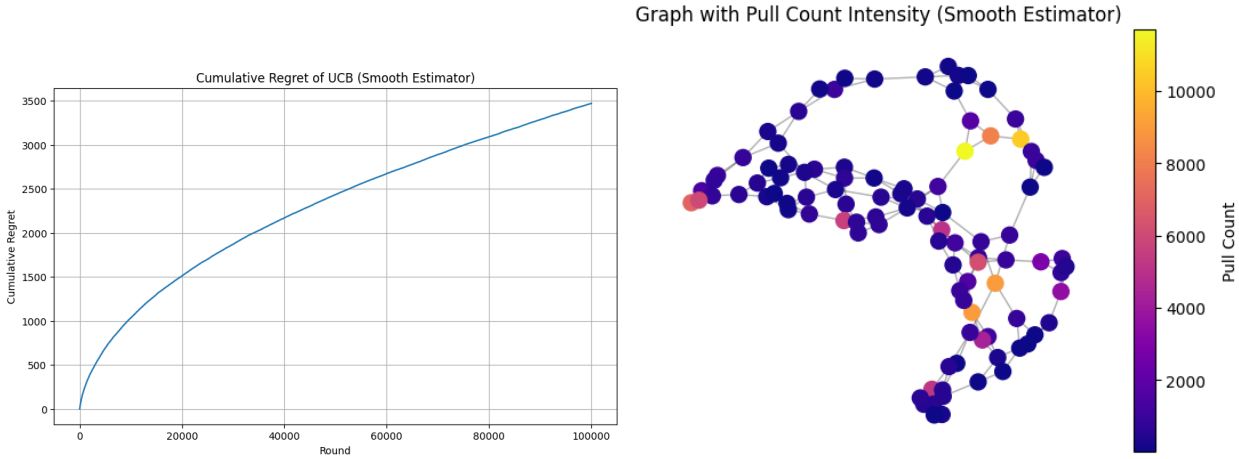
Input: all nodes K, horizon T, sampling probability p, noise standard deviation σ
Initialize reward estimates $\hat{\mu}_i$, pull counts $N_t(i)$ for all nodes i, and total regret
Initialize empty graph $G_{learned}$ with K disconnected nodes
for each round t in T **do**
  Create $G_t = (V, E_t)$ where 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 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

7.4 UCB - Empirical Test Results[3]



(a) Cumulative Regret (b) Heat Map of Played Nodes

Figure 4: UCB Base Model



(a) Cumulative Regret (b) Heat Map of Played Nodes

Figure 5: UCB Smooth Model

These models shows that the UCB performed with much higher regret. Looking at the heatmap of nodes pulled, it is shown that the algorithms were fooled by many suboptimal nodes, and could only choose the best node a little over 10,000 times, whereas epsilon-greedy could choose it at least 70,000 times. This will lend the belief that the bonus term has been improperly implemented as it is confusing the algorithm into thinking suboptimal nodes have higher true estimates. This hypothesis implies that the estimate metric (realized average against smoothed) is not being tested here, as irrespective of the estimate, basically the same choice is happening.

[3]Under the following parameter conditions ($T = 10^5, p = 0.1, \sigma = 0.1, \lambda = 0.1, K = 100$)

8 Final Comparisons: Varying Probability

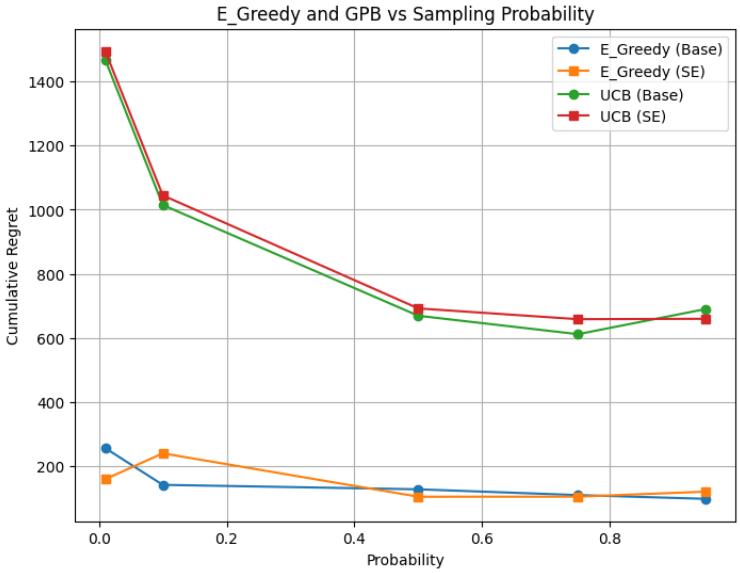


Figure 6: Comparing Algorithms across probabilities ($T = 10^4$

From this modeling, I consider the varying probability for all four algorithms. The initial tests were done with $p = 0.1$. It shows that UCB is very dependent on the probabilities, and with higher probabilities, achieves better regret. This could be because at these higher probabilities, the realized rewards are more frequent (from side observations). It requires these increased number of observations on the estimated rewards (either smoothed or realized average) to combat the bonus term's inability to differentiate nodes. For the epsilon-greedy setting, the probabilities have less of an impact. In both cases, the smoothened statistic matches roughly with realized, showing that it may be viable under different settings to induce this estimation.

9 Regret Bound Intuition

I will not seek to derive full regret bounds, as there is likely some difficulty in guaranteeing these proofs, but I will provide some intuition on how these proofs differ from the traditional settings. It can be further taken from these bounds on the number of rounds in which suboptimal arms are chosen, to bound final regret.

9.1 Epsilon Greedy: Bounding the number of times suboptimal arm i is chosen

I take the known Hoeffding concentration bound (Panageas, (2020)), holding for all arms, where $\tilde{N}_i(t)$ is the number of times arm i has been *observed* until time t, and $\hat{\mu}_i$ is some estimate of true μ_i (either realized average or smoothened):

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

This implies that in the Epsilon Greedy case, arm i will be chosen at time t only if:

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

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

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

Since the algorithm 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):

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

This summation is just the degree of node i. Finally this implies that the number of times arm i actually incurs regret is:

$$N_i(t) \leq \frac{\log(t)}{(1+pd)\cdot\Delta_i^2}$$

9.2 UCB: Considerations for β

Suboptimal arm i is only pulled at time t when $\mu_\star - \mu_i \leq 2\beta_t(i)$ for some β that captures the variance and bias of our estimator.

Assuming that the estimator had no bias, the known concentration inequality would hold when arm i has been *observed* $\tilde{N}_i(t)$ times: Estimator $\hat{\mu}_t(i)$ is within $\sqrt{\frac{\log(1/\delta)}{\tilde{N}_i(t)}}$ of true mean reward μ_i. I introduce the bias of our estimator because our estimate did introduce some controlling feature on when it saw non-smooth parts. The magnitude of bias is bounded by the λ regularization that occurred and the Laplacian, deriving something of the form based on the Cauchy-Schwarz inequality:

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

9.3 UCB: Bounding the number of times suboptimal arm i is chosen

With this found, it follows a similar proof as before, taking the known fact that Suboptimal arm i is only pulled at time t when $\mu_\star - \mu_i \leq 2\beta_t(i)$, for $\beta = \sqrt{\lambda S} + \sqrt{\frac{\log(1/\delta)}{\tilde{N}_i(t)}}$:

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

Solving for $\tilde{N}_i(t)$:

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

Using the fact from the Epsilon Greedy Proof where the algorithm 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 and d_i as the degree of node i):

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

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

10 Future Considerations

Further research can be done do discern why the Epsilon-Greedy algorithm worked better. This is hypothesized here to relate to the bonus term that was used. A more tight bound could be found based upon the side observation methods.

Regarding the smooth estimator, this research provides a framework for how to test and bound the properties of the smooth estimator as well as intuition and experimentation that show it may be viable. The mean realized rewards maintained better value, possibly because the smooth setting is not perfectly met. Other tighter definitions of smoothness may ensure that the smooth estimator is more realistic on the dataset. Instituting higher variance noise may also make the mean realized rewards less remarkable.

To truly test the effect of the smooth estimator, it will make sense to experiment with the removal of side observations. This is because the smooth estimation and side observation have similar effect on the protocol (learning about neighbors). The side observations are direct observations and so are obviously more accurate whereas smooth estimations have go through a smoothing process and a noise estimation.

Finally, formal regret bounds can be derived in terms of the actual number of times the nodes were selected to incur possible regret, which was derived here under some intuition.

11 Other Project Work

Project Presentation Slides

*Note that the experimentation in the slides was done on a different smooth graph dataset because the original pictures were not usable here. The code below can generate new graphs. All analysis still holds similarly.

Project Experimentation Code can be found: Google Colab.

12 References

Belkin, M., Matveeva, I., and Niyogi, P. (2004). Regularization and Semi-Supervised Learning on Large Graphs. In *Conference on Learning Theory*. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e4836ac9ec94daa9650c608a459ff50d5890f281

Nguyen, D. T., and Lauw, H. W. (2014). Dynamic clustering of contextual multi-armed bandits. *Singapore Management University Institutional Knowledge*. https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?referer=&httpsredir=1&article=3328&context=sis_research

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

Valko, M., Kveton, B., Munos, R., and Szepesvári, C. (2014). Spectral Bandits for Smooth Graph Functions. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*. https://proceedings.mlr.press/v32/valko14.pdf

Wang, C., Wang, T., Wang, Y., and Li, Y. (2021). Adversarial Linear Contextual Bandits with Graph-Structured Side Observations. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*. https://arxiv.org/pdf/2012.05756

Wu, Q., Wang, H., Gu, Q., and Wang, H. (2016). Contextual bandits in a collaborative environment. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. https://dl.acm.org/doi/pdf/10.1145/2911451.2911528

Yang, L., Han, D., Liu, M., and Mannor, S. (2019). Laplacian Regularized Graph Bandits: Algorithms and Theoretical Analysis. *arXiv preprint*. https://arxiv.org/pdf/1907.05632