Maximizing User Engagement in Social Network Advertising

Jonathan Valverde
Department of Computer Science
University of Maryland, College Park
valverde@umd.edu

Marco Bornstein
Department of Mathematics
University of Maryland, College Park
marcob@umd.edu

Ran Li
Department of Mathematics
University of Maryland, College Park
ranli@umd.edu

1 Introduction

In this paper, we pose the problem of maximizing user engagement with advertisements in a social network. The problem framework we propose is inspired by social networks, such as Instagram and Twitter, in which a few users are followed by a large number of users. These heavily-followed users are often called influencers: someone (or something) with the power to affect the buying habits or quantifiable actions of others by uploading some form of original—often sponsored—content to social media platforms like Instagram, YouTube, Snapchat or other online channels.

If we consider the social network as a directed graph, with edges going from one user to another if the latter follows the former, then high-degree nodes are highly valuable. In the case that an influencer engages with an ad and purchases a product, the influencer may recommend the product to their followers. Therefore, many more users become exposed to the product, increasing the likelihood it will be purchased. While showing an ad to an influencer has a high potential reward, there is also a high risk. There may be limited chances of exploring with high-influence nodes because there are higher stakes when showing an ad to a celebrity.

Realistically, ads might only be shown to influencers at the same or lower rate than normal users. Moreover, there is a higher cost to exploring (choosing potentially sub-optimal ads) with an influencer than with a non-influencer, as showing an unsuccessful ad to an influencer entails a missed opportunity of reaching a large number of users with a single ad interaction. In contrast, follower nodes are more abundant and cheaper to explore with.

These follower nodes have another key advantage: they can provide valuable insight into the interests of the influencer they follow. The collective interests of the set of users that follow an influencer likely reflect the interests of the influencer. Therefore, there is a benefit to explore the users that follow an influencer: there is a lower cost to show them an ad that does not result in user engagement, and if we sample enough of them, we can get information about which ads might successfully interact with the influencer. This scenario motivates our work.

In our paper, we model the problem of maximizing user engagement with advertisements in a social network as a multi-armed bandit (MAB) problem. Multi-armed bandits are a simple and powerful framework where algorithms are used to make decisions over time under uncertainty [7]. The multi-armed bandit problem seeks to find the optimal action, or allocation of resources, over a set of possible actions in a way to maximize expected gain. The difficulty in this problem lies in the fact that the set of actions have partially, or completely, unknown properties (related to the expected gain) which only become better understood after repeatedly selecting them.

In a scenario like online advertising, we might have to choose from n_a actions, or arms, for each user in a social network. These might be ads to show to each user. We can then formulate a multi-armed bandit, where over t_{end} time steps we must identify the best ads to show to each user. More importantly, we explore the setting in which we must gain observations and information from any related follower nodes to better select ads for an influencer node. The complete formulation of our multi-armed bandit, as well as the detailed description of our setting, is described further on in Section 3.

2 Related Work

2.1 Multi-Armed Bandits and Advertisement

Many advertising campaigns face the dilemma of exploring different ads versus exploiting the ads that perform best. The exploration-exploitation nature of advertisements makes it a prime candidate to apply MABs to solve. As a result, there are multiple papers which study MABs within an advertisement setting [1, 6, 10].

In the work of [6], a similar problem to ours is formulated. Within this paper, a firm is trying to select which group of ads should be shown on a set of websites in order to maximize the total number of customers acquired from ad impressions. To do so, a batched [4] and hierarchical [11] MAB is formulated to allocate an optimal proportion of the available impressions on each ad within each website in each decision period. In each period, the firm is able to make different allocations of impressions of the ads on the websites.

Within [10], ad selection is also modeled as a MAB problem. Specifically, they aim to best match online advertiser bids to available advertising inventory on a demand-side platform. In this complicated and real-time bidding scheme, [10] develop a dynamic contextual MAB. The authors construct a MAB formulation that is dynamic to changes in the conversion rate of each arm (important within the real time bidding environment), can model the overlapping of arms (allowing samples to no longer be i.i.d), and will incorporate delayed feedback present in the testing environment.

A similar setting of advertising systems embedded within an online social network is considered in [1]. Closely related to our work, [1] models an MAB which incorporates observations of neighboring nodes' rewards for an action taken by the current node. These neighboring node observations are known as side observations. Within our work, we utilize a variation of side observations. If an action is taken for a current node which results in a reward (a click of the ad), then all neighboring nodes are displayed this ad (amounting to a simultaneous free pull of the arm for each neighboring node). The work in [1] is the closest application of a MAB to a setting like our own, and directly leads into the other area of literature we have studied: graph multi-armed bandits.

2.2 Graph Multi-Armed Bandits

There have been multiple papers exploring the incorporation of graph structures within bandit problems, and a comprehensive survey is given in [9]. We will follow this survey roughly in this section. It categorizes the use of graphs within bandits to achieve three different goals: smooth rewards, modelling of side observations, and posing influence maximization problems.

As explained in [9], when modelling smooth rewards, the arms usually correspond to nodes in a graph. The rewards then vary smoothly as a function of how the nodes are connected. In targeted advertisements, the graph represents a social network structure, and the problem is to find a group of users interested in a product. In recommender systems, the graph might represent the similarity between products, and the problem is to recommend products to a given user. The work in [5] is closely related to our framework. In that setting, the users are partitioned into clusters that are unknown to the learner. Users within a cluster yield similar rewards for a given arm.

With regards to modeling of side observations, the scenario involves, for an arm pull, observing not only the reward from that arm pull, but the rewards of related arms as well. In a graph, this can be querying a node in a graph and obtaining information about the node and its neighbors node. Applications to recommender systems can model similarity between different items to recommend to a user [9]. Note that this framework is not immediately applicable to our problem, because our arms correspond to ads, which do not reveal information about one another. In our case, since pulling

an arm for a user reveals rewards about similar users, this is more similar to the smooth rewards in graphs scenario.

A variation on side observations is networked bandits, proposed in [3] and also studied in [8]. This is similar to the case of side observations, except that the arm pulls from the node neighbors count towards the total reward, and are not just considered observations. These are also called side rewards [8]. Note that this can change which arm yields the optimal expected reward, compared to the case of side observations. This scenario resembles the problem we are posing more closely, except that, in general, work has focused on the case in which the rewards obtained from the neighbors are not a function from the reward obtained from the original node (in our case, a click from an influencer is needed to obtain arm pulls from the followers).

In [3], the authors consider the nodes to correspond to arms and pose the problem as a contextual bandit, where a context vector represents the user features and the features of the ad or message to be shown. Playing an arm, which amounts to choosing a user to show an ad to, gives the reward of that user and of all neighboring users. Nonetheless, the authors do not impose a constraint on showing ads to users with high in-degrees, which differs from our setting (in a simple case with one central user and a high number of followers, ads would only get shown to the central user, because they result in the highest possible number of clicks, but we argue that we cannot spam high-degree nodes with sub-optimal ads). Furthermore, the authors assume that the expected reward for an ad shown to a user is linear in a context vector. In spite of these differences from our setting, it might be possible to adapt their algorithm by posing our problem as a contextual bandit and incorporating the constraints or costs for choosing high-degree nodes. Such an adaptation will be left for future work.

Finally, influence maximization deals with selecting nodes in a graph structure that can optimally propagate some sort of information (e.g. a product) throughout a network [9]. Such a scenario is similar to our framework, except that we consider the problem of choosing actions for the nodes instead of node selection. Moreover, we place restrictions on showing ads to high-in-degree nodes. Our problem involves, for different users, executing different exploration/exploitation strategies and identifying different optimal arms.

3 Problem Formulation

3.1 General Framework

Consider a directed graph G with n_u users, with an edge set E, where edges represent a user following another user. There are n_a ads, which correspond to the bandit's arms. We enumerate the users and ads for notational convenience. When the ith user is presented with the jth ad, the user will click on the ad with a fixed, but hidden, probability $p_{i,j}$ (the click-through probability, CTR). A reward of 1 is received if the user clicks, and 0 otherwise. Moreover, *if* the user clicks on the ad, then all of the user's followers will be exposed to the ad and click with a probability given by their individual CTRs. An additional reward of 1 is gained for any new clicks. In other words, we simulate a pull of the jth arm on all of the ith user's followers if user i clicks. However, we will assume that this chain of clicks stops at the ith user's followers (this assumption could be relaxed in future work).

Furthermore, for the jth ad, we assume the CTR probability function $p_{*,j}$ has some degree of smoothness in terms of the arrangement of nodes on the graph. This can be simulated in different ways, such as generating random $p_{i,j}$ values with a correlation structure based on the graph, with close nodes having high correlation in their $p_{i,j}$ values. Alternatively, the CTR probability functions might be defined around a few high-degree nodes in the graph, such that the CTRs of a follower are based on the CTRs of who they follow, plus some noise.

At each time step, we show an ad to a user with the goal of obtaining the maximum cumulative expected reward. We assume that there is some limitation on showing ads to high-in-degree nodes, such as a cost proportional to the in-degree of the nodes or a schedule that limits how often a high-in-degree node can get shown an ad. The agent might or might not have control over which user gets chosen at a given time step.

Note that this general formulation is very open-ended and can result in different variants of the problem. Our goal is to study this framework by starting with a specific, simplified instance of this problem. Future work could focus on relaxing these simplified assumptions to solve more complicated versions of the problem.

3.2 Problem Simplification: Star Network

To the best of our knowledge, this problem has not been studied in such a formulation. Although similar scenarios have been studied, as presented in Section 2.2, we have not seen authors address the problems of (1) availability of side rewards being dependent on the outcome of the arm pull from the queried node and (2) limited ability of exploration with high-degree nodes. Therefore, we will first study a simplified scenario consisting of a single influencer node connected to n_f "follower" nodes. The edges are directed from the influencer to his followers.

We will enforce smoothness in the CTR probabilities by assuming that, for any ad j, the CTR of follower i for that same ad is generated by adding random noise $\epsilon_{i,j}$ to the CTR probability of the influencer.

Furthermore, we will limit how often an ad can be shown to the influencer by creating a schedule based on epochs. At each epoch, the followers are shuffled. The agent chooses an ad for each of these followers, and at the end of the epoch, the agent is able to show an ad to the influencer. The goal is to maximize the expected reward across all epochs.

4 Algorithms

A solution to the problem presented in Section 3.2 should: (1) exploit the similarity of the reward distributions between the followers and the influencer while (2) trying to avoid bias in the empirical estimates of the arm rewards. These requirements are somewhat at odds with each other. For the first requirement, we might want to pool together observations from all of the followers. After all, if the mean rewards of the followers are close to one another, it would be wasteful to not take into account the observations from the other followers. We could then use these to estimate the mean rewards of the influencer. However, if we consolidate all reward observations into a single empirical estimate for all users, and we try to use it as an approximation of the mean rewards of some given follower, the estimate will most likely be biased, because it will not necessarily converge to the mean rewards of that given follower. In other words, for a given action, the mean reward across all users will most likely be different than the mean reward of that action for any given single user.

We therefore propose two algorithms, each prioritizing one of these objectives. The first algorithm, UCB User-Wise (UCB-UW), keeps a separate UCB instance for each user. The second algorithm, UCB Centralized (UCB-Cent), pools all of the follower observations into a single empirical estimate for all of the follower's rewards. We further discuss these algorithms in the following sections. UCB-UW will have unbiased estimates of the rewards, at the cost of additional arm exploration, while UCB-Cent should explore more efficiently, at the cost of having biased arm reward estimates.

Before introducing these algorithms, we present some notation common to them. We let the followers be labeled 1 through n_f. The influencer (central user) will have an index of 0. Let $\bar{x}_{ia}$ and o_{ia} be the empirical mean reward estimate and the number of observations, respectively, for action a and user i. These will both count instances in which user i is presented with action a by means of the influencer (a successful arm pull of action a on the influencer).

4.1 UCB-UW

This algorithm is a slight modification of UCB-SO [1]. We outline the steps in Algorithm 1. The main idea is to have one UCB instance per each user. The most notable difference with respect to traditional UCB is that, upon a successful arm pull from a user, the results of the arm pulls of its followers are recorded and used to update the empirical means of the rewards for that action for the followers. While the algorithm in [1] is similar, we make changes to reflect that (1) rewards for followers are obtained only upon a successful arm pull and (2) we follow the user sampling schedule we have defined in Section 3.2. Note that, when an arm has not been pulled for a given user, its UCB bound is infinite, so all arms must be sampled at least once for all users. This would likely not be practical in the setting of online advertisement, especially if we want to avoid suboptimal ad choices for the influencer, but the algorithm provides a useful benchmark nonetheless.

Algorithm 1: UCB-UW

INPUT: t_{end}, arms, users
for $t = 1, ..., t_{end}$ **do**
  $u \leftarrow$ shuffle $(\{1, ..., n_f\})$
  for $i = u_1, ..., u_{n_f}$ **do**
    $a = \arg\max_a \bar{x}_{ia} + \sqrt{\frac{\log(t)}{o_{ia}}}$
    Sample action a for user i and obtain reward r
    Update $\bar{x}_{ia}$ and o_{ia} based on r
  end
  $a = \arg\max_a \bar{x}_{0a} + \sqrt{\frac{\log(t)}{o_{0a}}}$
  Sample action a for user 0 and obtain reward r
  Update $\bar{x}_{0a}$ and o_{0a} based on r
  if $r > 0$ **then**
    for $i = 1, ..., n_f$ **do**
      Sample action a for user i and obtain reward r
      Update $\bar{x}_{ia}$ and o_{ia} based on r
    end
  end
end

4.2 UCB-Cent

The main idea behind UCB-Cent, shown in Algorithm 2, is to keep a single UCB instance for all of the followers, since the reward profiles for them are similar. Let $\bar{x}_{fa}$ and o_{fa} be the empirical mean reward and the number of observations for action a across all followers. Actions are chosen for the followers by using UCB based on $\bar{x}_{fa}$ and o_{fa}. Note the use of t_{tot}, the running number of total arm plays across all followers, instead of t in the UCB term for the followers.

As discussed at the beginning of Section 4, the estimates $\bar{x}_{fa}$ do not actually reflect the mean reward for action a for any of the given followers, because their mean rewards for a are all (likely) different. This could mean that, for any given follower, we could converge to a suboptimal action. However, these centralized estimates are a straightforward way to incorporate the smoothness prior into our estimates, while allowing us to experiment with how well we can infer the influencer mean rewards.

For the influencer, we use one of two strategies, specified as the algorithm "mode": GreedyInf or UCBInf. Both are based on the motivation of behind the problem we posed, to try to infer the influencer preferences based on the behavior of the followers. For GreedyInf (Greedy Influencer), we pick the arm with the highest empirical mean estimates pooled from all of the followers, $\bar{x}_{fa}$. For UCBInf (UCB Influencer), we use a UCB algorithm, with the empirical reward estimate approximated from the pooled estimate from the followers: $\bar{x}_{fa} + n_f \bar{x}_{fa}^2$. This is chosen because, if the true CTR for the influencer is p, and we approximate the followers as having exactly the same CTR (with no noise), then the expected value for the arm pull is $(1-p) \times 0 + p(1+n_f p) = p + n_f p^2$. We introduce this UCB strategy for the influencer to encourage some exploration and help reduce possible bias effects (introduced by using the estimates $\bar{x}_{fa}$ to approximate the arm CTRs of the influencer).

It is possible to use several other strategies to choose the influencer's arm. We could incorporate the empirical reward estimates from the followers, and then correct the estimate once sufficient observations from the influencer become available. This could give us both faster convergence towards optimal arms while removing some of the bias in the estimates. A similar strategy could be used for the followers. Another approach might incorporate the smoothness of the rewards by computing a loss based on the squared differences between the reward estimates of neighboring nodes and using it to regularize the estimates. Such a loss is used in spectral clustering [9] and has been incorporated for linear graph bandits by [2]. We leave such exploration of methods incorporating smoothness into the observation updates for future work, but UCB-Cent provides us with a way to measure the value of using the follower observations to try to infer, as quickly as possible, the influencer preferences.

Algorithm 2: UCB-Cent

INPUT: t_{end}, arms, users, mode
$t_{tot} = 1$
for $t = 1, ..., t_{end}$ **do**
  $u \leftarrow$ shuffle $(\{1, ..., n_f\})$
  for $i = u_1, ..., u_{n_f}$ **do**
    $a = \arg\max_a \bar{x}_{fa} + \sqrt{\frac{\log(t_{tot})}{o_{fa}}}$
    Sample action a for user i and obtain reward r
    Update $\bar{x}_{fa}$ and o_{fa} based on r
    $t_{tot} = t_{tot} + 1$
  end
  if *mode* $=$ *GreedyInf* **then**
    $a = \arg\max_a \bar{x}_{fa}$
  end
  else if *mode* $=$ *UCBInf* **then**
    $a = \arg\max_a \bar{x}_{fa} + n_f \bar{x}_{fa}^2 + \sqrt{\frac{\log(t_{tot})}{o_{fa}}}$
  end
  Sample action a for user 0 and obtain reward r
  if $r > 0$ **then**
    for $i = 1, ..., n_f$ **do**
      Sample action a for user i and obtain reward r
      Update $\bar{x}_{fa}$ and o_{fa} based on r
    end
  end
end

4.3 Epsilon-Greedy LP

Another algorithm that we test to solve the problem presented in Section 3.2 is the Epsilon-Greedy LP presented in [1]. Similar to our problem, [1] want to find a solution to allocating advertisements in an online social network in the presence of side observations. To accomplish this, a Greedy-Epsilon algorithm is introduced which allows "exploration for each user at a rate proportional to her network position" [1]. This algorithm utilizes a Linear Program (LP) to first minimize the exploration between adjacent nodes in a network. The LP is shown below in Equation (1).

$$\begin{aligned} \min \sum_{i \in N} z_i \\ \text{subject to:} \quad A_i \cdot \boldsymbol{z} \geq 1, \quad \forall\, i \in N \\ z_i \geq 0, \quad \forall\, i \in N \end{aligned} \tag{1}$$

The variable N denotes the set of nodes (users) and A_i is the ith row of the node adjacency matrix A. The weights z_i serve the function of exploration weights for each user in the graph.

As mentioned above, one can interpret this LP as minimizing the total exploration among a node and its neighbors. The reason to minimize exploration between adjacent nodes is that a neighbor node's exploration is equivalent to a node's own exploration due to the presence of side observations. Thus, less exploration is needed within a dense connection of nodes to gain better insight into each node's preferences. This is formulated as saying that at least one unit of exploration is required to be split up between a node and its neighbors, while the total exploration weights are minimized.

The optimal solution of Equation (1) determines the units, or weights, of exploration for each node z_i. For a specific node, the weight of exploration z_i plays a crucial role in how much to explore versus how much to exploit in the Epsilon-Greedy LP shown below in Algorithm 3. Also playing a role are the hyperparameters c and d as well as the total number of actions M and the current round t. The hyperparameters c and d adjust how quickly the probability of exploring ϵ decays (at a rate $\frac{cM}{d^2 t}$). In our work, we choose $c = 1$ and $d = 0.5$ (close to the values set in [1]) so that ϵ begins to decrease when $t > 4M$. Once the weights of exploration are known for each node, a standard

Epsilon-Greedy method is applied with a twist: the probability of exploration for each node ϵ_i is set directly proportional to the flow of exploration for a specific node z_i. Thus, nodes that have a high weight of exploration will have a relatively higher probability for exploration. This is how the Epsilon-Greedy LP algorithm manages to create "exploration for each user at a rate proportional to her network position" [1] as mentioned above.

In our problem formulation, the influencer node is densely connected. In fact, the only edges that exist in our graph are between the influencer and their followers. No edges exist between followers for our simplified problem formulation. Therefore, the solution to Equation 1 in our formulation will result in $z = (1, 0, 0, \ldots, 0, 0)^T$. One can see this, as each follower node can shift the weight of exploration all onto its adjacent influencer node. This makes sense, as each exploration of the influencer will result in an exploration for each of the follower nodes. As expected, the optimal LP solution yields the following: the network position of the influencer is quite high.

Due to the optimal solution of the LP for our simple formulation, the Epsilon-Greedy LP algorithm will explore considerably for the influencer node and not at all for the follower nodes. This is not what we expect to be an optimal strategy, however it is included as it will serve as an important and interesting benchmark for us (majority exploration for the influencer, no exploration for followers). For this algorithm, we expect to see a large amount of regret for the influencer node and lower (closer to optimal) regret for the follower nodes. We have modified the Epsilon-Greedy LP algorithm from [1] to fit our problem formulation, and show it in Algorithm 3.

Algorithm 3: Epsilon-Greedy-LP

INPUT: arms, users, optimal solution z^* of LP in Equation (1), $c > 0$, $d \in (0, 1)$
for $t = 1, ..., t_{end}$ **do**
  $u \leftarrow$ shuffle $(\{1, ..., n_f\})$
  for $i = u_1, ..., u_{n_f}$ **do**
    $\epsilon_i = z_i^* \min(1, \frac{cM}{d^2 t})$
    $a = \arg\max_a \bar{x}_{ia}(t)$
    With probability $1 - \epsilon_i$ select a, else select an action uniformly at random
    Obtain reward r and update $\bar{x}_{ia}$ based of r
  end
  $\epsilon_0 = z_0^* \min(1, \frac{cM}{d^2 t})$
  $\hat{a} = \arg\max_a \bar{x}_{0a}(t)$
  With probability $1 - \epsilon_0$ select $\hat{a}$, else select an action uniformly at random
  Obtain reward r and update $\bar{x}_{0,\hat{a}}$
  if $r > 0$ **then**
    for $i = 1, ..., n_f$ **do**
      Sample action $\hat{a}$ for user i, obtain reward r, and update $\bar{x}_{i\hat{a}}$ based on r
    end
  end
end

5 Empirical Results

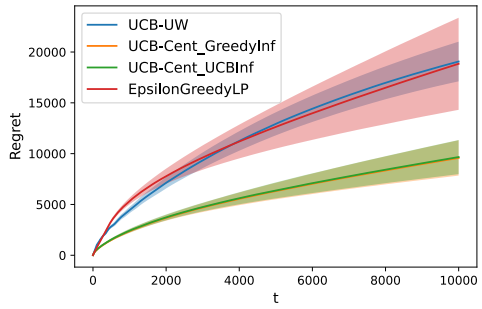
We generate an instance problem as follows. First, we generate the CTRs for the influencer by sampling uniformly from the interval $[0.1, 0.9]$. Next, for each influencer, we generate uniform noise

Table 1: Full Regret Comparison

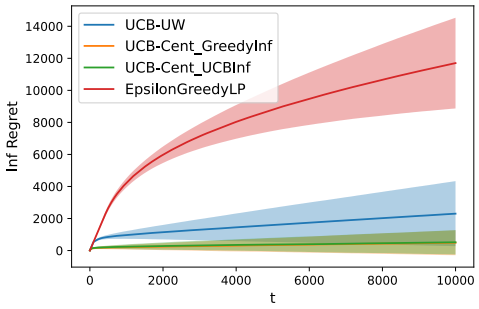
Followers	UCB-UW	UCB-Cent-GreedyInf	UCB-Cent-UCBInf	Epsilon Greedy LP
10	19,064	9,578	9,664	18,841
20	37,950	16,117	16,185	36,581
40	76,705	29,091	28,995	74,129

Table 2: Influencer Regret Comparison

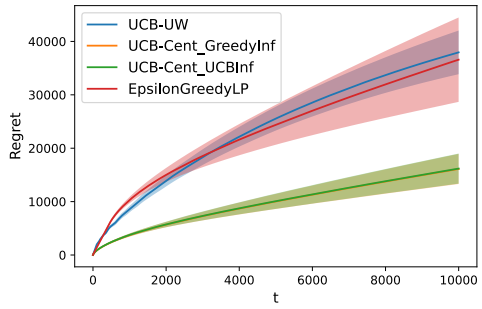
Followers	UCB-UW	UCB-Cent-GreedyInf	UCB-Cent-UCBInf	Epsilon Greedy LP
10	2,302	487	516	11,702
20	4,477	523	574	21,977
40	9,816	719	636	43,880



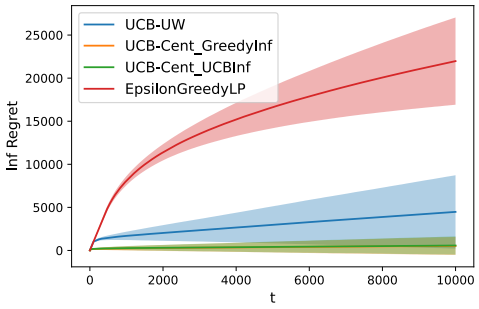
(a) 10 followers, full regret



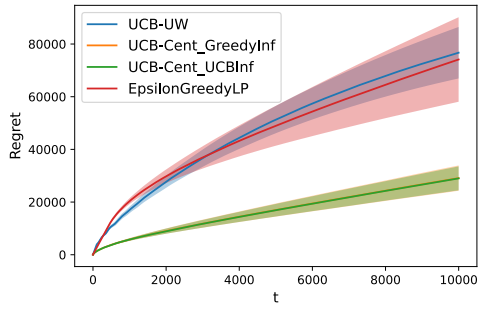
(b) 10 followers, influencer regret



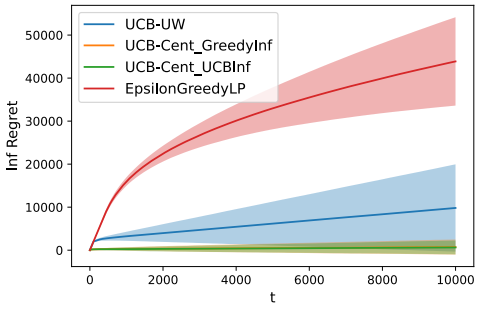
(c) 20 followers, full regret



(d) 20 followers, influencer regret



(e) 40 followers, full regret



(f) 40 followers, influencer regret

Figure 1: Full regret (left) and influencer regret (right) for 10, 20, and 40 followers. Where not visible, the UCB-Cent_GreedyInf algorithm curve is behind the UCB-UCB_Inf curve.

in the interval $[-0.1, 0.1]$, and add it to the influencer's CTRs to obtain the followers' CTRs. Finally, we run $1,000$ episodes and compute average statistics.

For this simulation, we set the number of ads to be 100, and generate experiments with 10, 20, and 40 followers. We want to study the case of less followers than ads, because this means that finding the optimal ad efficiently requires pooling of the observations from all followers. We use $10,000$ time steps in our simulations (for each user), or 100 times the number of ads.

The results are shown in Figure 1. First, we plot the full regret, where we accumulate the regret for all users for each time step (one full round of arm pulls across all users). Next, we plot the influencer regret, defined as the regret for the influencer, computed with respect to the total reward from the influencer and their followers.

One of the immediate takeaways from the results in Figure 1 is that the UCB-Cent algorithms greatly outperform the UCB-UW and Epsilon Greedy LP algorithms. This is true for both the full regret and influencer regret metrics. The UCB-Cent algorithm experiences less than half of the full regret than the other algorithms experience for *all* number of followers tested. This is consistent with our expectation that UCB-Cent more efficiently explores the ads by keeping a centralized estimate of the ad CTRs, but it is notable that it outperformed the other algorithms in spite of its use of biased estimates. In an infinite horizon, however, UCB-Cent might converge to slightly suboptimal arms, resulting in linear regret (with a potentially small slope).

The difference in regret for the influencer, the most important and rewarding node in our network, is even more stark. The regret is approximately ten times greater for UCB-UW than the UCB-Cent algorithms. The Epsilon Greedy LP fares even worse, with regret that is approximately 40 times worse than the UCB-Cent algorithms. From these differences, it is clear that most of the contribution to the full regret for UCB-Cent comes from the follower nodes. When only the influencer regret is considered, the advantage of using UCB-Cent is higher. This is consistent with our design choice of UCB-Cent, which was to use exploration of the follower nodes to prioritize minimizing the regret incurred with the influencer. Furthermore, the fact that there is little difference between the GreedyInf and UCBInf variants of UCB-Cent indicates that the driving factor behind the reduction in regret is the pooling of follower observations. Although we designed UCBInf to have less bias issues than GreedyInf, we do not observe a significant difference in the regret incurred by both variants of the algorithms.

We also note that, every time we double the number of followers, the full regret roughly doubles as well for all algorithms. This also occurs with the influencer regret, except in the case of UCB-Cent. UCB-Cent achieves smaller increases in influencer regret, with an increase of roughly 40% from 20 to 40 followers for GreedyInf, and around 10% in all other cases. Overall, the doubling of regret is to be expected in UCB-UW and Epsilon-Greedy LP. This is because the number of individual bandit problems is roughly doubled, and these algorithms do not use observations for one follower to update the empirical estimates for the other followers in any way. In contrast, for UCB-Cent, doubling the number of followers will increase the amount of information available to estimate the arm rewards for the influencer. We expected this to result in fewer sub-optimal choices for the influencer. This can be partially seen in the influencer regret sub-figures (the right-hand sub-figures of Figure 1), where there is a much slower growth of regret. Note that doubling the number of followers also roughly doubles the regret for a suboptimal arm choice in expectation, even for influencer regret. The fact that the influencer regret does not double for UCB-Cent is an indicator that the algorithm is indeed making less suboptimal choices by taking advantage of the increased amount of observations.

Overall, UCB-Cent incorporates the prior on the reward structure (similar rewards for all users) to achieve a lower regret growth. Nonetheless, it is worth mentioning that the bias issues for UCB-Cent would require a correction to avoid linear regret in a long horizon, whereas UCB-UW and Epsilon-Greedy LP would not suffer from this issue. One option would be using UCB-UW as a warm start, and then switching to UCB-UW or Epsilon-Greedy LP.

One particular issue that could dictate algorithm design is the trade-off between full regret and influencer regret. When increasing the number of followers, the influencer regret grows slower than the full regret for UCB-Cent. Furthermore, while UCB-UW and Epsilon-Greedy LP achieve similar full regret, the influencer regret for UCB-UW is much lower. This occurs because the LP optimized by Epsilon-Greedy LP seeks to minimize a notion of collective exploration for all nodes while ensuring all arms get some minimum amount of exploration. The result is that the influencer is

heavily used to explore. If we modify the problem objective to minimize exclusively the influencer regret, it might be possible to modify the LP to achieve a lower influencer regret.

Whether full regret, influencer regret, or a weighted combination thereof should be minimized might be a function of the problem setting, with issues such as the practical cost of suboptimal ad choices for the influencer to be taken into consideration.

6 Conclusions

This project serves as an initial study into the problem of prioritizing regret reduction for high influence nodes in a multi-armed bandit graph problem. While previous work in literature has addressed some other aspects of side rewards and side observations in graph structures in bandits, there is value in studying the setting of prioritizing few high-degree nodes, for applications such as social networks.

Our results highlight the importance of incorporating the reward structure (i.e. the smoothness of the rewards across the graph) into algorithms to solve this problem. Furthermore, it is clear that the choice between full regret or the influencer regret as the problem objective dictates algorithm choices.

While our results did not show a high cost of bias in the mean reward estimates for the algorithms, this might be because the only unbiased algorithms did not make an efficient use of the reward structure of our problem. Where the reward structure incorporated into the algorithms does not match the true reward structure, or where incorporating this structure introduces bias in the algorithms, there is a possibility of not converging towards the optimal arm. The best of both worlds could entail using a reward structure to warm start an algorithm, and using algorithms that rely on unbiased estimators afterwards.

7 Future Work

The algorithms we studied in this project need to be significantly changed to handle more general scenarios. The most immediate modification would be to have multiple influencers, with followers possibly following more than one influencer. A question then becomes what is a reasonable model for the relationship between a user's CTRs and those from the influencers followed by the user. Here, notions of function smoothness over graphs would be very useful, although more elaborate, realistic models might be used depending on the application (such as social network advertisement).

We have also not studied the setting of being able to choose what followers are sampled before sampling the influencer. An interesting problem might be to have a limited budget of arm pulls and have the follower rewards reflect those of the influencer at different degrees. This would require adaptively identifying the followers that provide the behavior most similar to that of the influencer, and prioritizing sampling accordingly. Yet another possibility is to have the graph edges be weighted by some external model of predicted similarity between followers and influencers. These might then also inform empirical reward estimate updates and uncertainty estimates.

While we have avoided relying on the assumption of the bandit having linear rewards, some of these settings might require incorporating such a structure. If such an assumption was incorporated, there are existing algorithms in the literature that might be leveraged or adapted for these scenarios.

References

[1] Swapna Buccapatnam, Atilla Eryilmaz, and Ness B Shroff. Multi-armed bandits in the presence of side observations in social networks. In *52nd IEEE Conference on Decision and Control*, pages 7309–7314. IEEE, 2013.

[2] Nicolo Cesa-Bianchi, Claudio Gentile, and Giovanni Zappella. A gang of bandits. *arXiv preprint arXiv:1306.0811*, 2013.

[3] Meng Fang and Dacheng Tao. Networked bandits with disjoint linear payoffs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1106–1115, 2014.

[4] Zijun Gao, Yanjun Han, Zhimei Ren, and Zhengqing Zhou. Batched multi-armed bandits problem. *arXiv preprint arXiv:1904.01763*, 2019.

[5] Claudio Gentile, Shuai Li, and Giovanni Zappella. Online clustering of bandits. In *International Conference on Machine Learning*, pages 757–765. PMLR, 2014.

[6] Eric M Schwartz, Eric T Bradlow, and Peter S Fader. Customer acquisition via display advertising using multi-armed bandit experiments. *Marketing Science*, 36(4):500–522, 2017.

[7] Aleksandrs Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019.

[8] Shaojie Tang, Yaqin Zhou, Kai Han, Zhao Zhang, Jing Yuan, and Weili Wu. Networked stochastic multi-armed bandits with combinatorial strategies. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 786–793. IEEE, 2017.

[9] Michal Valko. *Bandits on graphs and structures*. PhD thesis, École normale supérieure de Cachan-ENS Cachan, 2016.

[10] Hongxia Yang and Quan Lu. Dynamic contextual multi arm bandits in display advertisement. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1305–1310. IEEE, 2016.

[11] Yisong Yue, Sue Ann Hong, and Carlos Guestrin. Hierarchical exploration for accelerating contextual bandits. *arXiv preprint arXiv:1206.6454*, 2012.