Sampling For Community Structure in Multiplex Networks with Multilevel Multi-Armed Bandit

Ricky Laishram  Sucheta Soundarajan  Jeremy D Wendt
Department of Electrical Engineering & Computer Science
Syracuse University, Syracuse, NY USA
{rlaishra, susounda}@syr.edu
Sandia National Laboratories
Albuquerque, New Mexico USA
jdwendt@sandia.gov

Abstract

In this paper, we look at the problem of sampling (through crawling) for the community structure in a layer of interest in a multiplex network. If the cost of a query on the layer of interest is higher than that of the other, we can explore the cheaper layers and use the information gathered to make better decisions on which node to query on in the expensive layer.

The problem that we address is found in a lot of real world cases. For example, if we want to find the community structure of the Twitter Friends-Followers relationship, the API rate limits is a huge hindrance. However the API rate limits for querying the user timeline to fetch the mentions and replies layer is huge comparatively.

To solve this problem, we propose an algorithm called MultiComSample. This algorithm uses multiple multi-armed bandits to figure out what are the best layers, communities and roles to select nodes to query are.

We tested our algorithm against four baselines on four real-world multiplex networks. The experimental results show that our algorithm outperforms all the baselines by a significant margin.

1 Introduction

In a network as community is defined as a group of densely connected nodes (Wasserman and Faust 1994). Finding communities in networks is of great research interest because it give us valuable insight into collective behavior of the nodes, and there has been a lot of works in this area (Newman and Girvan 2004; Blondel et al. 2011).

More recently the has been interest in sampling for community structures (Maiya and Berger-Wolf 2010). This is necessitate by the fact that a large number of graphs are too large to be analyzed as a whole. In many cases, we do not even have access to the full network - we have to explore the network as well.

In the real world, there are many networks where the nodes have different modes of interaction. Many existing works focus on only one type of interaction - discarding a lot of potentially useful information. Such types of networks where the nodes have different modes of interaction are a special type of multilayer network known as multiplex network (Lee, Min, and Goh 2015).

Another aspect that we need to consider is that these different interaction (or layers as we shall refer to in the rest of the paper), can have different cost of exploration associated with them. As an example, consider the Twitter Follower-Following network. If we want to explore this network, there is a constrain on how fast we can collect the data due to the API rate limit. The Twitter API limits queries for friends and followers to only 15 every 15 minutes (Twitter ). However, the Twitter Timeline API allows 1500 queries every 15 minutes. So, we can leverage the information gathered from the layers generated from the timeline API to make better choices of which nodes to query on in the Friends layer that maximizes (or minimizes) the objective.

In this paper, we look at the problem of sampling to find the community structure in a layer of a mutiplex network which is costly to explore. To achieve this we can utilize the information gathered from the other layers that are cheaper to explore (Section 3). We propose an algorithm, which we refer to as MultiComSample, that solves this problem by making use of multiple multi-armed bandits to select the best layer, community and role combinations to query on in the expensive layer.

The challenges associated with this problem are:

1. The network can be explored only thorough crawling.
2. The layer of interest is costly to explore.
3. We do not know which of layers that are easier to explore are the important (for example highly correlated) with the layer of interest.

We compare our proposed algorithm against various baseline on six different networks. We observe that our algorithm consistently outperforms all the baselines in all the networks considered.

In Section 2 we discuss some related works, and describe the problem and evaluation metric in Section 3. We describe our proposed algorithm in Section 4, and the experimental results in Section 5. Finally we conclude with some discussion about the results and discussion about future works in Section 6.
2 Related Works

There has been a lot of work on community detection in single-layer networks (Fortunato 2010; Girvan and Newman 2002; Blondel et al. 2011). However the size of the not a constrain in these works. Maiya et al. (Maiya and Berger-Wolf 2010) have worked on the problem of generating a sample of a network that is representative of the community structure of the original network. There has also been

In the case of community detection in multilayer network, there are far fewer works. The main reason for this seems to be because there are very few generalization of the single layer community detection methods to multilayer networks (Kivelä et al. 2014). One huge problem in the development of a community detection algorithm is the lack of a null model that takes the inter-network edges into consideration. So, most of the works have been on special types of multilayer networks.

Mucha et al. (Mucha et al. 2010) generalized the community detection methods to a special class of multilayer networks called multislice networks. In their work they generalize modularity to node-tuple pairs and group the node-tuple pairs into communities.

In this paper, we are concerned with only multiplex networks, and we are interested in only communities in individual layers. So, while the works by (Barigozzi, Fagiolo, and Mangioni 2011; Berlingerio, Pinelli, and Calabrese 2013) are related, they are not directly applicable.

However, the major difference of our problem from the other works is that our objective is not to find the communities, but rather to develop an algorithm to explore a multilayer network such that we can obtain a sample that is representative of a layer of interest. As such, we deal with missing information in our work, and as far as we know no other research on multiplex (or multilayer) networks has addressed this specific problem.

3 Problem Definition

Assume that we have an multiplex network, for example Twitter. In the Twitter multiplex network, there are different layers corresponding to different interactions such as friendship, mention, reply and retweet. There are different API rate limits for these layers. So, it is more time consuming to get all the neighbors of a user in the friendship layer compared to the other layers. Under this condition, if we have a limited amount of time (i.e. budget) and we want to find the community structure in the friendship layer, how can we select the nodes to query for neighbors?

Let $M = (L_0, L_1, \ldots, L_i)$ be a multiplex network where $L_i = (V, E_i)$ are the individual layers. Assume that, initially, we have $V_0 \subset V$ such that $|V_0| \ll |V|$, and we want to find the community structure in a layer of interest, say $L_0$.

Assume that we can explore more of $M$ only through crawling. Let us denote the set of nodes found so far by $V^S$. Since the query method is crawling, we can only query for the neighbors of $u \in V^S$ in any layer. Let $\Gamma(u, L_x)$ denote the neighbors of $u$ in layer $L_x$.

Let the ground truth community in $L_0$ be $K_G$ and the estimated community through sampling is $K_S$. In our work, the estimated community is simply the community found in our sample of $L_0$ with the nodes observed but not queried filtered out. Assume that the cost of a query in layers $L_0, L_1, \ldots, L_i$ are $c_0, c_1, \ldots, c_i$ respectively. Let $C$ be the sequence of costs of each query during the sampling process.

Let the ground truth community in $L_0$ be $K_G$ and the estimated community through sampling is $K_S$. In our work, the estimated community is simply the community found in our sample of $L_0$ with the nodes observed but not queried filtered out. Assume that the cost of a query in layers $L_0, L_1, \ldots, L_i$ are $c_0, c_1, \ldots, c_i$ respectively. Let $C$ be the sequence of costs of each query during the sampling process.

In the case of multiplex networks, when we query for the neighbor of a node we need to specify the layer as well. The cost of a query in different layers might not be the same. Assume that the cost of a query in layers $L_0, L_1, \ldots, L_i$ are $c_0, c_1, \ldots, c_i$ respectively. Let $C$ be the sequence of costs of each query during the sampling process.

Let the ground truth community in $L_0$ be $K_G$ and the estimated community through sampling is $K_S$. In our work, the estimated community is simply the community found in our sample of $L_0$ with the nodes observed but not queried filtered out. Assume that the cost of a query in layers $L_0, L_1, \ldots, L_i$ are $c_0, c_1, \ldots, c_i$ respectively. Let $C$ be the sequence of costs of each query during the sampling process.

In this section we describe the evaluation we use for comparing the similarity between two sets of communities. Assume we have two sets of communities $K_S$ and $K_G$, such that $K_G$ is the ground truth community and $K_S$ is the community in the sample.

Since the goal of our algorithm is to obtain a representative sample (Maiya and Berger-Wolf 2010), the evaluation need to take into account both the community quality, $\psi_q(K_S, K_G)$, and community representation, $\psi_r(K_S, K_G)$.

The community quality $\psi_q(K_S, K_G)$ measures if the nodes grouped together in the same community in $K_S$ are also grouped together in $K_G$. To measure this, any standard measure of cluster quality (Amigó et al. 2009) can be used. We use Normalized Mutual Information because it is unaffected by the number of clusters (Strehl and Ghosh 2002).

Since, $K_S$ is generated from the sample network, $\bigcup_{k \in K_S} k \subseteq \bigcup_{k \in K_G} k$. So, we need to consider only the nodes that are in both $K_S$ and $K_G$. Equation 1 gives the definition of community quality.
ψ_q(K_S, K_G) = NMI(K_S, \{k_x \cap \bigcup_{k_y \in K_S} k_y : k_x \in K_G\}) \tag{1}

The community representation ψ_r(K_S, K_G) measures how many communities in K_G are represented in K_S.

Let us define a many-to-one mapping \( \beta : K_S \rightarrow K_G \) such that \( \beta(x) = y \implies |x \cap y| \geq |x \cap z| \forall z \in K_G \). Then, the community representation \( \psi_r(K_S, K_G) \) is given by equation 2.

\[
|\{y \in K_G : \exists x \in K_S, \beta(x) = y\}| \tag{2}
\]

We combine \( \psi_q(K_S, K_G) \) and \( \psi_r(K_S, K_G) \) by taking the harmonic mean to give us the evaluation function \( \Psi(K_S, K_G) \).

\[
\Psi(K_S, K_G) = \frac{2 \cdot \psi_q(K_S, K_G) \cdot \psi_r(K_S, K_G)}{\psi_q(K_S, K_G) + \psi_r(K_S, K_G)} \tag{3}
\]

Since, \( 0 \leq \psi_q(K_S, K_G) \leq 1 \) and \( 0 \leq \psi_r(K_S, K_G) \leq 1 \),

\[
0 \leq \Psi(K_S, K_G) \leq 1 \tag{4}
\]

The evaluation metric used here is equivalent to the one used in (Maiya and Berger-Wolf 2010) (because the objective is similar). In our case we do not use partition distance to measure the community quality because partition distance is harder to normalize (i.e. to find the theoretical maximum). In (Maiya and Berger-Wolf 2010), they do not go into details on how they computed their community representation. However, the one we present in Equation 2 is equivalent.

A lot of works on community detection also uses Jaccard similarity to evaluate. However, it is not applicable in our work because the community detection algorithm is fixed, and only the sample is different.

As an example, consider a ground truth community, \( K = \{k_0, k_1, \ldots, k_n\} \), and communities from two different samples, \( K^1 = \{k_0^1, k_1^1, \ldots, k_i^1\} \) and \( K^2 = \{k_0^2, k_1^2, \ldots, k_i^2\} \) such that \( k_1^2 \subset k_1, k_i^2 \subset k_i \) and \( |k_1^2| < |k_i^2| \). Then, the Jaccard similarity will give higher score to the community from the larger sample, even though both matches the community structure of the ground truth exactly. So, Jaccard similarity is not a suitable measure in this work.

\[
J(K_1, K) = \frac{1}{L} \sum_{i \in [0, l]} \frac{|k_i^1 \cap k_i|}{|k_i^1 \cup k_i|} = \frac{1}{L} \frac{|k_1^1|}{|k_1^1|} \tag{5}
\]

\[
J(K_2, K) = \frac{1}{L} \sum_{i \in [0, l]} \frac{|k_i^2 \cap k_i|}{|k_i^2 \cup k_i|} = \frac{1}{L} \frac{|k_1^2|}{|k_1^2|} \tag{6}
\]

So,

\[
J(K_1, K) < J(K_2, K) \tag{7}
\]

The inequality 7 shows that the jaccard similarity will give higher score to the community from the larger sample, even though both matches the community structure of the ground truth exactly. So, Jaccard similarity is not a suitable measure in this work.

## 4 Methodology

In this section, we describe our algorithm (MultiComSample) in details. Before go into the main algorithm, we will first provide an explanation about the major components of the algorithm.

Assume that \( V^S \) is the set of nodes observed so far, and \( Q_x \) is the set of nodes on which we have queried for neighbors in layer \( L_x \). Also assume that \( L^*_x \) is the sample of layer \( L_x \) obtained so far.

Our algorithm consists of two stages - sample layers \( L_{x \in [0, l]} \) through random sampling (RndSample) to get \( \hat{L}^S_{x \in [0, l]} \), and sample \( L_0 \) with multi-armed bandits (MabSample) to get \( \hat{L}^S_0 \). We describe the RndSample in Section 4.1 and MabSample in Section 4.2. In Section 4.3 we describe how we bring together these components in MultiComSample.

### 4.1 Random Sampling on \( L_{i>0} \) (RndSample)

Assume that the budget allocated for layers \( L_{x \in [0, l]} \) are represented by \( C_{x \in [0, l]} \). (Section 4.3 describe how the budgets are allocated.)

For each layer \( L_x \), the algorithm starts from some node \( u \in V^S \setminus Q_x \) chosen at random. At each step, with probability \( (1 - \alpha) \) a node from \( \Gamma(u, L_x) \setminus Q_x \) is randomly selected as the next node to query on. With probability \( \alpha \) a random nodes from \( V^S \setminus Q_x \) is selected as the next node to query on. Then, \( L_x, Q_x \) are updated with the new nodes and edges discovered. These steps continues as long as the total cost consumed during queries is less than \( C_x \).

### 4.2 Sampling \( L_0 \) with Multi-armed Bandit (MabSample)

Consider a set of nodes \( E' \) (Equation 8) such that it is the set of all edges such that \( (u, v) \in E' \implies (u, v) \in L_0 \).

\[
E' = \{(u, v) : u \in Q_0, v \in V^S, (u, v) \in E_0\} \tag{8}
\]

Then consider the set of nodes \( E'' \) from Equation 9. \( E'' \) is the set of edges observed in \( L_{x \in [0, l]} \), excluding the ones involving nodes in \( Q_0 \). So, \( E'' \) is the set of edges in \( L_{x \in [0, l]} \) which may or may not exist in \( L_0 \).

\[
E'' = \{(u, v) : u \in V^S \setminus Q_0, (u, v) \in \bigcup_{i \in [1, l]} E_i\} \tag{9}
\]

During the budget allocation (Section 12), more budgets are allocated to the layers that have been found to have more edges in common with \( L_0 \). So, \( E'' \) should consist of more edges that actually exist in \( L_0 \). So the sample \( \hat{L}^S_0 \) is initialized as \( (V^S, E''_0) \) where \( E''_0 \) is given by Equation 10.

\[
E''_0 = E' \cup E'' \tag{10}
\]

Now we need to “clean up” the edges between node \( u, v \in V^S \setminus Q_0 \) in \( 
\hat{L}^S_0 \). The set \( E'' \) could have added some edges that do not actually exist in \( L_0 \) or there could be edges that are missing. However, we do not need to find all the missing
and extra edges. Since the objective is to find the community structure, we need to find only those that changes the community structure. To do this, we use a three level multi-armed bandits.

The bandits we use are \( \epsilon \)-decreasing, i.e. with probability \((1-\epsilon)\) it selects the best arm, and a random arm otherwise. In all the bandits, the mean of the last \( w \) rewards is considered as the expected reward for each arm.

Assume that \( C_0 \) is the budget allocated for this step. The first bandit is the layer bandit (LBandit) which selects a layer \( L_x \), \( \epsilon \in [0, 1] \). Then the community bandit (CBandit) selects a community \( k_y \) in layer \( L_x \), and then the role bandit (RBandit) selects a node \( u \in k_y \setminus Q_0 \).

The neighbors of \( u \) in \( L_0 \) are queried, and \( Q_0, E_0, V_0, V_0^S \) are updated. These steps are repeated as long as the cost of the queries is less than \( C_0 \). Figure 1 shows the flow chart of the MabSample algorithm.

The different bandits are described in more details below.

**Layer Bandit**: The first bandit, referred to as LBandit, has \( \{L_i : i \in [0, \ell]\} \) as the arms. In LBandit, the layer importance (Section 4.4) is used as the reward. The layer importance is a measure of how similar the layer is to \( L_0^S \). In this paper, the layer importance of layer \( L_x \) is denoted by \( \Lambda(L_x) \).

For a layer \( L_x \), if the layer importance is \( \Lambda(L_x) \), it means that the probability of an edge \( \exists e \in E_x^S : e \in E_0 \) is \( \Lambda(L_x) + 1 \) (more details in Section 4.4). So, if \( \Lambda(L_x) \) is high, the information from \( L_x \) is more reliable. That is why we want the layer bandit to select the arm with higher layer importance more frequently.

**Community Bandit**: Once a layer \( L_x \) has been selected by the layer bandit, CBandit selects the community in \( L_x^S \) in which we should look for the node to query on. Each layer has its version of CBandit and it has all the communities in the sample of that layer as the arm.

Assume CBandit selects community \( k_y \) and RBandit selects node \( u \in k_y \). Let the communities of \( L_0^S \) initially be \( K' \). After querying for the neighbors of \( u \), and updating \( L_0^S \), assume that the communities is now \( K'' \). Let \( \Delta(K', K'') \) be the partition distance between \( K' \) and \( K'' \) (Section 4.5).

If \( \Delta(K', K'') \approx 0 \) the query on \( u \) does not help us in the "clean up". Conversely, if \( \Delta(K', K'') \) is high, the query on \( u \) found new edges are important the community structure. So, we want to query on more nodes close to \( u \), and give \( \Delta(K', K'') \) as the reward to the arm \( k_y \) in CBandit.

**Role Bandit**: After the layer \( L_x \) and community \( k_y \) has been selected by LBandit and CBandit respectively, RBandit selects the node \( u \in k_y \setminus Q_0 \) to query for neighbors in \( L_0 \). Similar to the community bandit, each layer has its own version of RBandit, and the arms RBandit are pre-defined roles of nodes (such as high degree, high clustering coefficient etc).

Again similar to the case of CBandit, we want to find the role such that nodes with that role if queried in \( L_0 \) results in greater partition distance. So, the partition distance is used as the reward for RBandit as well.

**Hierarchical Community Bandit**: If the number of communities in \( L_x^S \) is large, the algorithm will take a long time to stabilize to a good combination of layer, community and role. So we modify CBandit to use the community hierarchy (Vieira et al. 2014) instead of all the communities in such cases.

In this modification, CBandit initially has two arm - the two communities at the highest level. At each step of the sampling process, the partition distances of the two arms are compared. If one arm is found to have significantly higher partition distance, it means that querying for nodes in that community gives more edges that are important to the community structure. So, the two children of that community in the community dendrogram are selected as the new arms of the community bandit.

If the two partition distances of the two arms are very low, it means that the two communities are not contributing edges important to the community structure. So, the parent of the nodes and the sibling of the parent are selected as the new arms so that new areas can be explored.

### 4.3 MultiPlex Network Community Sampling (MultiComSample)

In this section, we bring together RndSample and MabSample described in Section 4.1 and 4.2 respectively to describe the complete algorithm (MultiComSample).

If the total budget allocated is \( C_{\max} \), we split it into \( \eta \) equal slices, \( C' = \frac{C_{\max}}{\eta} \). The parameter \( \eta \) determines how many times the algorithm will switch between RndSample and MabSample.

For the first iteration, the budget \( C' \) is divided such that all the layers get enough budget to query the same number of nodes. That is,
4.4 Reward Function: Layer Importance

Let the importance of layer $L_x$ be $\psi(x)$. Consider a sampled layer $L^S_x$. The ratio of edges that exist in $L^S_x$ and also exist in $L_0$, to the total number of edges in $L^S_x$ is given,

$$\rho(L^S_x) = \frac{|\{(u,v): u \in Q_0, v \in V^S, (u,v) \in E^S_0 \cap E^S_x\}|}{|E^S_x|}$$  \hspace{1cm} (13)

Assume $L^S_{x, rand}$ is a random layer with the same number of nodes and edges as $L^S_x$. Then the layer importance of $L_x$ is defined as,

$$\Lambda(L_x) = \frac{\rho(L^S_x) - \rho(L^S_{x, rand})}{\rho(L^S_{x, rand})}$$  \hspace{1cm} (14)

4.5 Reward Function: Partition Distance

As mentioned in Section 4.2, $C$Bandit and $R$Bandit uses the partition distance as the rewards. Consider two communities $K_x$ and $K_y$. Let us define a function $d(K_x, K_y)$ as the minimum number of nodes that needs to be deleted from $K_x$ and $K_y$ so that they are identical (Gusfield 2002).

We then, normalize $d(K_x, K_y)$ to get the partition distance between $K_x, K_y$,

$$\Delta(K_x, K_y) = \frac{d(K_x, K_y)}{\max(\sum_{k \in K_x} |k|, \sum_{k \in K_y} |k|)}$$  \hspace{1cm} (15)

5 Experimental & Results

In this section we describe the datasets we use for our experiments (Section 5.1), the experimental setup (Section 5.2) and the results (Section 5.3).

5.1 Datasets

In our experiments, we use two sets of data. The first set is the Twitter multiplex network, and consist of three layers - friends, mention and retweet. The second set of networks are co-authorship networks, and the layers are publications in different research areas.

The Twitter networks we use to test our algorithms were collected using the Twitter API. Each networks consist of three layers - Friend, Mention and Reply. The Friend layer is collected using the Friends and Followers API and converting them into undirected edges. The Mention and Reply layers were collected using the Timeline API. We used random walk to select Twitter users to run API queries on.

If user $u_0$ follows or is followed by user $u_1$, there is an edge between them in the Friend layer. For the Mention layer, users $u_0$ and $u_1$ have an edge between them if $u_0$ mentions $u_1$ in a tweet, or $u_1$ mentions $u_0$. Similarly, there is an edge between $u_0$ and $u_1$ in the Reply layer if $u_0$ replied to a tweet by $u_1$, or $u_1$ reply to an edge by $u_0$.

The users in our networks are only those on whom we have run all the API queries on. This is done to ensure that if an edge is missing in our network, it is actually missing in the real Twitter network. Table 1 shows the number of nodes and edges in the different layers of the multiplex networks used for experiments.

| Network       | $|V|$ | $|E_0|$ | $|E_1|$ | $|E_2|$ |
|---------------|------|--------|--------|--------|
| Twitter-KP    | 2420 | 7262   | 2447   | 1055   |
| Twitter-OW    | 2182 | 7322   | 2514   | 1046   |
| Twitter-SC    | 2116 | 7943   | 2436   | 915    |
| Twitter-TR    | 3036 | 9123   | 3071   | 1421   |
| Ca-AstroPh-CondMat | 3197 | 5733   | 3707   |        |
| Ca-HepPh-HepTh | 1324 | 1585   | 1395   |        |

Table 1: Number of nodes and edges in the different layers of the multiplex networks used for experiments.
nodes, and edges in different layers of the networks used for our experiments.

In our experiments, we consider the Friends layer to be the expensive layer. Mentions and Replies layers are considered to be the cheap layers. This is consistent with the API rate limits in Twitter.

The co-authorship networks we use were downloaded from (Leskovec, Kleinberg, and Faloutsos 2007). The first network consists of ca-HepPh and ca-HepTh, and the second consist of ca-AstroPh an ca-CondMat as the different layers. We consider only the nodes that exist in both layers and the edges between them to create the multiplex networks. The layers ca-HepPh and ca-AstroPh are considered as the expensive layers. In these networks, an edge between two node means that the two persons have co-authored a paper in the research area represented by that layer.

5.2 Experimental Setup & Baseline Algorithms

In our experiments, set a maximum budget ($C_{\text{max}}$) of 2000, and the costs of a query in $L_0$ is set to 1, and that on $L_{x \in [0, \ell]}$ are set to 0.5 each. If we consider the real world case of Twitter, the ratio of $c_0$ to $c_{x \in [0, \ell]}$ is 1 : 100. So, the budget we use here is much stricter - and the performance on the real world budget ratio would not be worse that the one we consider.

The only parameter input the algorithm takes from the user is the number of slices ($\eta$), and we set it to 10. If the value of $\eta$ is too low, our algorithm will invest too much of the budget in sampling the cheaper layers during RndSample, without knowing which are the important layers to sample more on. On the other hand, if it is too high, there will not be enough time for MabSample to figure out a good combination of arms. We set the value of $\eta$ to 10 in our experiments, after experimenting with a few values. We leave the problem of finding an optimum value of $\eta$ as a future research direction.

As mentioned in Section 2, we are not aware of any works that address the problem we tackle in our paper. So to compare the performance of our algorithm, we used Max Degree Sampling, Random Walk, and Random Walk on aggregate layers.

The Max Degree baseline we use is equivalent the the MOD algorithm (Avrachenkov et al. 2014). This baseline operates on only $L_0$, and allocates all its budget into querying nodes in $L_0$. In the rest of the discussion we refer to this baseline as MaxDegree.

The Random Walk baseline also operates on only $L_0$. At each step, with probabilit $1 - \alpha$ it select a neighbor of the current node that has not been queried on as the next node to query on. With probability $\alpha$, it select an observed but not queried node as the next node. We use RandomWalk to refer to this baseline in the following discussion.

For the baseline, Random Walk on aggregate layers (Aggregate), we aggregate all the layers $L_{x \in [0, \ell]}$ into a single layer and perform random walk on this layer. Ideally, if the the layers $L_{x \in [0, \ell]}$ are very highly correlated with $L_0$ this should be equivalent to RandomWalk. At each step, the amount of budget consumed is $\sum_{x \in [0, \ell]}$.

It is worth pointing out that, since MaxDegree and RandomWalk uses all the budget on $L_0$, it queries on more nodes in $L_0$ compared to our algorithm MultiComSample.

5.3 Experimental Results

Figure 3 shows the experimental results. We perform each experiment 10 times, and the lines represent the mean and the shaded area is the standard deviation. In the figure black, green, red and blue lines represent the results from MultiComSample, MaxDegree, RandomWalk and Aggregate respectively.

In all the figures, the plots for MultiComSample do not start at 0, rather they start slightly after 0. This is because MultiComSample performs RndSample to initially.

From Figures 3b, 3c we can observe that the plots of MultiComSample and MaxDegree converges. This is due to the size of the layer $L_0$ for these graphs - at cost 2000, almost all the nodes in $L_0$ has been queried by MaxDegree. Overall, it can be observed that MultiComSample outperforms the baselines in all the networks considered.

6 Conclusion

In this paper, we introduced the problem of sampling for the community structure in a layer of interest in a multiplex network under the condition that there are different costs associated with a query on each layer.

We propose an algorithm that switches between sampling the cheaper layer through random walk, and sample the expensive layer through multi-level multi-armed bandits. The algorithm we propose (MultiComSample) is easy to tune as there is only one tunable parameter - $\eta$ the number of slices.

We tested our algorithm on six real world multiplex networks. Four of the networks are Twitter networks and two are co-authorship networks. We found that MultiComSample outperforms all the baseline algorithms consistently.

There are a lot of scope for more works in this area. Of line of work is to consider multiplex communities instead of communities on individual layers. If we want to generate a sample that is representative of the multiplex community, how do we change the algorithm to accommodate for that?

Another direction of future research is in determining the ratio of $c_0$ to $c_{x \in [0, \ell]}$, at which the trade-off of spending some budgets to query on the cheap layer to spending all the available budget on the expensive layer become impractical. Yet another direction is determining the optimum value of $\eta$, the number of slices used by MultiComSample.

7 Acknowledgements

This research was supported in part through computational resources provided by Syracuse University.

References

Figure 3: Comparison of MultiComSample against various baselines. The y-axis represents $\Psi(\cdot, \cdot)$, and the x-axis is the amount of the budget consumed at that point. The black line represents the results from MultiComSample. The green, red and blue lines represent the results from MaxDegree, RandomWalk and Aggregate. It can be observed that MultiComSample outperforms all the baselines by a significant margin.


