Crawling the Community Structure of Multiplex Networks

Ricky Laishram
Syracuse University
Syracuse NY, USA
rlaishra@syr.edu

Jeremy D. Wendt
Sandia National Laboratories
Albuquerque, NM, USA
jdwendt@sandia.gov

Sucheta Soundarajan
Syracuse University
Syracuse NY, USA
susounda@syr.edu

Abstract

We examine the problem of crawling the community structure of a multiplex network containing multiple layers of edge relationships. While there has been a great deal of work examining community structure in general, and some work on the problem of sampling a network to preserve its community structure, to the best of our knowledge, this is the first work to consider this problem on multiplex networks. We consider the specific case in which the layers of a multiplex network have different query (collection) costs and reliabilities; and a data collector is interested in identifying the community structure of the most expensive layer. We propose *MultiComSample* (`MCS`), a novel algorithm for crawling a multiplex network. `MCS` uses multiple levels of multi-armed bandits to determine the best layers, communities and node roles for selecting nodes to query. We test `MCS` against six baseline algorithms on real-world multiplex networks, and achieved large gains in performance. For example, after consuming a budget equivalent to sampling 20% of the nodes in the expensive layer, we observe that `MCS` outperforms the best baseline by up to 49%.

1 Introduction

In this paper, we examine the problem of collecting data by crawling a *multiplex network*, in which different types of edges represent different types of relationships, with the end goal of finding the community structure in a layer of interest. In particular, we consider the case where one can query nodes to obtain some or all of their neighbors (e.g., through an API), but queries on the layer of interest are costly (in terms of time, money, or resources) relative to other layers.

A *multiplex network* is a type of multilayer network where the nodes participate in multiple types of interactions, or layers (Mucha et al. 2010; Lee, Min, and Goh 2015). For example, communication between a group of people can be represented as a multiplex network, where the different layers represent the modes of communication – email, face-to-face, phone calls, etc. In order to study the community structure of such networks, one must first collect appropriate data, usually through a network crawling algorithm. For optimal analysis results, the sample that the algorithm generates should be representative of the community structure of the original network.

Collecting data from multiplex networks is a practical and important problem, and raises challenges that one does not encounter in the single-layer case. In particular, data collection costs are unlikely to be uniform across layers: for example, when observing a ‘dark’ criminal network, family relationships may be easily obtained, but face-to-face interactions are much harder to observe. Additionally, certain layers may be more reliable than others: for example, individuals will likely not recall all of their personal interactions, but may keep a complete record of their e-mail communications. Most existing work on network data collection focuses on only one type of interaction, or aggregate multiple types of interactions into one, discarding potentially useful information.

We consider two problem settings based on the query response. In the *Reliable Query Response (RQR)* model, we assume that when a node is queried in a particular layer, we obtain all of its neighbors in that layer. In the *Unreliable Query Response (UQR)* model, when we query a node, each neighbor has some probability of being excluded.

The challenges associated with this problem are: (1) The layer of interest is costly to explore, so queries on this layer must be made wisely. (2) The network can be explored only through crawling. Thus, we do not know the true properties of many nodes. (3) Initially, none of the layers are fully explored. Hence, one must balance the trade-off between exploring a cheap layer, which may or may not be correlated with the expensive layer, vs. querying the expensive layer.

We propose *MultiComSample* (`MCS`), a multi-armed bandit-based algorithm that uses queries on the cheaper layers to guide a network crawler in exploring the more expensive layer of interest. With a budget equivalent to sampling 20% of the nodes, `MCS` outperforms the best baseline by up to 25% in the case of *RQR*, and 49% for *UQR*.

The main contributions of this paper are:

1. To the best of our knowledge, we are the first to consider the problem of crawling a multiplex network with the goal of generating a sample that is representative of the underlying community structure of the layer of interest.
2. We propose `MCS`, a novel sampling algorithm for crawling the community structure of multi-layer networks.
3. We perform extensive evaluations, and demonstrate that `MCS` outperforms a wide variety of baseline algorithms in crawling the community structure of the layer of interest.

2 Related Works

There has been a significant amount of work on community detection in single layer networks (Fortunato 2010; Girvan and Newman 2002; Blondel et al. 2011), but comparatively little for the case of multilayer networks. One reason for this is that generalizing single layer community detection methods to the multilayer setting is often nontrivial (Kivelä et al. 2014). A huge problem in the development of a community detection algorithm is the lack of a null model that takes the inter-layer edges into consideration. Thus, most work has been on special types of multilayer networks.

Mucha *et al.* (Mucha et al. 2010) generalized community detection methods to a special class of multilayer networks called multi-slice networks, by generalizing modularity to consider node-tuple pairs. Tang *et al.* (Tang, Wang, and Liu 2009) approached the problem of community detection in multi-dimensional networks by applying multi-objective optimization to maximize modularity. Other works begin with community detection in the individual layers (Barigozzi, Fagiolo, and Mangioni 2011; Berlingerio, Pinelli, and Calabrese 2013). Berlingerio *et al.* (Berlingerio, Pinelli, and Calabrese 2013) defined a multiplex community as a closed frequent item-set where each item is a tuple of the layer and the community assigned to the node.

In this paper, we are concerned only with communities in individual layers. Thus, while the works by (Barigozzi, Fagiolo, and Mangioni 2011; Berlingerio, Pinelli, and Calabrese 2013) are related, they are not directly applicable. Moreover, these works do not discuss crawling, but instead describe techniques for finding communities in an existing network.

Crawling single layer networks has been extensively studied. In particular, when the network is too large to analyze as a whole, a sample of the original network is usually used. 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. This method, however, downsamples from a fully-known original network, rather than crawling an unobserved network. Domenico *et al.* (De Domenico et al. 2014) have explored the effect of performing a random walk on a multiplex network. They discovered that multiplex networks can be more efficiently explored by random walk compared to single layers. In this work, they assume that the different layers have the same cost of exploration. Wendt *et al.* (Wendt et al. 2016) analyzed the problems associated with generating a multiplex sample when the layers have different costs. They explored the Twitter multiplex network through API queries, and observed that naive algorithms like BFS produces samples in which the inter-layer correlations are affected.

The major difference between our work and existing work is that our objective is not to find the communities, but rather to develop an algorithm to explore a multiplex network such that we can obtain a sample that is representative of the community structure 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.

Notation	Description
$L_x = \langle V, E_x \rangle$	Layer x in the original network
$L_x^S = \langle V^S, E_x^S \rangle$	Sample of L_x after S units of budget used
K_x	True communities in L_x
K_x^S	Observed communities in L_x^S
$\Psi(\cdot, \cdot)$	Community similarity between X and Y
Q_x	Set of nodes queried in L_x
$\Gamma(u, L_x)$	Neighbors of u in layer L_x
$\Gamma'(u, L_x)$	Neighbors of u in layer L_x returned in *UQR*
c_x	Cost of a query in L_x
c_Σ	Sum of c_x for all layers
$\mathcal{C}$	Sequence of query costs during sampling
B_x	Budget allocated for layer x
B	Total budget allocated
β_x^u	Uncertainty factor of u in L_x
$(\beta_x^\mu, \beta_x^\sigma)$	Uncertainty factor of L_x
$\Delta_{x,y}$	Community update distance of x w.r.t. y
$\Lambda_{x,y}$	Edge overlap of x w.r.t. y

Table 1: Notations used in this paper.

3 Problem Definition

Let $M = \langle L_0, \ldots, L_l \rangle$ be a multiplex network where $L_i = \langle V, E_i \rangle$ are the individual layers. The layers corresponds to various types of interaction, and have different exploration or query costs. (For example, in the Twitter network, layers correspond to interactions such as friends, mentions, and replies; each of which has a different API rate limit[1].) Under these conditions, if we have a limited budget (time, money etc.) and want to generate a sample that is representative of the community structure of a layer of interest how can we select the best nodes to query?

Assume that initially, we know $V' \subset V$ such that $|V'| \ll |V|$, and we want to find the community structure in the layer of interest, L_0. Initially, no edges are known; the nodes in V' serve only as starting points to begin exploration. Also assume that the similarity between the true and observed communities of L_0 is measured with $\Psi\left(K_0^S, K_0\right)$ (Table 1).

We assume that the network can be explored only through crawling[2]; i.e., we can query for neighbors of a node that has already been observed. Each query is performed on a node specifying a particular layer; and returns some or all of the neighbors of that node in the specified layer.

We consider the following problem: *Given an initial set of nodes V', query budget B, and layer of interest L_0, how can we sample M through crawling so that $\Psi(K_0^B, K_0)$ is maximized without exceeding the given query budget*?

In this paper, we consider only the scenario where L_0 is the most expensive layer to query; if this were not the case, we could ignore the more expensive layers in favor of querying L_0 directly. We also make the assumption that the edges are undirected; otherwise, we can extend as demonstrated in (Laishram, Areekijseree, and Soundarajan 2017).

We consider two types of responses in return to a neighborhood query on a node:

[1]We can query the friends layer 15 times every 15 minutes, but for the mention and reply layers, the API allows 1500 queries every 15 minutes

[2]This is a common constraint in many real world networks.

Reliable Query Response: In *RQR*, a query for the neighbors of node in a particular layer returns all neighbors of that node in that layer. This is generally the case when we have a database to query against, and information is accurately available. For example, in an email communication network, we can lookup all the email communications of a node[3].

Unreliable Query Response: In the case of *UQR*, there is a *node uncertainty factor* (β_x^u) associated with node u in layer L_x. A query on u in layer L_x returns $\Gamma'(u, L_x) \subseteq \Gamma(u, L_x)$ where for every $v \in \Gamma(u, L_x)$, there is a probability β_x^u that $v \in \Gamma'(u, L_x)$. As an example, if we interview people to gather data of face-to-face communication, it is unlikely that everyone will remember all of their interactions.

We assume that for each layer L_x, the node uncertainty factor follows a normal distribution with mean β_x^μ and standard deviation β_x^σ. We refer to $(\beta_x^\mu, \beta_x^\sigma)$ as the *layer uncertainty factor*. These uncertainty values are not known beforehand.

4 Methodology

In this section, we describe *MultiComSample* (`MCS`), a novel multi-armed bandit algorithm for crawling multiplex networks. We first consider the case of *RQR*, and in Section 4.5, we discuss modifications to handle the case of *UQR*.

`MCS` consists of two stages – sampling layers $L_{x\in(0,l]}$ through a random walk (`RNDSample`) to get $L^S_{x\in(0,l]}$, and using this information, sampling the layer of interest L_0 with a multi-armed bandit method (`MABSample`). We describe the `RNDSample` stage in Section 4.1 and the `MABSample` stage in Section 4.2. In Section 4.3 we describe how we bring together these components in `MCS`.

In `MCS`, we use three bandits to make the selection on which node to query in L_0. The first bandit, `LBandit`, selects which layer's information to use to guide the node selection in L_0. Each layer is associated with two additional bandits: `CBandit`, which selects a community in the layer, and `RBandit` which selects a structural role (e.g., high degree nodes) in the community.

4.1 Random Sampling on $L_{x\in(0,l]}$ (**`RNDSample`**)

Random walks have been used extensively to discover community structure in single-layer networks (Rosvall and Bergstrom 2008). Thus, in this stage, we use a random walk to separately crawl each of the layers in the network.

We allocate a small amount of budget $B_{x\in(0,l]}$ to explore layers $L_{x\in(0,l]}$ (Section 4.3). For each layer $L_{x\in(0,l]}$, the algorithm begins a random walk with jump starting (to an observed but unqueried node) at a random observed node $u \in V^S \setminus Q_x$. After each query, L_x and Q_x are updated with the new nodes and edges discovered. These steps continue as long as the total cost consumed during queries is less than B_x. Then it continues to another layer.

[3]Some APIs return only a maximum of k neighbors in response to a query; in such cases, a node can be treated as multiple nodes, each with at most k neighbors, as in (Laishram, Areekijseree, and Soundarajan 2017)

4.2 Sampling L_0 with Multi-Armed Bandits (**`MABSample`**)

In this step, we sample the layer of interest L_0 using the information from the random walk crawls of the cheaper layers. The first step in this process is to initialize L_0^S, the sample of L_0, using $L_{x\in(0,l]}$.

Initialization: Assume E' is the set of all edges that we know exist in L_0 (because they involve at least one node in Q_0, the set of nodes already queried in L_0).

Next, consider the set of edges in $L^S_{x\in(0,l]}$, excluding the ones involving nodes in Q_0. Denote this set by E'', so E'' is the set of edges observed in $L^S_{x\in(0,l]}$ which may or may not exist in L_0.

During the budget allocation, more budget is allocated to the layers that have high *edge overlap* with E'. Thus, E'' will consist of more edges that actually exist in L_0. Then, the sample L_0^S is initialized as $\langle V^S, E_0^S\rangle$ where $E_0^S = E' \cup E''$.

Further Exploration with Bandits: We need to 'clean up' the edges between node $u, v \in V^S \setminus Q_0$ in L_0^S. There might be extra edges in E'' that do not actually exist in L_0, or it could be missing edges that have not been observed. Because our goal is to sample for community structure, we do not need to check all of these edges; rather, we need to find only those that change the community structure. To do this, we use a set of three multi-armed bandits (we use ϵ-greedy bandits[4] (Katehakis and Veinott Jr 1987)).

As mentioned before, we use three bandits - `LBandit`, `CBandit` and `RBandit`. Intuitively, these three bandits together determine which cheap layer, region of that cheap layer, and type of node are most informative with respect to the community structure of the layer of interest.

The different bandits are described in further detail below. The reward functions used by the various bandits are described in Section 4.4.

Layer Bandit: The first bandit, referred to as `LBandit`, has arms corresponding to layers $L_{x\in(0,l]}$. In `LBandit`, *edge overlap* (Section 4.4) is used as the reward, measuring the similarity between the sample of a layer and observed sample of L_0. We denote the edge overlap between layer L_x^S and L_0 by $\Lambda(L_x^S, L_0)$. Because we do not know L_0, we measure the edge overlap against E'. Intuitively, we want to select those layers with high overlap more often.

In the case of unreliable query responses, the layer uncertainty factor $(\beta_x^\mu, \beta_x^\sigma)$ is also a very important consideration. Because we do not know the actual values of $(\beta_x^\mu, \beta_x^\sigma)$, we use the estimated values $(\beta_x^{\mu\prime}, \beta_x^{\sigma\prime})$ (Section 4.5) . Hence, the layer reward in this case is $\Lambda\left(L_x^S, L_0\right)\beta_x^{\mu\prime}$.

Community Bandit: Once a layer L_x has been selected by the layer bandit, `CBandit` selects the community in L_x^S in which it should look for the node to query. Each layer has its version of `CBandit`, which has communities from the sample of that layer as the arms.

Let the observed communities of L_0^S be K_0^S. Assume that after `CBandit` selects community k, a node $u \in k$ is then selected by `RBandit`. After querying on u, let L_0^{S+1} be the updated sample, and let K_0^{S+1} be the new communities.

[4]See Section 4.6 for discussion on choice of bandit algorithm.

Let $\Delta(K_0^S, K_0^{S+1})$ be the *Community Update Distance* (Section 4.4). If $\Delta(K_0^S, K_0^{S+1}) \approx 0$ the query did not help us in the 'clean up'. Conversely, if it is high, the uncertainty of the community structure is high and the query on u found new edges important to the community structure. So, we want to query more nodes close to u, and give $\Delta(K_0^S, K_0^{S+1})$ as the reward to the arm k in `CBandit`.

If the number of communities in L_x^S is large, the algorithm will take a large number of queries to stabilize to a good combination of layer, community and role. Thus, `CBandit` uses the modularity community hierarchy (Vieira et al. 2014) instead of all the communities in such cases.

Initially, `CBandit` has two arms – the two communities at the highest level. At each step the partition distances of the two arms are compared. If one arm is found to have significantly higher past rewards, it means that querying 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 two new arms of the community bandit. If the past rewards of the two arms are very low, 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.

Role Bandit: Each layer has its own `RBandit`, where different structural roles are the arms. In our implementation, the roles used are predefined as maximum and minimum degree, clustering coefficient and betweenness centrality. Other types of roles (Henderson et al. 2012) can also be used here[5].

A set of candidate nodes κ is generated. For ***RQR***, the set of candidate nodes is given by $\kappa = V^S \setminus Q_0$. In ***UQR***, there might be nodes Q_0 that have unobserved neighbors. Suppose ρ_0^u is the probability that u has an unobserved neighbor in L_0 (Section 4.5). Then, we generate a set $Q_0' \subseteq Q_0$ such that $u \in Q_0'$ with probability $(1 - \rho_x^u)$. Hence, $\kappa = V^S \setminus Q_0'$.

After the layer L_x and community $k \in K_x^S$ have been selected by `LBandit` and `CBandit`, respectively, `RBandit` selects a role r. Then it selects a node from $\kappa \cap k$ that satisfies the role r and returns it to be queried in L_0.

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 a sample whose community update distance is high. Thus, $\Delta(K_0^S, K_0^{S+1})$ is used as the reward for arm r in `RBandit`.

4.3 MultiComSample (`MCS`)

In this section, we bring together `RNDSample` and `MABSample`, described in Section 4.1 and 4.2, respectively, to describe the complete algorithm (`MCS`).

If the total budget allocated is B, `MCS` allocate some portion, B', of the budget for the first iterations of `RNDSample` and `MABSample`. B' is divided so that all the layers get enough budget to query the same number of nodes.

`MCS` performs `RNDSample` in $L_{x\in(0,l]}$ with the budgets B_x, and then `MABSample` is performed in L_0.

After running `MABSample`, we have the layer edge overlaps (Section 4.4), as used by `LBandit`. Because the layers with higher edge overlap are more similar to L_0, we want to allocate more budget to these layers. So, intuitively, we want $B_x \propto \Lambda(L_x)$. On the other hand, allocating more budget to the layers that are cheaper to query gives a greater quantity of data. That is, it is desirable for $B_x \propto \frac{1}{c_x}$. Thus, we allocate the budgets for $L_{x\in(0,l]}$ as

$$B_x = \frac{\Lambda(L_x)/c_x}{\sum_{i\in[0,l]} \Lambda(L_i)/c_i} B'. \tag{1}$$

This process repeats until all the budget has been used up.

4.4 Reward Functions

Because we do not know K_0, we cannot use $\Psi(K_0^S, K_0)$ directly as the reward functions in `MCS`. So we use the edge overlap and community update distance as "stand-in" reward functions.

Edge Overlap: The edge overlap for layer L_x measured against layer L_y is the ratio of ratio of edges that exist in L_x^S and in L_y^S, to the total number of edges in L_x^S. That is

$$\Lambda_{x,y} = \frac{\left|\left(E_0^S \cap E_x^S\right) \cap \left(Q_0 \times V^S\right)\right|}{\left|E_x^S \cap \left(Q_0 \times V^S\right)\right|}. \tag{2}$$

In `MCS` since the layer of interest is L_0, we use $\Lambda_{x,0}$ as the reward for `LBandit`.

Community Update Distance: Consider two community partitions K, K'. We define the community update distance as the normalized partition distance between K, K'

$$\Delta_{K,K'} = \frac{\delta\left(K, K'\right)}{\max\left(\sum_{k\in K}|k|, \sum_{k\in K'}|k|\right)} \tag{3}$$

where $\delta(\cdot,\cdot)$ is the partition distance (Gusfield 2002).

Let K_x^S, K_x^{S+b} be the communities found in L_x^S and L_x^{S+b} (that is after using up b additional queries). In `MCS`, we use $\Delta_{K_x^S, K_x^{S+b}}$ as the the reward for `CBandit` and `RBandit` in layer L_x.

4.5 Estimating Uncertainty Factors

In the case of the ***UQR***, we need to estimate the node uncertainty factor β_x^u and layer uncertainty factor $(\beta_x^\mu, \beta_x^\sigma)$.

Let us denote the number of times u has been queried in layer x by q_x^u, and let $r_x^{u,v}$ be the number of times node v was included in the response to a query on node u in L_x. The selection of nodes for inclusion in $\Gamma'(u, L_x)$ are independent of each other. So, we can estimate β_x^u as

$$\hat{\beta_x^u} = \frac{1}{|\Gamma(u, L_x^S)|} \sum_{u\in\Gamma(u,L_x^S)} \frac{r_x^{u,v}}{q_x^u}. \tag{4}$$

We can then calculate ρ_x^u, the probability that u still has unobserved neighbor in L_x, as $\rho_x^u = \left(1 - \hat{\beta_x^u}\right)^{q_u^x}$. The value of ρ_x^u is used to determine if u is included in the candidate list for nodes to query in L_x.

For a layer L_x we estimate the layer uncertainty factors, $(\hat{\beta_x^\mu}, \hat{\beta_x^\sigma})$, as the mean and standard deviation of the uncertainty factors all the nodes in Q_x. If a node has been queried only once, its estimated uncertainty $\hat{\beta_x^u} \approx 1$. In this case, we assign it a random value from the normal distribution with mean $\hat{\beta_x^\mu}$ and standard deviation $\hat{\beta_x^\sigma}$.

[5]Because initially $|V^S| \ll |V|$, role extraction algorithms are not very accurate, so we use fixed roles.

4.6 Choice of Multi-Armed Bandit Algorithm

In our implementation of MCS, the multi-armed bandit algorithm that we use is ϵ-greedy. However, MCS is not tied to only this bandit algorithm – we can use any other algorithm.

We have replaced ϵ-greedy in MCS with ϵ-decreasing, VDBE (Tokic and Palm 2011) and UCB1 (Auer, Cesa-Bianchi, and Fischer 2002), and compared the results to ϵ-greedy. We observed that the results are not very different. So, for simplicity and efficiency, we use ϵ-greedy.

4.7 Running Time

Assume that d is the average degree and l is the number of layers. The part of MCS that takes the most time is the community detection. So, the running time of MCS is $\mathcal{O}\left(\frac{ldB\lambda}{\zeta}\log\left(\frac{dB\lambda}{\zeta}\right)\right)$, where $\lambda = \max_{x\in[0,l]}\left(\frac{\Lambda_{x,0}}{c_x}\right)$, and $\zeta = \sum_{x\in[0,l]}\left(\frac{\Lambda_{x,0}}{c_x}\right)$.

4.8 Sensitivity Analysis

The performance of MCS depends on a number of factors: (1) the edge overlap between the the different layers, (2) the number of layers, and (3) the relative cost of a query on the cheaper layers to the expensive layer.

Assume that the average degree of layer L_x is d_x. For layer L_x, let $o_x(i)$ be the expected fraction of unobserved edges on the next query on the layer, where i is the number of queries already made. If we have budget B, the number of unique edges we expect to observe with MCS is

$$\sum_{x=0}^{l}\left(d_x\Lambda_{x,0}\left(\prod_{y=0}^{x-1}(1-\Lambda_{y,x})\right)\left(\sum_{i=0}^{Bc_x/c_\Sigma}o_x(i)\right)\right). \quad (5)$$

Sensitivity to number of layers: There are two competing effects due to the number of layers. If there are few layers, the budget allocated for each layer is high. So, we expect the performance of MCS to increase initially. However, if the number of layers is high, the budget allocated for each layer is lower, and the probability of observing an edge not seen in another layer decreases. So, the performance of MCS will then drop after a while as the number of layers grows.

Sensitivity to relative layer costs: For lower relative cost, the budget allocated to each layer is higher (because c_Σ is lower). So, the performance of MCS will increase with lower relative cost.

5 Experimental Analysis

In this section, we perform extensive evaluations to compare the performance of MCS against baseline methods (Section 5.1) and oracle method (Section 5.2). We also experimentally evaluate the running time (Section 5.3) and effect of number of layers (Section 5.4). We also run MCS on the real Twitter API to investigate its scalability (Section 5.6).

The networks we use for our experiments are given in Table 2. The Twitter networks are collected using the Twitter API (Wendt et al. 2016). These networks contain 3 layers, and approximately 2k nodes and 10k edges. For the co-authorship network[6], caHepTh is considered as the expensive layer, and caHepPh is considered as the cheap layer. This network contains approximately 1.3k nodes and 1.9k edges. The NoordinTop network[7] has five layers – communication, kinship, friendship, classmates and mentors. The communication layer is considered to be the expensive layer. The NoordinTop network has 120 nodes and 750 edges across all layers. The last network we use is the DBLP dataset[8] between 2013 and 2017. There are two layers: co-authorship, which is treated as the expensive layer, and citation. We treat all edges in all these networks as undirected, and there are around 60k nodes and 300k edges.

Network	c_0	c_1	c_2	c_3	c_4	C_m
TwitterKP	1.0	0.5	0.5	-	-	50%
TwitterOW	1.0	0.5	0.5	-	-	50%
TwitterSC	1.0	0.5	0.5	-	-	50%
TwitterTR	1.0	0.5	0.5	-	-	50%
CaHepPhTh	1.0	0.5	-	-	-	50%
NoordinTop	1.0	0.25	0.25	0.5	0.5	50%
DBLP	1.0	0.5	-	-	-	5%

Table 2: Query cost and budget of the different layers of the multiplex networks used for experiments.

5.1 Performance Comparisons

In these experiments, we run MCS on the networks described and compare the performance against six baseline algorithms.

Baseline Algorithms We are not aware of any works that address the problem we tackle here. To compare the performance of MCS, we have three variants of Max Degree Sampling and Random Walk. Max Degree Sampling is known to be effective in finding the community structure in a single layer network (Maiya and Berger-Wolf 2010), and Random Walk is good at finding the dense regions of a network.

In the first set of variants, the baseline algorithms are only aware of the layer of interest. Because we do not know the true degree of the nodes, the SMD baseline is equivalent to the MOD algorithm in (Avrachenkov et al. 2014). In each iteration, the node with the highest observed degree is queried. The SRW baseline performs a random walk on only L_0, with a reset probability of 0.2.

The second set of variants are similar to the first ones, except they operate on the aggregate of all the layers. This gives the crawler more ways to get to different parts of the network. We refer to these baselines as AMD and ARW respectively.

The third category of baseline algorithms take into account the multiplex structure of the networks. At each step, a layer L_x is selected based on a selection probability $p(L_x) \propto \frac{\Lambda(L_x)}{c_x}$. Once a layer is selected, an unqueried neighbor of the current node in that layer is selected randomly (MRW) or based on the observed degree (MMD). The node is then queried for neighbors in both L_x and L_0.

[6]https://snap.stanford.edu/data/

[7]https://sites.google.com/site/sfeverton18/research/appendix-1

[8]https://dblp.uni-trier.de/

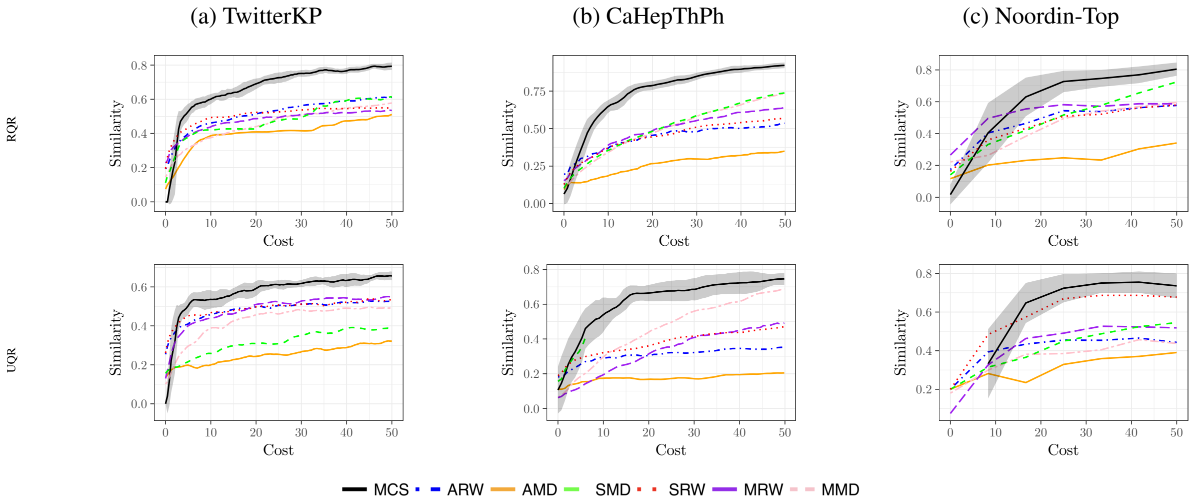


Figure 1: Comparison of `MCS` against various baseline algorithms on different networks in the case of ***RQR*** (top row) and ***UQR*** (bottom row). The y-axis is the similarity between the community structure in the sample and the ground truth, and the x-axis is the budget consumed as a percentage of the total budgets required to query all nodes in L_0.

To account for ***UQR***, we need to make a small modification to the candidate nodes selection in the baseline algorithms. The probability that a u is added to the candidate list is given by $\frac{|\Gamma'(u,L_x)\cap\Gamma(u,L_x^S)|}{|\Gamma'(u,L_x)\cup\Gamma(u,L_x^S)|}$. Nodes that have been observed, but not queried are guaranteed to be in the candidate list.

Experimental Setup We use the Louvain community detection method (Blondel et al. 2011)[9]. The budget and the layer costs are given in Table 2. We perform 10 trials of each experiment to account for randomness in sampling as well as in the Louvain method. We also keep the ordering of the nodes consistent to get rid of the effect of node orders during the community detection process.

For ***UQR***, $(\beta_x^\mu, \beta_x^\sigma)$ of all layers in the Twitter, co-authorship and DBLP networks are set to $(0.2, 0.1)$. For the NoordinTop network, it is set to $(0.2, 0.1)$ (communication), $(0.5, 0.1)$ (friendship), $(0.25, 0.1)$ (mentors), and $(0.75, 0.1)$ (kinship and classmates). (Higher values here means greater uncertainty of the information gathered from that layer.)

Evaluation To evaluate the detected communities, we must take into account community quality- i.e., whether nodes grouped together in the detected communities are also grouped together in the true communities, and community representation- i.e., how well the true communities[10] are represented by the detected communities. To measure community quality, we use Normalized Mutual Information, and to measure community representation, we use the measure presented in (Maiya and Berger-Wolf 2010), which gives a ratio of the communities found to the true communities. Finally, we report the harmonic mean of these two measures as the community similarity measure $\Psi\left(K_x^S, K_x\right)$.

Network	Query Mode	`MCS`	Best Baseline
TwitterOW	RQR	0.685	0.592 (`ARW`)
	UQR	0.607	0.498 (`MRW`)
TwitterTR	RQR	0.721	0.592 (`ARW`)
	UQR	0.625	0.529 (`SRW`)
TwitterSC	RQR	0.685	0.636 (`SMD`)
	UQR	0.618	0.556 (`ARW`)
DBLP	RQR	0.679	0.554 (`SMD`)
	UQR	0.628	0.517 (`MMD`)

Table 3: Similarity comparison between communities in L_0 and L_0^S generated by `MCS` and the best baselines algorithm at budget of 20% for different networks under ***RQR*** and ***UQR***.

Experimental Results Figure 1 shows the performance comparison between `MCS` and the various baselines for different budgets under the two query response models (***RQR*** and ***UQR***) on the TwitterKP, CaHepPhTh and NoordinTop networks. The results for the rest of the networks are presented in Table 3 due to space limitations.

We can observe that `MCS` outperforms all the baseline algorithms in all the cases, and the improvement is significant. In the case of ***RQR*** at budget of 20% we observe improvement of 9% to 25%, and 8% to 49% in ***UQR***.

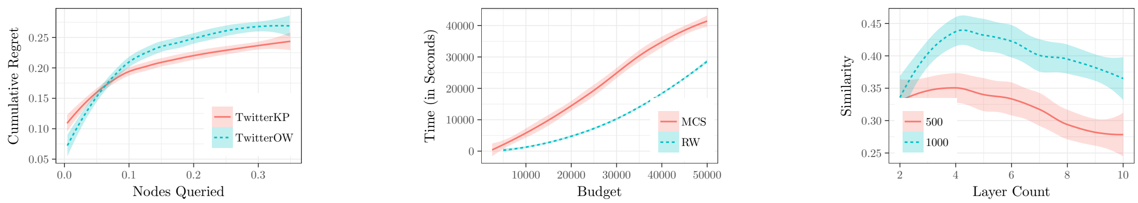
It can also be observed that there is no single best baseline method, and results vary by network structure and query model. However, `MCS` can adapt to these variations and consistently outperforms the baselines.

5.2 Regret Analysis

We evaluate the regret of the policy in `MCS` against the dynamic oracle described in (Besbes, Gur, and Zeevi 2014). The oracle has knowledge of complete network, and at each step, it knows what the next best combination of arm to maximize the similarity of K_0^S to K_0. It should be noted that `MCS` does

[9] Although we use the Louvain method here, `MCS` is not dependent on the choice of community detection algorithm.

[10] The true communities are considered as the ones found by running the community detection algorithm on the full network.



(a) Regret for MCS on TwitterKP (red) and TwitterOW (blue).

(b) Running time of MCS against the budget on DBLP network.

(c) Sensitivity of MCS to number of layers at different budgets.

Figure 2: Results for regret analysis (2a), running time analysis (2b), sensitivity to number of layers (2c) of MCS.

not have any knowledge about K_0.

We perform experiments on the TwitterKP and TwitterOW networks, and provide the results in Figure 2a. We observe, the regret of MCS grows rapidly at first, but very slowly after around $10\% - 20\%$ of the nodes are queried. This shows that performance of MCS become close to the oracle very quickly.

5.3 Running Time

In this section, we experimentally investigate the running time of MCS with respect to the budget provided. Figure 2b shows the running time vs budget for MCS (Red) on the DBLP network. The running time for random walk (green) is also added as a reference. We can observe that the running time of MCS is super-linear but sub-quadratic, which is consistent with the results of the theoretical analysis in Section 4.7. Results on the other datasets are similar and are omitted.

5.4 Effect of Number of Layers

In this section, we investigate the effect of the number of layers on the performance of MCS. We create multiple synthetic cheaper layers by taking the layer of interest, and keeping 25% of the edges (randomly) and adding 5% noise.

We run experiments with different budgets and number of layers, and examine the effect on the performance of MCS. Figure 2c shows the results of these experiments on the DBLP dataset. We can observe that for all the budgets, there is a peak at around four layers, and the performance drops on either side. This is consistent with the theoretical results in Section 4.8. The results for the sensitivity to query cost are also consistent with the theoretical results and is omitted.

5.5 Effect of Role and Community Bandits

One can ask the question of how much the different bandits contribute to the performance of MCS. To answer this, we compare MCS, which has all the bandits, against ones without CBandit and RBandit. We observed that the algorithms with only two bandits perform well initially. But MCS with all three bandits catches up, and outperforms the others[11].

5.6 Real Twitter API Case Study

In this section, we run MCS on the real Twitter API and evaluate its scalability. As a baseline we also ran random walk. We assume that the Twitter Friends/Followers layer is the one of interest, and mentions, replies are the cheaper ones. Although these API return different number of nodes, for simplicity we assume that they return the same amount, and assign layers cost as 1 for followers/friends, and 0.01 for mentions and replies. We run MCS and random walk on the real Twitter API for 168 hours.

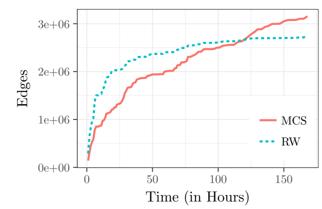


Figure 3: Number of edges found vs time by MCS (red) and random walk (green) using the real Twitter API.

Since we do not have access to the ground truth community of the entire Twitter network, we cannot evaluate the community similarity. However studies have indicated that algorithms that maximizes the number of edges produces samples with community structures that are similar to the full network (Maiya and Berger-Wolf 2010). Figure 3 shows the number of edges found against time for MCS and random walk on the Twitter API. We observe that MCS scales very well with network size, and overtakes the random walk.

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 in each layer, and there are some uncertainty associated with the query responses.

We proposed MCS, a novel multi-armed bandit algorithm that determines which node in the layer of interest to query by using the information from other, cheaper layers.

We tested our algorithm on multiple real world multiplex networks under two settings – *RQR* and *UQR*. We found that MCS outperforms all the baseline algorithms consistently, by up to 49% over the next best method.

[11] The results are not included here due to space limitations. They are included in the supplemental materials.

Acknowledgments

Laishram and Soundarajan are supported by the U. S. Army Research Office under grant number #W911NF1810047. Wendt's work is supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This research was supported in part through computational resources provided by Syracuse University.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Avrachenkov, K.; Basu, P.; Neglia, G.; Ribeiro, B.; and Towsley, D. 2014. Pay few, influence most: Online myopic network covering. In *IEEE INFOCOM International Workshop on Network Science for Communication Networks (NetSciCom 2014)*, 813–818. IEEE.

Barigozzi, M.; Fagiolo, G.; and Mangioni, G. 2011. Identifying the community structure of the international-trade multi-network. *Physica A: statistical mechanics and its applications* 390(11):2051–2066.

Berlingerio, M.; Pinelli, F.; and Calabrese, F. 2013. Abacus: frequent pattern mining-based community discovery in multidimensional networks. *Data Mining and Knowledge Discovery* 27(3):294–320.

Besbes, O.; Gur, Y.; and Zeevi, A. 2014. Stochastic multi-armed-bandit problem with non-stationary rewards. In *Advances in neural information processing systems*, 199–207.

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, É. 2011. The louvain method for community detection in large networks. *J of Statistical Mechanics: Theory and Experiment* 10:P10008.

De Domenico, M.; Solé-Ribalta, A.; Gómez, S.; and Arenas, A. 2014. Navigability of interconnected networks under random failures. *Proceedings of the National Academy of Sciences* 111(23):8351–8356.

Fortunato, S. 2010. Community detection in graphs. *Physics reports* 486(3):75–174.

Girvan, M., and Newman, M. E. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99(12):7821–7826.

Gusfield, D. 2002. Partition-distance: A problem and class of perfect graphs arising in clustering. *Information Processing Letters* 82(3):159–164.

Henderson, K.; Gallagher, B.; Eliassi-Rad, T.; Tong, H.; Basu, S.; Akoglu, L.; Koutra, D.; Faloutsos, C.; and Li, L. 2012. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 1231–1239. ACM.

Katehakis, M. N., and Veinott Jr, A. F. 1987. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research* 12(2):262–268.

Kivelä, M.; Arenas, A.; Barthelemy, M.; Gleeson, J. P.; Moreno, Y.; and Porter, M. A. 2014. Multilayer networks. *Journal of complex networks* 2(3):203–271.

Laishram, R.; Areekijseree, K.; and Soundarajan, S. 2017. Predicted max degree sampling: Sampling in directed networks to maximize node coverage through crawling. In *IEEE INFOCOM International Workshop on Network Science for Communication Networks (NetSciCom 2017)*, number 2017, 940–945.

Lee, K.-M.; Min, B.; and Goh, K.-I. 2015. Towards real-world complexity: an introduction to multiplex networks. *arXiv preprint arXiv:1502.03909*.

Maiya, A. S., and Berger-Wolf, T. Y. 2010. Sampling community structure. In *Proceedings of the 19th international conference on World wide web*, 701–710. ACM.

Mucha, P. J.; Richardson, T.; Macon, K.; Porter, M. A.; and Onnela, J.-P. 2010. Community structure in time-dependent, multiscale, and multiplex networks. *science* 328(5980):876–878.

Rosvall, M., and Bergstrom, C. T. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences* 105(4):1118–1123.

Tang, L.; Wang, X.; and Liu, H. 2009. Uncovering groups via heterogeneous interaction analysis. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, 503–512. IEEE.

Tokic, M., and Palm, G. 2011. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual Conference on Artificial Intelligence*, 335–346. Springer.

Vieira, V. d. F.; Xavier, C. R.; Ebecken, N. F.; and Evsukoff, A. G. 2014. Modularity based hierarchical community detection in networks. In *International Conference on Computational Science and Its Applications*, 146–160. Springer.

Wendt, J. D.; Wells, R.; Field, R. V.; and Soundarajan, S. 2016. On data collection, graph construction, and sampling in twitter. In *Advances in Social Networks Analysis and Mining, 2016 IEEE/ACM International Conference on*, 985–992. IEEE.