Clustering, Prominence and Social Network Analysis on Incomplete Networks

Kshiteesh Hegde, Malik Magdon-Ismail, Boleslaw Szymanski and Konstantin Kuzmin

Abstract Social networks are a source of large scale graphs. We study how social network algorithms behave on sparsified versions of such networks with two motivations in mind:

1. In practice, it is challenging to collect, store and process the entire often constantly growing network, so it is important to understand how algorithms behave on incomplete views of a network.
2. Even if one has the full network, algorithms may be infeasible at such large scale, and the only option may be to sparsify the networks to make them computationally tractable while still maintaining the fidelity of the social network algorithms.

We present a variety of methods for sparsifying a network based on linear regression and linear algebraic sampling for graph reconstruction. We *compare the methods against one another* with respect to clustering. Specifically, given a graph G, we sample the columns of its adjacency matrix and reconstruct the remaining columns using only those sampled columns to obtain $\hat{G}$, the reconstructed approximation of G. We then perform clustering on G and $\hat{G}$ to get two sets of clusters and compute their modularity, fitness and centrality. Our thorough experimentation reveals that graphs reconstructed through our methodology preserve (in some cases, even improve) community structure while being orders of magnitude more efficient both in storage and computation. We show similar results if the target is prominence of nodes rather than clusters.

1 Introduction

The ever increasing popularity of social networks has resulted in increasing availability of massive graphs. Their sheer size renders them unwieldy for carrying out

Kshiteesh Hegde e-mail: hegdek2@rpi.edu · Malik Magdon-Ismail e-mail: magdon@rpi.edu · Boleslaw Szymanski e-mail: szymab@rpi.edu · Konstantin Kuzmin e-mail: kuzmik@rpi.edu, Dept. of Computer Science, Rensselaer Polytechnic Institute, Troy, NY

downstream machine learning operations. Further, such networks are difficult to measure entirely and often we can only access partial snapshots. We need ways to extract information from partially observed networks. This is one of the key motivations for our work, which is to address the question: Are there ways of sampling the edges of the network (perhaps re-weighting them) so that machine learning on the sparsified (incomplete) network produces results that are faithful to the full network.

The task of sampling a graph has applications across many domains. For example, in a social network with a billion nodes, questions arise like: who are a person's potential friends or who are the leaders and influencers of a given group of people? In a very large research collaboration network, we may want to know which researchers are leaders in a particular field or who are the best collaborators between different fields. In a product rating setting, sellers may want to know which products (movies, books) in one genre are a gateway to another genre. Getting a bird's eye view of these large networks (and many other types of networks) can be instrumental in solving interesting problems quickly. In this paper, we propose ways to address these problems using techniques from graph sparsification and reconstruction.

As discussed later in the related work section, there is a body of research accumulating in the Linear Algebra community which tries to approach this problem by treating the networks as matrices. The benefit is that the spectral structure of matrices can be preserved up to a finite rank if the samples are chosen carefully. In this work, we use these techniques in the social network analysis (SNA) setting. We also use linear regression in one of our methods where we choose a subset of the columns of the adjacency matrix of the full dataset and regress on the remaining columns to get our estimate.

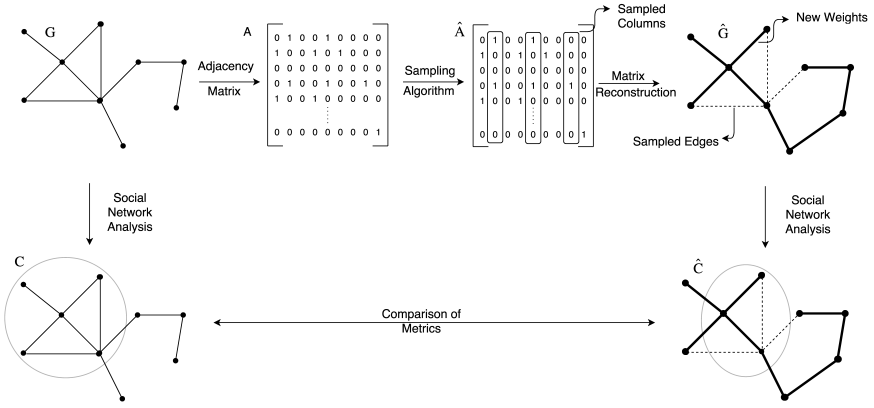


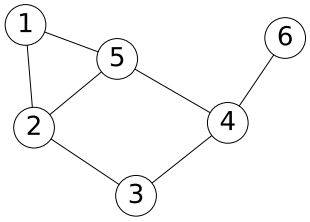
Fig. 1: Outline of our workflow. The dataset is represented as an adjacency matrix. The columns of this matrix are sampled to yield a new adjacency matrix which has new weights for its edges. A new graph is constructed by reconstructing the missing edges using this adjacency matrix. The clustering metrics are computed on both the full and sampled graphs.

The essence of our work is visualized in Fig. 1. The dataset could be in multiple formats but we represent it as an adjacency matrix A with $A(i,j) = A(j,i) = w$ if there is an edge e between i^{th} and j^{th} node with weight $w > 0$. Let r_i be the i^{th} row of A. We choose a small subset of the columns of A. This corresponds to choosing certain nodes from the graph and all the edges that those nodes are involved in. Then using the symmetric property of the adjacency matrix, linear regression and linear algebraic sampling methods (see section 3) we reconstruct the missing edges and nodes. The weights of the edges in the new graph will change depending on the probabilities with which the rows of A are chosen. We call the adjacency matrix of our reconstructed graph (which corresponds to the modified dataset) $\hat{A}$. Now, to evaluate the performance of our method, we compute clustering metrics on this new dataset. We compare them with those obtained from the full dataset.

We formulate two problems in this study:

1. Given A, sparsify to $\hat{A}$, so that machine learning tasks on $\hat{A}$ are faster and produce almost as accurate predictions as from A.
2. Knowing nothing about A, identify a few columns to sample to get $\hat{A}$, so that machine learning tasks on $\hat{A}$ are more efficient and produce almost as accurate predictions as from A.

To give a better idea of this process, we illustrate it by using a toy graph accompanied by the corresponding adjacency matrix A in Fig. 2. Each of the edges in the graph is assumed to have a unit weight unless it is shown thicker in which case it would mean that it has a weight $w > 1$.



	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0

Fig. 2: A toy graph and its associated adjacency matrix

Let the columns $1,2,3$ and 6 be sampled from A. The entries in the unseen columns 4 and 5 are partially populated by using the symmetric nature of A. We apply our reconstruction techniques on these sampled columns and the partially populated columns to get estimates of A. In one method (Algorithm 1), we use linear regression to guess the unseen edges and in the other method (Algorithm 2) we rescale the weight of the seen edges by $\sqrt{1/p_i}$ where p_i is the probability of i^{th} being chosen to "make up" for the lost edges. For example, if columns from A are chosen uniformly, then $p_i = 1/6$. Let the estimates obtained from these two very different approaches be $\hat{A}_1$ and $\hat{A}_2$ respectively. We show the corresponding graphs in Fig. 3. Finally, we perform clustering (see Section 3.4) on $\hat{A}_1$ and $\hat{A}_2$ and compute

some metrics (see Section 3.5) to measure the performance of our algorithms. Fig. 3 also shows the clustering on $\hat{A}_1$ and $\hat{A}_2$. Note that all the nodes of the same color belong to the same cluster.

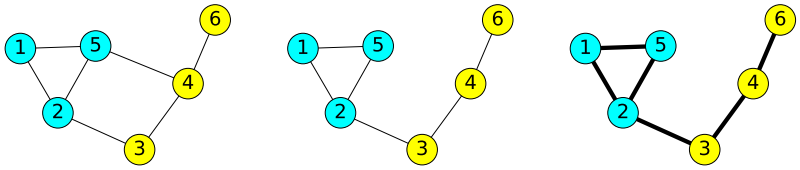


Fig. 3: Toy graph and clustering for its estimates

Our Contribution and Summary of Results

In this work we examine the feasibility of sampling and reconstructing large graphs when we do not have access to the entire graph while proposing two methods to address the problem. We simulate the issue of having incomplete graphs by choosing a subset of the full graph and working only with this small subset to build the unseen graph. Specifically, we choose some well-known metrics pertaining to graphs and use them as a yardstick to measure the performance of the different sampling algorithms which treat graphs as matrices. Our main contribution is to show that it is feasible to extract useful information from incomplete graphs and designing two algorithms to do so.

Some of the key observations that we were able to make are discussed below. We were able to improve the modularity of the clusters even when progressively sampling only 0.15% of the nodes and their related edges. The expansion of the clusters actually improved and was better in the sampled datasets. We were able to achieve this by using only a tiny fraction of the time required for processing the full graph. For example, the Amazon dataset (see section 3.6), which has over $300,000$ nodes and $900,000$ edges, took almost an hour to be evaluated while with just 0.45% of the data, we were able to evaluate it with reasonable accuracy in about 6 minutes. Some metrics were more robust to sparsification than others. Prominence (centrality) measures weren't preserved as well as clustering metrics. We believe the reason for this is that clustering is inherently more robust compared to centrality in the sense that it is less specific. A more detailed analysis can be found in section 4.

2 Related Work

There is some work done related to sampling of graphs. A few researchers [23], in a collaborative effort, compared breadth first search random walk based sampling

methods and their conclusions were not promising. In a relatively older work [20], the authors came up with a scheme where a few "landmark" nodes are selected beforehand and the shortest path distances between two nodes are estimated based on that at runtime. The "landmark" node in a way summarizes a few nodes and thus can be treated as a representative for those nodes. This is not sampling of edges or nodes per se but we are mentioning this work because it tries to make the graph "small" before going ahead with downstream computations. However, another work [21] actually samples the edges and keeps the number of nodes unchanged in order to achieve faster graph clustering. They rank the edges using a similarity heuristic and then retain a set number of edges per node. Another interesting work [22] treats the graph as an electrical network and computes effective resistances of the edges and sparsifies the graph. There has been some prior work [13] which looks at sampling of graphs but they only sample randomly and do not consider graphs as matrices. Another work [14] contains a comparison of community detection algorithms on graphs but does not take into consideration the issues arising from working with large scale graphs.

There is another line of work which looks at computing centrality measures on large graphs quickly and efficiently. For example, in [24] the authors try to use virtual nodes in graphs in an attempt to quickly compute betweenness centrality. They assume the graphs are large, sparse and lightly weighted and inject virtual nodes into them and then compute betweenness centrality. One of the breakthrough works [6] significantly reduces the time required to compute betweenness centrality. Later work [4] proposed further improvements, so we think running those algorithms on sampled graphs would greatly increase the size of datasets on which such computations are feasible; especially when combined with parallel methods like multithreading [15].

Another approach, perhaps very relevant to the kind of sampling algorithms that we study in this paper, is the use of matrix sparsification techniques with a goal of sparsifying them as discussed in [1] and [3]. Finally, [16] covers a lot of randomized algorithms aimed at obtaining an approximation of a matrix.

We do not perform a thorough survey of all the clustering algorithms available as it is beyond the scope of this work. Interested readers can refer to [9] for such an analysis. In our work, we compare clustering metrics, as we will discuss soon, computed on large graphs and their reconstructed counterparts.

3 Methodology

In this section we discuss the algorithms, metrics, datasets and experimental setup used in this paper. We investigate two methods of solving the problem of reconstructing incomplete graphs. We use MATLAB®™ notation in Algorithms 1, 2.

1. **Linear Regression**: Given the square symmetric adjacency matrix $A \in \mathbb{R}^{n \times n}$ of graph G with n nodes, we randomly choose $k < n$ ($k \ll n$ if n is very large) columns of A. Let this be $X \in \mathbb{R}^{n \times k}$. We can use the symmetric property of A to

partially fill out $Y \in \mathbb{R}^{n\times(n-k)}$. The indices of the k columns that were selected are stored. Now, we use linear regression to get an estimate $\hat{Y}$ of Y. We have $\hat{Y} = X(X^{\dagger}Y)$ where $X^{\dagger}$ represents the Moore-Penrose pseudo-inverse of X.
We get the estimate $\hat{A} \in \mathbb{R}^{n\times n}$ of A by using X, $\hat{Y}$ and indices of k sampled columns. The above process is shown in Algorithm 1.

Algorithm 1 Linear Regression

1: $A = \texttt{get\_adjacency\_matrix}(G)$	▷ Graph G is given
2: $K = \texttt{randsample}(N, k)$	▷ Store the indices of k chosen columns
3: $X = A(:, K)$; $Y = \mathbf{0}^{k\times n-k}$	▷ Get the k columns from A
4: $Y(K, :) = X(N-K, :)^T$	▷ Use symmetric properties of A to partially fill Y
5: $\hat{Y} = X(X^{\dagger}Y)$	▷ Perform Linear Regression to build unseen graph
6: $\hat{A} = \mathbf{0}^{n\times n}$	▷ The new adjacency matrix
7: $\hat{A}(:, K) = X$	▷ Sampled columns
8: $\hat{A}(:, N-K) = \hat{Y}$	▷ Reconstructed columns

Note that Algorithm 1 can be applied multiple times to the full adjacency matrix to get multiple reconstructions of the graph. These estimates can then be combined to get a new estimate. In fact, in section 4, we test this approach by taking up to three estimates while evaluating the performance.

2. **Linear Algebraic Sampling Method**: In this approach, we initially choose k columns randomly from A. Instead of working with two matrices X and Y like in 1, we work with only one $X_i \in \mathbb{R}^{n\times n}$ matrix. The way X_1 is built is as follows. The chosen columns are rescaled by a factor of the probability with which they were chosen. This acts as the reconstruction step because in a way we are accounting for the missing information by giving more importance to the entries that we have. In addition, since A is symmetric, we further fill X_1 using this information. Now, we use one of the sampling algorithms which will be described in Section 3.3 to get a set of probabilities to further sample A. Using this set of probabilities, we will have k more columns. We can build X_2 in a similar fashion to X_1. With these two estimates of A, we can now build $\hat{A}$ as follows.

$$\hat{A} = \alpha X_1 + (1-\alpha)X_2 \tag{1}$$

where α can be varied between 0 and 1 to get a weighted average of the estimates. Note that this process can be repeated to get different estimates. This is shown in Algorithm 2.

3. **Sampling Algorithms**: The following sampling methods can be used in step 7 of Algorithm 2.

   a. Leverage Score Sampling (LVG): Given an $m \times n$ matrix A with $m > n$, let U denote an $m \times n$ matrix consisting of the left singular vectors of A. If the row vector $U_{(i)}$ is the i^{th} row of the matrix U, then $l_i = \|U_{(i)}\|_2^2$ for $i \in \{1,\ldots,m\}$ are the leverage scores [17] of the rows of A. The leverage scores signify the "influential" rows that can be "good representatives" of a matrix. We compute

Algorithm 2 Linear Algebraic Sampling (LAS)

1: $A = \texttt{get\_adjacency\_matrix}(G)$ ▷ Graph G is given
2: $K = \texttt{randsample}(N,k)$ ▷ Store the indices of k chosen columns
3: $X_1 = \mathbf{0}^{n\times n}$
4: $X_1(:,K) = A(:,K)$ ▷ Get the k columns from A
5: $X_1(K,:) = A(K,:)^T$ ▷ Use symmetric properties of A
6: $X_1 = \texttt{diag}(P_1) \times X_1$ ▷ P_1 is the vector of rescaling factors of length n
7: $P_2 = \texttt{smpl\_algo}(X_1)$ ▷ Get a new set of probabilities using one of the sampling algorithms
8: $K = \texttt{sample}(P_2, N-K, k)$ ▷ Get new unseen k columns w.r.t. P_2
9: $X_2 = \texttt{construct\_X}(K_2, P_2)$ ▷ Repeat steps $3-6$ on X_2
10: $\hat{A} = \alpha X_1 + (1-\alpha)X_2$ ▷ Reconstructed $\hat{A}$

these scores for the given matrix and use them as probabilities for selecting a particular column from that matrix.

b. Dual-Set Sparsification (DSS): Described in [5], DSS is a deterministic algorithm that selects rows from matrices with orthonormal columns. It is based on [22] that we reviewed in Section 2. We recommend referring to Algorithm 1 in [5] to get more details about this method. In short, it returns a set of n weights out of which r are non-zero, which are the sampling probabilities for our purposes, for an $l \times n$ matrix A of rank k in $O(rnk^2 + nl)$ time.

c. Adaptive Sampling (AS): For a detailed discussion of this method refer to Section 2 in [8]. To summarize, this algorithm does sampling in multiple iterations and in an adaptive manner. The rows in each new iteration get picked with probabilities proportional to their squared distances from the span of the rows that have already been picked previously.

All the algorithms above come with some form of theoretical guarantees for preserving the spectral structure of the Laplacian [17], [5], [8].

4. **SpeakEasy**: This [10] is a label propagation clustering algorithm which robustly detects both overlapping and non-overlapping clusters. The nodes in SpeakEasy update their labels based on their neighbors' labels and take into account their global popularity in the network. Note that we do not aim to improve clustering performances, but use this state-of-the-art "off the shelf" method. It could be an interesting extension to this work to use different clustering algorithms.

5. **Performance Metrics**: To compare the quality of the sparsified graph $\hat{G}$ with the ground truth G we use the clusters and prominence measures obtained from both graphs. Let the community partition be given for a network $G = (V,E)$ with $|E|$ edges. Let C be the set of all communities, c a specific community in C with $|c|$ number of nodes, $|E_c^{in}|$ the number of edges between nodes within community c, $|E_c^{out}|$ the number of edges from the nodes in community c_i to the nodes outside c.

   a. Modularity (Q) [18], [19]: Modularity for unweighted and undirected networks is defined as the ratio of difference between the actual and expected

(in a randomized graph with the same number of nodes and the same degree sequence) number of edges within the community.

$$Q = \sum_{c \in C} \frac{|E_c^{in}|}{|E|} - \left(\frac{2|E_c^{in}| + |E_c^{out}|}{2|E|} \right)^2 \tag{2}$$

b. Contraction [7]: It measures the average number of edges per node inside a community. The larger the value of this metric, the higher the quality of the community. For undirected networks (the ones examined in this work), this would be $\frac{2|E_c^{in}|}{|c|}$
c. Expansion [7]: It measures the average number of edges outside a community. The smaller the value of this metric, the higher the quality of the community. Using the previous notation, expansion would be $\frac{|E_c^{out}|}{|c|}$
d. Conductance [7]: It measures the fraction of the total number of edges that have an endpoint outside a community. A smaller value of conductance means a better community. Conductance is defined as $\frac{|E_c^{out}|}{2|E_c^{in}|+|E_c^{out}|}$
e. Intra-Density [7]: The internal density of a community. The larger the value of this metric, the higher the quality of communities. For a particular community c, intra-density is defined as $\frac{2|E_c^{in}|}{|c|(|c|-1)}$
f. Fitness [7]: The ratio between the internal degree and the total degree of a community. Higher the value of fitness, better the quality of the community. Fitness is defined as $\sum_{c \in C} \frac{|E_c^{in}|}{|E_c^{in}|+2|E_c^{out}|}$

6. **Datasets**: We used a variety of data sets in our experiments ranging from e-commerce to collaboration networks to social networks. We summarize the datasets here.

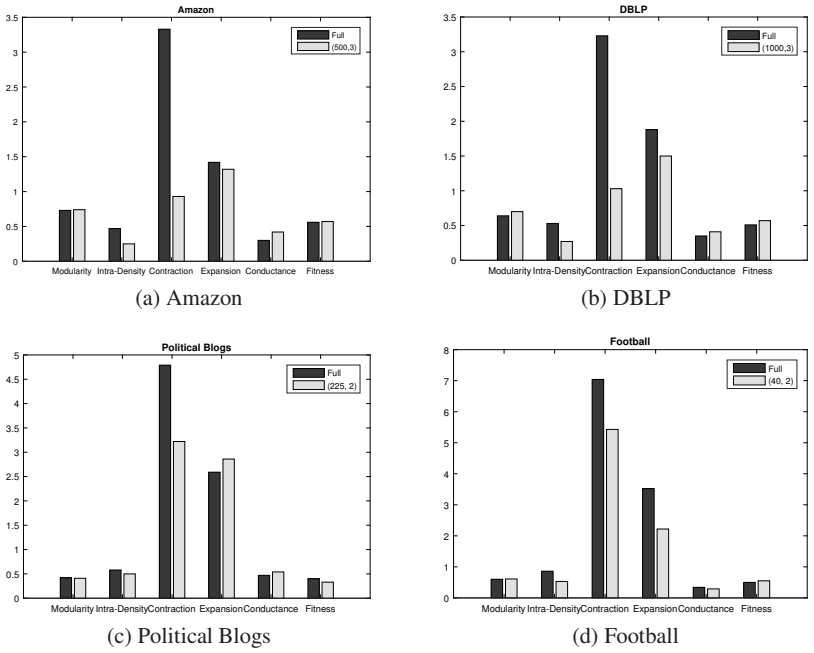
a. Amazon [12]: This is a product co-purchase network of amazon.com. If a product is frequently co-purchased with another product then those two products have an undirected edge between them. There are 334,863 nodes and 925,872 edges.
b. DBLP Collaboration Network [25]: In this co-authorship network, two authors are connected if they have published at least one paper together. It has 317,080 nodes and 1,049,866 edges.
c. Political Blogs [2]: This is a directed network of hyperlinks between weblogs on US Politics during 2004 general election. It has 1,224 nodes and 19,022 edges.
d. College Football [11]: This network represents the schedule of games between college football teams in a single season. There are 115 nodes and 613 edges.
e. Zachary's Karate Club [26]: This network represents the friendships between 34 members of a karate club at a US university during two years. It has 34 nodes and 78 edges.

4 Performance Analysis

In this section, we describe the experimental setup and the choices that were made for the experiments. We sampled between 0.15% and 30% of columns from the datasets. We chose 3 different k's for each dataset: $500, 100$ and $, 2000$ for Amazon and DBLP, $150, 225$ and 300 for Political Blogs, $30, 40$ and 50 for Football and $5, 7$ and 10 for Karate. Also, for each dataset and each k, we ran 3 iterations of Algorithm 1. This way, we had 3 estimates of the dataset for each k. We also timed each process and the comparison between full datasets and their estimates is shown in Fig 5. In case of Algorithm 2, instead of running the same algorithm three times, we ran it only once for each of the sampling methods described before. Thus, we again obtained three estimates. Similar to the earlier process, we timed Algorithm 2 as well and the performance is shown in Fig 4. The parameter mentioned in Equation 1 was set to 0.3 to give importance to the latest reconstruction of the dataset. The dark bars represent the full datasets while the gray bars represent the best performing partial datasets. Y-axes in both Fig. 4 and Fig. 5 represent the value of the metrics.

After we had the estimates from either algorithm, we performed the task of clustering on them. We ran the clustering algorithm mentioned in Section 3.4 on new adjacency matrices to obtain new sets of clustering. Now, with the clustering set of the full graph and that from the estimated graph, we were able to compute the community quality measures defined in Section 3.5.

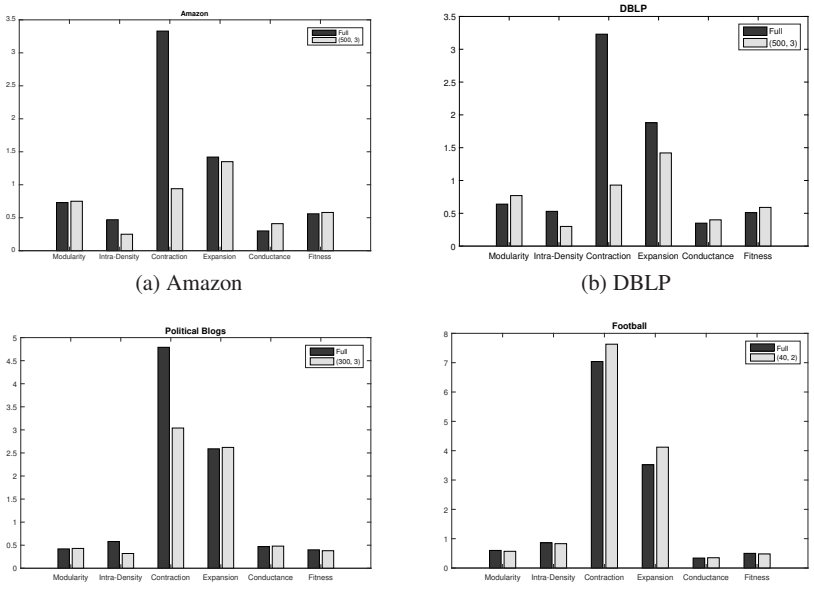
1. **Clustering**: We can see that modularity, intra-density, expansion, conductance and fitness are all very well preserved irrespective of the algorithm or the dataset. We show the results for the best k and best performing sampling algorithm. We would like to note that the metrics are also preserved for other values of k. Readers can refer to the legend in each of the figure to see what k and how many iterations of running the algorithm (in case of Fig. 5) and with which sampling algorithm (in case of Fig. 4) produced the best results. For Fig. 4, we use the notation: 1=LVG, 2=DSS, 3=AS. This, combined with the fact that expansion has improved (lower the better) in almost every case shows that the reconstructed graphs have a better community structure. In case of algorithm 1 (Fig. 5), we learned that running at least 2 iterations provides the best results. We omit the results for the Karate dataset to conserve space.
2. **Runtime**: In Fig. 6 it can be seen very clearly that using the algorithms proposed in this paper one can save a tremendous amount of time while preserving the community structure of the graphs. We show the runtime results only for Algorithm 1 to conserve space. Both algorithms perform very similarly. The difference in runtime is very clear for large graphs like Amazon and DBLP. Processing the full Amazon graph requires about 3500ms while the best performing iteration/sampling algorithm takes less than 500ms. This translates to our algorithm being roughly 7 times faster. Similar results can be observed with DBLP and the small datasets.



(a) Amazon (b) DBLP

(c) Political Blogs (d) Football

Fig. 4: Performance of Linear Algebraic Sampling Methods

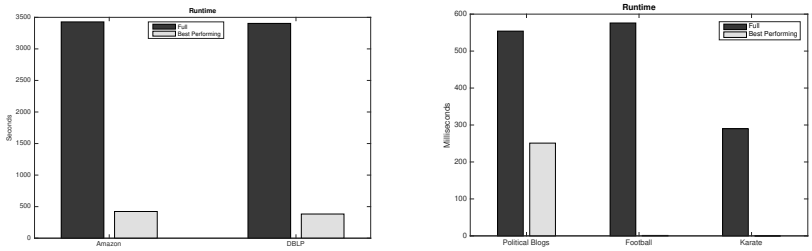


(a) Amazon (b) DBLP

(c) Political Blogs (d) Football

Fig. 5: Performance of Linear Regression

3. **Centrality**: As it was noted in the summary in Section 1, centrality measures like degree, betweenness and closeness were not as well preserved as the community structure. However, they tend to be closer to the full graph as we increased the number of sampled columns k. For example, $k = 10,000$ on Amazon dataset, for top 10% nodes in terms of degree centrality, yielded an F-measure of 0.02 and 0.002 for $k = 500$. In essence, if one is just interested in getting the community structure of a large graph, with minimal information, then the methodology proposed in this paper produces results of sufficient quality. If more specific features of the graph are required then one would have to invest more time and effort to get more information.



(a) Runtime (b) Runtime

Fig. 6: Runtime of Linear Regression

5 Conclusion

The results presented in this paper show that graphs can indeed be sampled like matrices using sampling techniques from the matrix algebra community while preserving clustering features. We present evidence that using only $0.15\% - 30\%$ of the edges of a graph yields communities whose quality is comparable to that of the full graph, according to the most important metrics. We think that going forward, with these results, sampling and reconstruction of large graphs can be considered an important first step before performing machine learning.

Acknowledgements This research was supported by the Army Research Laboratory under Cooperative Agreement W911NF-09-2-0053 (the ARL-NSCTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation here on.

References

1. Achlioptas, D., McSherry, F.: Fast computation of low-rank matrix approximations. JACM (2007)
2. Adamic, L.A., Glance, N.: The political blogosphere and the 2004 us election: divided they blog. Int. Workshop on Link discovery (2005)
3. Arora, S., Hazan, E., Kale, S.: A fast random sampling algorithm for sparsifying matrices. Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (2006)
4. Bader, D.A., Kintali, S., Madduri, K., Mihail, M.: Approximating betweenness centrality. Algorithms and Models for the Web-Graph (2007)
5. Boutsidis, C., Drineas, P., Magdon-Ismail, M.: Near-optimal column-based matrix reconstruction. SICOMP (2014)
6. Brandes, U.: A faster algorithm for betweenness centrality. J. of Math. Sociology (2001)
7. Chen, M., Nguyen, T., Szymanski, B.K.: A new metric for quality of network community structure. HUMAN (2013)
8. Deshpande, A., Vempala, S.: Adaptive sampling and fast low-rank matrix approximation. Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (2006)
9. Fortunato, S.: Community detection in graphs. Physics Reports (2010)
10. Gaiteri, C., Chen, M., Szymanski, B., Kuzmin, K., Xie, J., Lee, C., Blanche, T., Neto, E.C., Huang, S.C., Grabowski, T., et al.: Identifying robust communities and multi-community nodes by combining top-down and bottom-up approaches to clustering. Scientific Reports (2015)
11. Girvan, M., Newman, M.E.: Community structure in social and biological networks. PNAS (2002)
12. Leskovec, J., Adamic, L.A., Huberman, B.A.: The dynamics of viral marketing. TWEB (2007)
13. Leskovec, J., Faloutsos, C.: Sampling from large graphs. ACM SIGKDD (2006)
14. Leskovec, J., Lang, K.J., Mahoney, M.: Empirical comparison of algorithms for network community detection. WWW (2010)
15. Madduri, K., Ediger, D., Jiang, K., Bader, D., Chavarria-Miranda, D.: A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets. IPDPS (2009)
16. Mahoney, M.W.: Randomized algorithms for matrices and data. Foundations and Trends® in Machine Learning (2011)
17. Mahoney, M.W., Drineas, P.: CUR matrix decompositions for improved data analysis. PNAS (2009)
18. Newman, M.E.: Modularity and community structure in networks. PNAS (2006)
19. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. PRE (2004)
20. Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. CIKM (2009)
21. Satuluri, V., Parthasarathy, S., Ruan, Y.: Local graph sparsification for scalable clustering. SIGMOD (2011)
22. Spielman, D.A., Srivastava, N.: Graph sparsification by effective resistances. SICOMP (2011)
23. Wang, T., Chen, Y., Zhang, Z., Xu, T., Jin, L., Hui, P., Deng, B., Li, X.: Understanding graph sampling algorithms for social network analysis. ICDCSW (2011)
24. Yang, J., Chen, Y.: Fast computing betweenness centrality with virtual nodes on large sparse networks. PloS (2011)
25. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowledge and Information Systems (2015)
26. Zachary, W.W.: An information flow model for conflict and fission in small groups. JSTOR (1977)