Sampling Community Structure in Dynamic Social Networks

Humphrey Mensah and Sucheta Soundarajan

Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY U.S.A. {hamensah,susounda}@syr.edu

Abstract. When studying dynamic networks, it is often of interest to understand how the community structure of the network changes. However, before studying the community structure of dynamic social networks, one must first collect appropriate network data. In this paper we present a network sampling technique to crawl the community structure of dynamic networks when there is a limitation on the number of nodes that can be queried. The process begins by obtaining a sample for the first time step. In subsequent time steps, the crawling process is guided by community structure discoveries made in the past. Experiments conducted on the proposed approach and certain baseline techniques reveal the proposed approach has at least 35% performance increase in cases when the total query budget is fixed over the entire period and at least 8% increase in cases when the query budget is fixed per time step.

1 Introduction

Researchers are interested in a wide variety of problems related to communities in dynamic social networks, including understanding their growth, dissolution, and merging behaviors [17, 16, 10]. However, before studying such questions, a researcher must first obtain an appropriate dataset. Because typical social networks may contain millions or billions of nodes, it can be a challenge to collect adequate data within a reasonable amount of time, due to both the computational efforts required to collect such data as well as API rate limits imposed by the companies owning the data. For example, when crawling the Twitter friendship or follower network, the Twitter API allows only 15 queries per 15 minutes [1]. Given such a scenario, a data collector must make the most of a limited query budget: which areas of the graph should be explored in order to obtain information that is most useful for the analysis task at hand? This is a challenge even in static networks; and the challenge is compounded in dynamic networks, where individual nodes or edges may appear or disappear, and the structure of entire regions of the graph may change in a moment.

In this paper, we focus on the problem of crawling a dynamic social network with the goal of obtaining a sample with community structure that is as representative as possible of the true community structure. Here, a community in a network refers to a group of nodes that are densely connected to each other. In online social networks, a community can represent a group of likeminded users. Identifying such users could be used for marketing and recommendations [3].

Identifying the dynamic community structure in online social networks provides insights into questions such as: which group is migrating to what group, how long does it take for a particular group to collapse, when was a particular group formed, etc.

This paper presents Dynamic Sampler (DYNSAMP) which samples the dynamic community structure of online social networks over a period of time when there are resource constraints. Two resource constraint cases are considered: (1) The case where there is a limitation on the number of times one can request information over the entire period considered (e.g., one has a total amount of money to spend on data collection across the timeline), and (2) The case when there is a limitation at each time step on the number of times one can request information about a node (e.g., there is a daily limit on the number of queries that can be made). DYNSAMP works on the notion that the current community structure of a graph might be partially or wholly similar to previously discovered community structures. Experiments show that DYNSAMP has a performance improvement ranging from 35% to 53% when compared to baseline methods when the query limitation is considered over the entire period and 8% - 56% in cases when there is a limitation at each time step.

The rest of the paper is organized as follows. First, we discuss some related work. In section 3, we present the problem of sampling in dynamic networks. In section 4, we discuss the proposed approach. Section 5 presents the experiments performed and its set up. Finally, section 6 presents the conclusion to the paper and some future directions.

2 Related Work

There has been little work focused on sampling community structure in networks, and most existing work has focused on static networks.

Maiya and Berger-Wolf [12] proposed an expander graph based sampling approach for static networks. This method begins with a seed node and increasingly grows the sample by selecting a node from the neighborhood of the current sample that maximizes a quality function. Also, in the selection of the next node, there is an assumption that the neighborhood of all nodes are known which is not generalizable to most online social networks.

In [5], the authors proposed a link tracing approach for sampling the community structure of static networks. It begins with a seed node and grows the sample by selecting the node with the highest reference score, defined as the ratio of the number of already discovered connections pointing to a node so far in the crawling process to the degree of the node.

A PageRank-based sampling approach (PRS) proposed by Salehi et al. [14] obtains samples from a static network with high community structures. From the simulation results, authors argue PRS has significantly higher performance. However, PRS assumes it knows the number of communities in the network which is not realistic with online social networks.

Another link tracing approach (QCA) proposed for dynamic networks is described in [13]. This begins with an initial community structure. It computes

each of the existing communities' "force" of accepting the node. The community membership is selected based on the "force". QCA is able to compute community membership of discovered nodes. Even though QCA is one of the few techniques proposed for dynamic networks, it assumes it has an initial community structure which is not practical in most online social networks.

In [11], Lu et al., proposed two incremental sampling algorithms for dynamic graphs that preserves some property of interest. Even though it was demonstrated to be performing well, this approach (1) Makes a similar assumption to [12] by assuming it knows the entire graph and (2) does not sample for communities in the network.

In this paper, we propose a crawling based approach to sample the community structure of dynamic networks with a constraint on the number of times one can request information about a node without any knowledge of the community structure.

3 Preliminaries

3.1 Notations

- $G_t = (V_t, E_t)$ is a true, unobserved graph at time step t, where V_t and $E_t \subset V_t \times V_t$ are the set of nodes and edges, respectively, at time step t.
- $G_t^s = (V_t^s, E_t^s)$ is a sampled graph at time step t, where V_t^s and $E_t^s \subset V_t^s \times V_t^s$ are the sampled set of nodes and edges respectively at time step t.
- $G = \{G_1, G_2, ..., G_n\}$ is the true graph sequence and $G^s = \{G_1^s, G_2^s, ..., G_n^s\}$ represents a sampled graph sequence, where $G_i^s \subset G_i$.
- ω_t represents the community structure similarity metric between G_t and G_t^s at time step t.
- q^t represents the number of queries used at time step t to obtain G_t^s.
- q_v represents a vector of the number of queries made to obtain G^s. The i^{th} vector entry is the number of queries made on time step i.
- q represents the total number of queries made to obtain G^s.
- q_{max}^t represents the maximum number of queries allowed at time step t.
- q_{max} is a the total number of queries allowed over the entire timeline.
- The dynamic community similarity $\aleph$ of a sampled graph G^s and a ground truth graph G is defined as:

$$\aleph(G, G^s) = \frac{1}{n} \sum_{t=1}^{n} \omega_t$$

- τ is a dissimilarity threshold for which we declare two communities to be different.

3.2 Problem Formulation

In this work, we assume the true graph sequence G is not known. We also assume that we can determine whether a node is present in a given timestep at no cost, as in many online social networks. Example, the Twitter API allows up to 900 queries per 15 minutes when searching for a user. In each step, a node

can be queried, and all of its neighbors learned. Assuming the process begins with a query on v_1, the next query can only be made on discovered neighbors of previously queried nodes (either from the current or previous time steps). For dynamic networks, we assume there is a storage limitation on how many graphs can be stored for a period considered. Our goal is to generate a sampled graph sequence G^s such that $\aleph(G, G^s)$ is maximized.

We consider two different problem settings: (1) The query budget limits the total number of queries that can be made over the entire timeline (e.g., queries cost money, and we have a fixed amount of money for the entire sampling process). (2) There is a query limit for each timestep (e.g., queries take time, and each time step has a limited amount of time).

4 Proposed Approach

This work proposes a novel algorithm (DYNSAMP) for sampling a dynamic network such that the community similarity between the true and sampled networks is maximized. The intuition behind DYNSAMP is that the current snapshot of a graph may be similar to an earlier snapshot; or if not, portions may be similar.

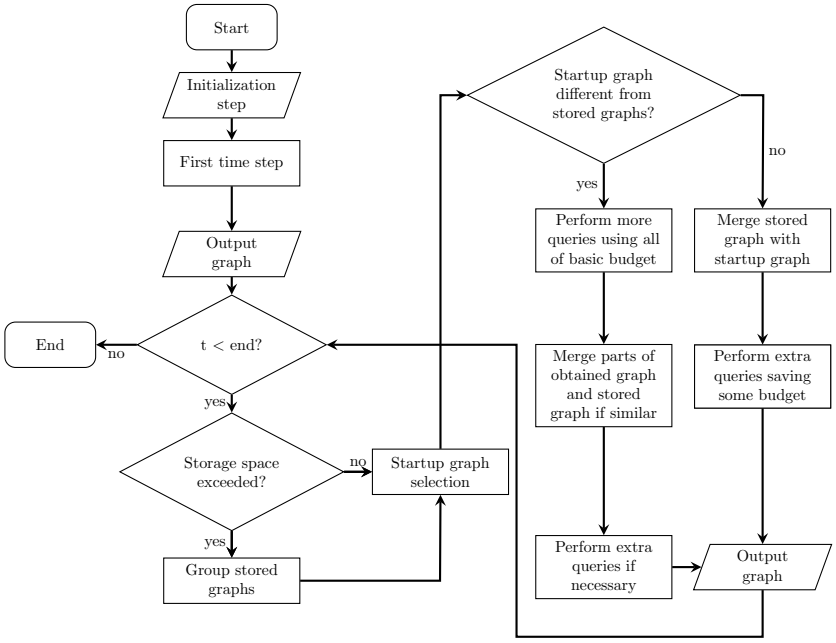


Fig. 1: A high level description of the steps involved in the proposed dynamic network sampling.

DYNSAMP begins by obtaining a sample for the first time step of the sampling process with an allocated number of queries. For subsequent time steps, a fraction of the budget allocated for that time is used to obtain a graph called the *startup graph*. The startup graph is then compared to previously discovered

graphs to determine if they are similar. If similar, a portion of the budget allocated for that time step is saved for future use. If not similar, the entire allocated budget for the time step is used. If there is saved budget, it is used to perform extra queries to grow the graph. Figure 1 shows a high-level view of the proposed approach to sampling dynamic social networks. A detailed description of the steps involved is described below.

4.1 Initialization

The sampling process requires as input either a total budget q or vector of daily budgets q_v, depending on the problem setting, the number of time steps n considered, and a dissimilarity threshold τ above which we declare two communities to be different. If the budget constraint applies to the entire period, for each time step t, we allocate a basic budget $\varphi_t = \varrho_t/n_t$ where ϱ_t and n_t is the budget and number of time steps respectively left as at time step t. However, if there is a limitation for each time step, a basic budget of $\varphi_t = q^t_{max}$ is defined for each time step t.

4.2 First time step

For the first time step of sampling, a *budget* φ_0 is used to generate a sample by beginning with a random node, and in each step, with probability p, querying the node with the maximum observed degree, or with probability $1-p$, jumping to a random node, and storing the observed graph. Our experimental results suggested that this technique works well in comparison to methods such as random node selection and random walk.

4.3 Startup graph selection

In subsequent time steps after the first time step, a fraction of the basic budget η_t is used to obtain a startup graph for the time step under consideration. In this work, a budget of $0.50*\varphi_t$ is used to obtain the startup graph. The nodes queried are noted and the amount of change in their neighborhood of all previously stored graphs over the period is computed using Jaccard similarity.

In generating the startup graph, DYNSAMP selects the top η_t queried nodes with the highest change in neighborhood as at the time under consideration. This selection process ensures that nodes whose neighborhood have not changed over a period are not selected often. The selected nodes are queried to obtain the startup graph. In cases where the number of queried nodes present in the current time step is less than η_t, a random number of nodes are selected from the current nodes to add up to η_t.

4.4 Comparing startup graph to previous graphs

Next, the startup graph is compared to all previously obtained graphs to identify any similarities. In this work, the dissimilarity between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is defined as $1 - |E_1 \cap E_2|/|E_1 \cup E_2|$.

In comparing a startup graph and a previously discovered graph, if all nodes queried in the startup graph are present in the stored graph, only such nodes and

Algorithm 1: DYNSAMP for sampling the community structure of dynamic networks when queries can be saved

```
function DYNSAMP (G, budget, τ, n);
  Input  : G, budget, τ, n
  Output: G^s
snaps = n, φ_1 = budget/snaps, G^s_0 ← getFirstTimeSample(G_0, φ_1) ;
Decrement budget, Decrement snaps;
Store graph at t = 0;
for t = 1 to n do
    if storage exceeded then
        groupGraphs()
    φ_t = budget/snaps,startup = 0.50 * φ_t;
    G^s_t ← getStartUp(G_t, startup);
    G_sel ←Find closest stored graph;
    δ ← graphDissim(G^s_t, G_sel);
    if δ > τ then
        use all of base size;
        G^s_t ← mergeComPart(G^s_t, G_sel) ;
        if δ > τ then
            extra = savedqueries;
            G^s_i ← performExtraQ(G_t, G^s_t, extra);
    else
        extra = (0.90 * φ_t) − startup;
        G^s_t ← merge(G^s_t, G_sel);
        performExtraQ(G_t, G^s_t, extra);
        update saved queries;
    update queried neighbors;
    Decrement budget, Decrement snaps;
```

their neighbors are considered. However, if all nodes queried in the startup graph are not present, a random set of queried nodes in the stored graph are selected and their neighbors are considered for comparison between the two graphs. The selection is done such that the number of nodes queried in both graphs are equal. DYNSAMP selects the stored graph most similar to the startup graph.

4.5 Performing Extra queries

If the startup graph is within τ of the closest graph, the connections in the stored graph that are between nodes present in the current sample are added to the initial graph obtained. In this work, an assumption is made that a check can be made to determine if a node is present or not. The remaining budget is used to perform some extra queries to grow the graph. If budget is allocated for the entire duration, a fraction is saved for future use. By performing extra queries in cases when the graphs are similar, it provides a means of growing the graph that was previously stored.

In cases where the startup graph is identified to be entirely different from all previously obtained graphs, the remaining budget is used to grow the network. A

further check is made if some parts of the startup graph are similar to the closest stored graph. Each community is merged with its closest community in the stored graph based on the defined threshold. Communities in this work were obtained using the Louvain method [6]. In the cases where a budget could be saved, the saved budget is used to perform additional queries, since the discovered community structure is deemed wholly new.

4.6 Handling storage limitations

Due to space limitations, it may not be possible to store all previous graphs especially when sampling for a larger number of time steps. To address this, for all time steps after the first time step, DYNSAMP checks if the entire storage is used before the sampling for that particular time begins. The stored graphs are clustered into groups.

When the storage limit is exceeded, a check is made to determine the number of unique graphs among all stored graphs. A stored graph is said to be unique when it is not similar to any of the stored graphs. If among all the stored graphs there is only one unique stored graph, this means that the graphs stored are all similar to each other and hence grouped into a single graph. As an example, assuming $G_1^s = (V_1^s, E_1^s), G_2^s = (V_2^s, E_2^s), ..., G_m^s = (V_m^s, E_m^s)$ are the currently stored graphs with a single unique stored graph. A new graph $G_\alpha^s = (V_\alpha^s, E_\alpha^s)$ such that $V_\alpha^s = \bigcup_{i=1}^{m} V_i$ and $E_\alpha^s = \bigcup_{i=1}^{m} E_i$ is obtained after the merging process. If there are k unique stored graphs, where $k > 1$, an initial attempt is made to group the graph into k groups. After the grouping into k, if storage is still exceeded, graphs with the least assessed time are repeatedly considered for eviction until the storage criteria is met. Algorithm 1 provides a step by step description of the proposed technique when queries can be saved while algorithm 2 gives the description of the technique when queries can not be saved.

5 Experiments

This section begins with a description of various real and synthetic datasets used for the experiment. It is followed with how the experiments were set up and the main objectives in the various experiments. The section ends with a discussion of the results of each dataset.

5.1 Datasets

We consider five datasets. These include three real world datasets: Autonomous Systems (AS-733) [9], Reality Mining (MIT contact)[8] and Enron email (Enron) [15]. We also include two synthetic datasets (Syn1 and Syn2), generated using Dancer [4]. Dancer generates evolving graphs with embedded community structure.

AS-733 is a communication network constructed from Border Gateway Protocol logs. It contains 733 daily instances, the largest of which has 6474 nodes and 12572 edges. Reality Mining is a human contact network among 100 MIT students. This dataset contains 229 daily instances describing contacts between

Algorithm 2: DYNSAMP for sampling the community structure of dynamic networks when queries can not be saved

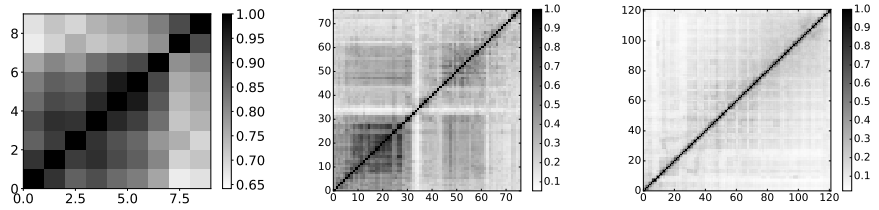
```
function DYNSAMP (G, q, τ, n);
  Input  : G, q, τ, n
  Output: G^s
G_0^s ← getFirstTimeSample(G_0, q_max^0) ;
Store graph at t = 0;
for t = 1 to n do
    if storage exceeded then
        groupGraphs()
    startup = 0.50 * q_max^t;
    G_t^s ← getStartUp(G_t, startup);
    G_sel ←Find closest stored graph;
    δ ← graphDissim(G_t^s, G_sel);
    extra = q_max^t − startup;
    if δ > τ then
        performExtraQ(G_t, G_t^s, extra);
        G_t^s ← mergeComPart(G_t^s, G_sel) ;
    else
        G_t^s ← merge(G_t^s, G_sel);
        performExtraQ(G_t, G_t^s, extra);
    update queried neighbors;
    updated stored graphs;
```

users, each of which has up to 76 nodes and 418 edges. We aggregate these daily instances using window sizes of 10 days, with a step of 3, to generate a total of 77 snapshots. Enron is an email network. We use a dataset containing daily snapshots during the year 2001, again aggregated as above, for a total of 122 snapshots. These graphs contain up to 7225 nodes and 15938 edges. All networks exhibit the addition and deletion of both nodes and edges. Some snapshots are aggregated to ensure all the different community structure behavior are considered in the experiment.

The synthetic networks both have an initial node count of 2000 initially grouped into 20 and 24 communities for Syn1 and Syn2 respectively. These go through different community evolution phases such as splitting and merging. [1]

The largest time step of Syn1 has 2293 nodes with 17813 edges. Syn1 over the period considered shows an addition and deletion of edges. However, it only demonstrates the addition of nodes over the period. In Syn2, the largest number

[1] This model requires a number of parameters. We set $k = 20, nBVertices = 2000, nbTimestamps = 10, prMicro = 0.2, prMerge = 0.4, removeVertices = 0.4, prSplit = 0.4, prChange = 0.4, addBetweenEdges = 0.2, addVertices = 0.1, removeBetweenEdges = 0.4, removeWithinEdges = 0.1, updateAttributes = 0.1$. For Syn2, the same settings were maintained with modification to the following: *prMicro*=0.5, *addBetweenEdges*=0.5, *removeBetweenEdges*=0.9, and k=24.



a) Syn1 b) MIT contact c) Enron

Fig. 2: A plot of the similarity between community structures over time for three different group of networks. 2a) is a network with totally stable community structure, 2b) has a partially stable community structure and 2c) is network with a completely unstable community structure. Black indicates two graphs are equal and white indicates they are completely different. These examples demonstrate stable (Syn1), mixed (MIT), and unstable (Enron) structures.

of nodes over the period is 2859 and the largest number of edges over the period is 21459. Syn2 tends to have communities splitting or migrating more often than communities in Syn1.

5.2 Experimental Setup

In our experiments, we set $\tau = 0.4$ and $p = 0.80$. The budget size in the experiments with a strict budget limitation for each time step is defined to 20% of the number of nodes at each time step. Budgets for the setting in which we have a total number of queries for the entire timeline are stated later in this section.

To the best of our knowledge, there is no sampling method that explicitly focuses on the community structure of dynamic networks without assuming knowledge of the entire network. The proposed method is therefore compared with random walk and breadth-first search baselines.

5.3 Evaluation Metrics

We use two metrics to evaluate DYNSAMP and the baselines above. The first metric is based on a Jaccard-based metric proposed in [18], modified for evaluating dynamic samples. Given a sampled set of communities C_i^s and a true set of communities C_i, this metric finds the closest true community to each sampled community, and vice versa, and averages these similarities. We also use the popular Normalized Mutual Information (NMI) metric, described in [2, 7].

5.4 Results and Discussion

For each of the datasets, we run DYNSAMP and the baselines 10 times to generate a dynamic sample with specific budget. We compare to communities detected on the complete network by the Louvain method [6]. Results for both evaluation metrics were similar, so we present results for NMI only in Figure 3.

In our experiments, we use budgets of 199000, 850, 13000, 2500 and 2500 respectively for AS-733, MIT-contact, Enron, Syn1 and Syn2. Figure 3 shows a

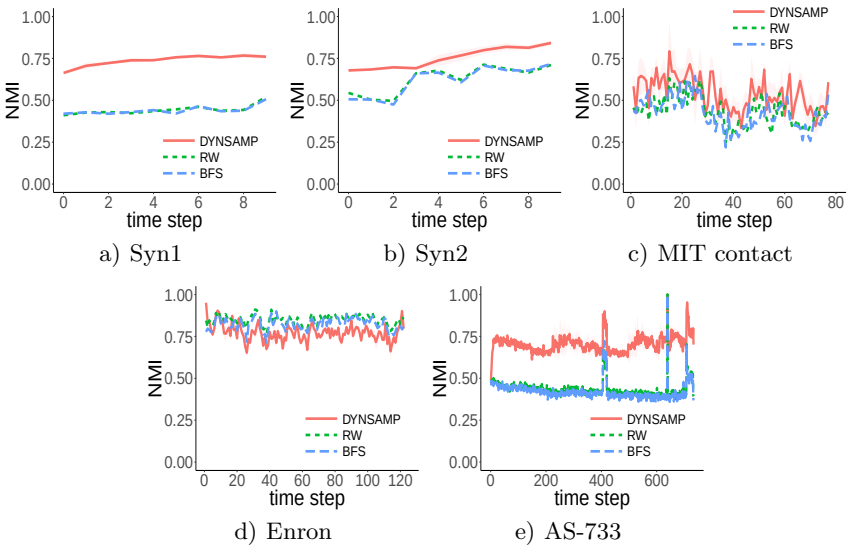


Fig. 3: A plot of the NMI between a sampled graph and its corresponding true graph over time. Shading represents the standard deviation over 10 trials. DYNSAMP outperforms the other methods with respect to NMI in most cases. When graph changes at each time step like 3d), it performs just as baseline methods.

similar plot of the NMI with respect to time (results were similar for the Jaccard-based evaluation metric). In these experiments, the setting where a budget is given over the entire period is used. Similarly, Figure 4 shows a similarity plot of the NMI with respect to time when there is a budget limitation per timestep.

Dynamic social networks can be categorized into three groups based on the stability of the community structure over the period considered (see Figure 2 for examples): those that are stable over the entire period (e.g., Syn1), those that are unstable (e.g., Enron), and those that are mixed (e.g., Reality Mining).

In a dynamic network where there is a complete or partial stability of the community structures over the period considered, DYNSAMP outperforms baseline methods substantially. When the community structure changes significantly at each time step, like the Enron dataset, there is no significant difference between DYNSAMP and the baselines, because it cannot learn from the past.

We next investigated whether the number of graph samples stored had a significant impact on the performance of DYNSAMP. The investigation was divided into two: graphs that have some stability over time (Syn1) and graphs with no stability over time (Enron). We observe that, in general, the performance of DYNSAMP is not dependent on the number of graphs being stored. If there is some stability, it will be merged over time and hence keeping several copies of them will neither improve or worsen the performance. In cases where there is no stability, the number of stored graphs has no impact on the learning process.

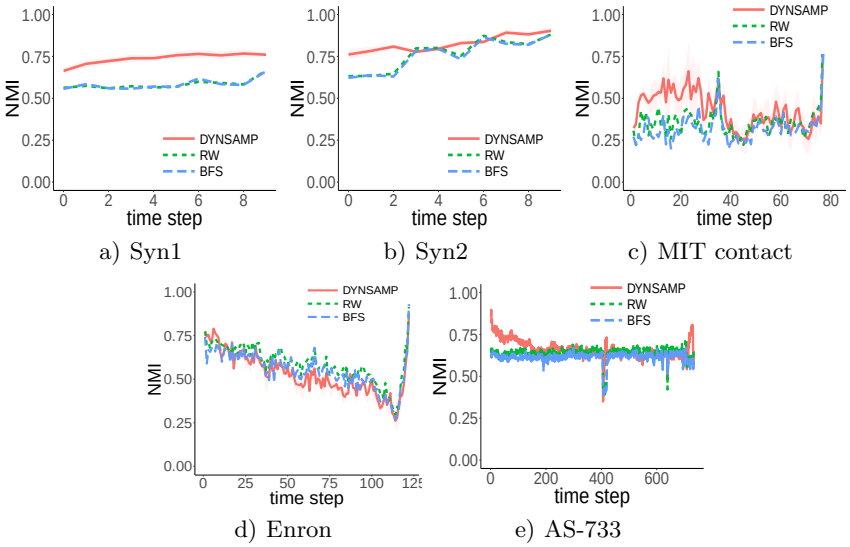


Fig. 4: NMI between a sampled and true graphs, with a sample budget for each time step. Shading represents the standard deviation over 10 trials. DYNSAMP outperforms the other methods with respect to NMI in most cases.

Overall, we observe that DYNSAMP performs better than baseline methods in most cases. With the Jaccard based measure, it outperforms RW by 42% and BFS by 46% on average, and by 35% and 53% as measured by NMI.

6 Conclusion

Sampling provides a means of selecting some parts of the graph such that certain features of the original graph are preserved. In this paper, we addressed the problem of sampling a dynamic social network when there is a limitation on the number of nodes that could be asked for information. We performed experiments on several real world and synthetic networks. We showed that in most cases the proposed approach outperforms baseline methods. However, in cases where the community structure for each time step changes significantly, the algorithm performs as well as the baseline methods.

References

1. Twitter developer documentation. `https://dev.twitter.com/rest/reference/get/followers/ids`. Accessed: 08-16-2017.
2. Hamidreza Alvari, Alireza Hajibagheri, Gita Sukthankar, and Kiran Lakkaraju. Identifying community structures in dynamic networks. *Social Network Analysis and Mining*, 6(1):77, 2016.
3. Punam Bedi and Chhavi Sharma. Community detection in social networks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(3):115–135, 2016.

4. Oualid Benyahia, Christine Largeron, Baptiste Jeudy, and Osmar R. Zaïane. Dancer: Dynamic attributed network with community structure generator. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 41–44. Springer, 2016.
5. Norbert Blenn, Christian Doerr, Bas Van Kester, and Piet Van Mieghem. Crawling and detecting community structure in online social networks using local information. *NETWORKING 2012*, pages 56–67, 2012.
6. Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.
7. Yuzhong Chen and Xiaohui Qiu. Detecting community structures in social networks with particle swarm optimization. In *Frontiers in Internet Technologies*, pages 266–275. Springer, 2013.
8. Nathan Eagle and Alex Pentland. Reality mining: sensing complex social systems. *Personal and ubiquitous computing*, 10(4):255–268, 2006.
9. Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.
10. Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L. Tseng. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 685–694. ACM, 2008.
11. Xuesong Lu, Tuan Quang Phan, and Stéphane Bressan. Incremental algorithms for sampling dynamic graphs. In Hendrik Decker, Lenka Lhotská, Sebastian Link, Josef Basl, and A. Min Tjoa, editors, *Database and Expert Systems Applications*, pages 327–341, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
12. Arun S. Maiya and Tanya Y. Berger-Wolf. Sampling community structure. In *Proceedings of the 19th international conference on World Wide Web*, pages 701–710. ACM, 2010.
13. Nam P. Nguyen, Thang N. Dinh, Ying Xuan, and My T. Thai. Adaptive algorithms for detecting community structure in dynamic social networks. In *Proceedings of the 2011 IEEE international conference on Computer Communications*, pages 2282–2290. IEEE, 2011.
14. Mostafa Salehi, Hamid R. Rabiee, and Arezo Rajabi. Sampling from complex networks with high community structures. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(2):023126, 2012.
15. Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696. ACM, 2007.
16. Gautam S. Thakur, R. Tiwari, M.T. Thai, S.-S. Chen, and AWM Dress. Detection of local community structures in complex dynamic networks with random walks. *IET systems biology*, 3(4):266–278, 2009.
17. Chang-Dong Wang, Jian-Huang Lai, and Philip S. Yu. Neiwalk: Community discovery in dynamic content-based networks. *IEEE transactions on knowledge and data engineering*, 26(7):1734–1748, 2014.
18. Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *Data Mining (ICDM), 2013 IEEE 13th international conference on*, pages 1151–1156. IEEE, 2013.