Application-Specific Graph Sampling for Frequent Subgraph Mining and Community Detection

Sumit Purohit, Sutanay Choudhury
Pacific Northwest National Laboratory
Richland, WA, 99352, USA
Sumit.Purohit@pnnl.gov,
Sutanay.Choudhury@pnnl.gov

Lawrence B. Holder
Washington State University
Pullman, WA, 99164, USA
holder@wsu.edu

***Abstract*—Graph mining is an important data analysis methodology, but struggles as the input graph size increases. The scalability and usability challenges posed by such large graphs make it imperative to sample the input graph and reduce its size. The critical challenge in sampling is to identify the appropriate algorithm to insure the resulting analysis does not suffer heavily from the data reduction. Predicting the expected performance degradation for a given graph and sampling algorithm is also useful. In this paper, we present different sampling approaches for graph mining applications such as Frequent Subgrpah Mining (FSM), and Community Detection (CD). We explore graph metrics such as *PageRank*, *Triangles*, and *Diversity* to sample a graph and conclude that for heterogeneous graphs *Triangles* and *Diversity* perform better than degree based metrics. We also present two new sampling variations for targeted graph mining applications. We present empirical results to show that knowledge of the target application, along with input graph properties can be used to select the best sampling algorithm. We also conclude that performance degradation is an abrupt, rather than gradual phenomena, as the sample size decreases. We present the empirical results to show that the performance degradation follows a logistic function. Original Datasets, implementation of sampling algorithms, and results are available online.** [1]

***Keywords*-Graph Mining; Sampling; Scalability**

I. Introduction

Graphs are a natural and flexible representation of a set of entities and the relationships among them. Recent advancements in the fields of social science, process automation, and mobile computing have increased the interest in the fields of graph theory, graph modeling, and graph mining. However, the ever-increasing size of graphs from these domains has made scalability a challenge.

Graph Mining is the process of extracting knowledge from semi-structured graph datasets. The graph structure is as important as the attributes of the entities involved. Different graph mining operations are of interest such as Frequent Subgraph Mining, Community Detection, and Clustering. Each of these operations relies on different aspects of the input graph, and different graph properties affect the correctness and performance of the operations. Sampling is the technique to generate a smaller representative graph that can be used for the purpose of analysis instead of the original graph. There is a plethora of sampling techniques, each preserving some properties of the graph. The optimal sampling algorithm is the one that generates smaller graphs with high accuracy, high quality, and low analysis run-time.

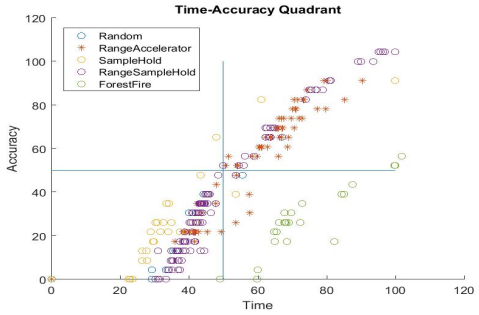

Figure 1. Performance Quadrants

Our contribution is to present the impact of input graph properties and application characteristics on the sampling process. We discuss the major approaches and their tradeoffs for given graph properties. We present empirical results to show that application-specific sampling approaches can generate sampled graphs with improved accuracy and runtime metrics as shown in Figure 1. We find that graph metrics such as motif and diversity can be used to sample heterogeneous graphs and that they perform better than degree based metrics as shown in Table I. Degree-based metrics such as *NodeRank* and *PageRank* are easy to calculate and can be used for homogeneous graphs. We conclude that a logistic function represents the performance degradation behavior. We measure that for some dense graphs it is possible to reduce problem size by 20%-30% without incurring more than 5% loss in accuracy. We present our results using three real-world graph datasets: (Citeseer, Internet P2P, Microsoft Academic Graph). We compare our approach with existing sampling methods such as Random, Sample and Hold, and Forest Fire.

Several approaches to graph sampling have been proposed in the literature. Al Hasan et al. [1] propose a generic sam-

[1] https://github.com/streaming-graphs/NOUS/tree/master/Sampling

Density	Heterogeneity	Metric(s)
Low	Low	Degree, PageRank
High	Low	PageRank
Low	High	Triangle, Diversity
High	High	Triangle

Table I
GRAPH METRICS FOR SAMPLING

pling framework that is based on the Metropolis-Hastings algorithm to sample the output space of frequent subgraphs [1]. Leskovec et al. [2] provide insight about required sample size, sampling method to use, and novel metrics to measure the goodness of the sampling method. Zou et al. [3] evaluate different sampling techniques and also provide sampling algorithms for Frequent Subgraph Mining (FSM) applications, similar to this research. However, our research reaches beyond the quantitative analysis of specialized sampling algorithms and attempts to predict the performance degradation and optimal sample size.

Gao et al. [4] propose Uniform Random Edge (URE) sampling to generate a sampled graph that accurately approximates various graph properties such as cut-set, volume, association, complement volume, and complement association. They present empirical results of their sampling approach on common graph mining tasks such as PageRank and Community Detection. Ahmed et al. [5] propose a generic stream sampling framework for big-graph analytics, called Graph Sample and Hold (gSH), which samples from massive graphs sequentially in a single pass, one edge at a time. There has been some recent work in *Task-Driven Sampling* that performs sampling for a specific task. Maiya et al. [6] propose a stratified sampling approach to identify community structure in the graph. Venu et al. [7] present a graph *sparsification* process for graph clustering. Our approach focuses on the graph properties and heterogeneity of the graph to identify the best sampling strategy.

II. GRAPH SAMPLING APPROACHES

A fundamental goal of any sampling approach is to create a representative sample of the input data. The selection of sampling method, sample size, and the stability of the sampled dataset as the sample size changes [2] are some of the parameters influencing sampling validity and performance. Some of the general purpose sampling approaches such as Random Sampling [8], Random Walk [9], and Forest Fire [10] are applicable to a wide range of applications and datasets.

All of the general purpose sampling approaches do not consider the target application and its requirements as one of the parameters. Many applications rely on specific graph properties. Graph Mining applications are dependent on graph structure, edge distribution, node distribution, edge growth rate, relative sub-graph frequency, etc. It is a process of identifying trends, signatures, and a summary of the graph. These are the examples of a comparative analysis that relies on the relative change in the graph properties rather than the absolute value of them. It is possible to exploit these characteristics while sampling a graph for a specific graph mining application.

FSM extracts frequent subgraphs relative to a support threshold. Intuitively for a given smaller graph, we can expect the same frequent subgraphs for a smaller threshold. Similarly, Community Detection identifies disjoint partitions of the vertices for a given size graph. For an optimal sample of the original graph, many of these communities should still exist although with weaker modularity.

This research concentrates on creating multi-dimensional sampling strategies keeping the target application in mind. The extent of the sampling impact on these applications is a topic of interest for this research. We focus on the FSM and CD applications and produce empirical results about the effect of various sampling approaches on them. We use *Accuracy*, *Run Time*, and *Interestingness* as the metrics to compare the performance of a sampled graph with respect to the original graph.

A. Node Frequency Based Sampling Algorithm

We introduce the Node Frequency-based Sampling (NFS) algorithm as a variant of Random Sampling. NFS samples an input graph based on the frequencies of the source and destination nodes of every edge. NFS works in two phases. In the first phase it computes the degree distribution for all the nodes. In phase one, NFS also identifies a sub-set of the nodes based on a *range* of the most and least available nodes in the graph. In the second phase, the same graph is traversed edge-by-edge and every edge is probabilistically skipped if the source and destination nodes of the edge are part of the subset created in phase 1.

This approach aggressively removes nodes but at the same time it has minimal negative impact on the correctness and performance of the FSM application. It removes some high frequency edges without changing the *frequent* status of a pattern. Similarly least frequent edges are also safely removed. The only patterns which rely on few low frequency edges, such as a *bridge edge* are prone to be tagged as *non-frequent*. A configurable scalar variable *accelerator* is also used to increase the rate at which nodes are excluded from the sampled graph. It is set to a default value of 1.

Similarly, it is possible to remove some high frequency nodes without changing the community structure. It does reduce the community size, but the reduction is proportional to the original community size. Algorithms 1, and 2 show the pseudo-code of major components of the NFS algorithms.

Algorithm 1 RangeAccelerator(G, R, α)

1: $n_d = vertexDegree(G)$
2: V_s = {v in V(G) s.t. d(v) $<$ R or d(v) $>$ (100-R) }
3: $G_s = SampleGraph(G, V_s, \alpha)$
4: return G_s

Algorithm 2 SampleGraph(G, V_s, α)

```
G_out = Empty Graph
for each e(u,v) ∈ G do
    if u in V_s OR v in V_s then
        P = α * max(d(u), d(v))/|E(G)|
        if random(0,1) > P then
            add e to G_out
        else
            skip e
```

B. Sample And Hold Algorithm

We also introduce a variant of NFS that uses the graph sample and hold technique [5]. Graph Sample and Hold is a one-pass algorithm that scans incoming edges one-by-one. For each incoming edge, the edge is identified as to whether the source or destination vertex of the edge is in the set of visible vertices so far. Sample and Hold algorithm sampling is governed by two scalar values p and q. If the edge is identified, it is selected using a probability q else it is selected with probability p.

C. Range With Sample Hold

RangeWithSampleHold is an improved version of the Graph Sample and Hold algorithm. In addition to the p and q it also uses a *range* parameter to identify graph vertices with degree in either very high or low value range. Algorithm 3 shows the pseudo-code of the approach.

Algorithm 3 RangeWithSampleHold(G, R, p, q)

```
V_vis = φ
G_out = φ
for each e(u,v) ∈ E(G) do
    if v in V(G) s.t. (d(v) <R ) ∨ (d(v) > (100-R)) then
        if u in V_vis OR v in V_vis then
            Add edge to G_out with probability q
            Add u,v to V_vis
        else
            Add edge to G_out with probability p
            Add u,v to V_vis
```

This algorithm processes every graph edge for which the associated nodes appear in the higher or lower range of the degree distribution. All such edges are either retained in or excluded from the output graph using the sample and hold approach.

D. Non-degree based Graph Sampling

All the methods above use degree distribution of the graph for the sampling. We identified that as the diversity and heterogeneity of the graph increases, other graph metrics case be use to sample the graph. We developed abstract version of the sampling approaches that takes a graph metric, and a filter function to sample the same graph. We looked at PageRank, Triangles, and Diversity of a given vertex to include or skip it in the sampled graph. It does require a preprocessing step where the graph metrics is calculated. The Microsoft Academic Graph [11] is a heterogeneous graph used for this analysis and the results presented in the Experiment section show that diversity, and triangle based sampling performs better than degree based sampling.

III. Experiments

Experimentation is done on a 3.1 GHz Intel Core i5 OS X Yosemite machine with 16GB DDR3 RAM and 4 Cores. Apache Spark is used to load the graph and compute graph statistics such as degree, page rank, diversity, and number of triangles. GraMi [12] and NOUS [13] are used as the frequent subgraph mining tool. Multiple runs of the algorithms are executed using different datasets to confirm the research findings and remove any dataset bias.

Community Detection experimentation is done using the C++ implementation of the Louvain method [14]. *SampleHoldRange* sampling is applied to create different sampled datasets of the original graph.

A. Citeseer Dataset

The Citeseer dataset [15] represents the citation network among research papers from. This network categorizes 3,312 papers into six domains and the interaction is represented by 4,782 edges. Five different sampling algorithms 1) Random Sampling, 2) SampleHold, 3) Range with Accelerator, 4) RangeWithSampleHold, and 5) ForestFire are used to generate multiple sampled graphs for each algorithm. All the relevant parameters such as p, q, R, α are varied in data generation scripts to create different size graphs. Resultant sampled graphs are used as an input to the FSM application using the GraMi.

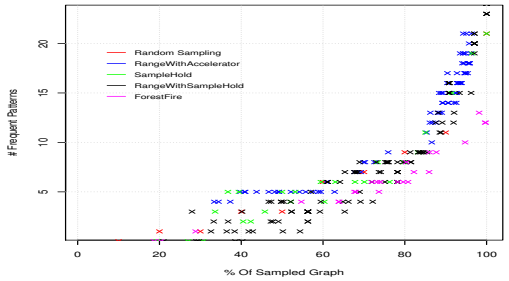

Figure 2. Citeseer Dataset Sampling Results

As expected, a high value of p generates an aggressively sampled graph with more than 70% edges of the original graph. Whereas a low value of p yields a graph with size dependent on the q value.

We also observed that the graph sampling is more sensitive to p than q. For a given p, the rate of decrease in the correctness is proportional to the decrease in the value of q. For a given q, the number of identified frequent patterns decreases sharply as p decreases. As shown in Figure 2 we observe that application specific *SampleHold* sampling performs better than general purpose *Random Sampling*. Also for any sampled graph size, the *SampleHoldRange* sampling approach performs better than SampleHold for the accuracy metric.

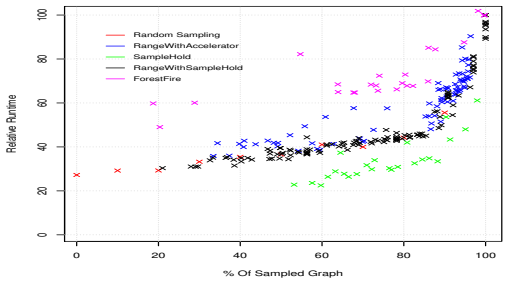

Figure 3. Citeseer Dataset Runtime Results

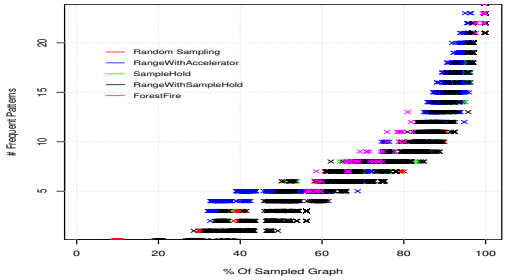

Figure 4. Performance over 30 Iterations

In addition to the accuracy metric shown above, run-time is also recorded for the Citeseer dataset in Figure 3. At the initial decrease in the sampling percentage of a sparse graph, the run-time decreases sharply similar to the sharp decrease in the accuracy shown in Figure 2. Gradually the run-time plot flattens as the sampled graph size decreases. The RangeWithSampleHold and SampleHold algorithms attain better results than any other sampling algorithm. Analysis stability is confirmed by multiple iterations of the experiment reaffirming the expected trends as shown in Figure 4.

Quality of result is another important performance metric while selecting a sampling algorithm. We analyzed the quality as an information theoretic measure that represents the loss of interesting patterns in the sampled graph. We are interested to discover how the sampling approaches perform in preserving larger size patterns. It is easy to retain smaller-size patterns as we reduce the sample size, but larger size patterns are the first ones to disappear. There is no strong conclusion that any of the mentioned algorithms retain larger size patterns. Although RangeAccelerator and RangeSampleHold do a slightly better job. For the existing algorithms, the larger size patterns disappear very early in the sampling process whereas for the suggested algorithms they do show up even for smaller size graphs and disappear gradually as shown in Figure 5.

We now turn to the Community Detection task and evaluate impact of the different sampling methods on this task. RangeWithSampleHold sampling algorithm is used to generate sampled graphs in *edge list* format. Multiple such graphs are used to generate communities using the Louvain

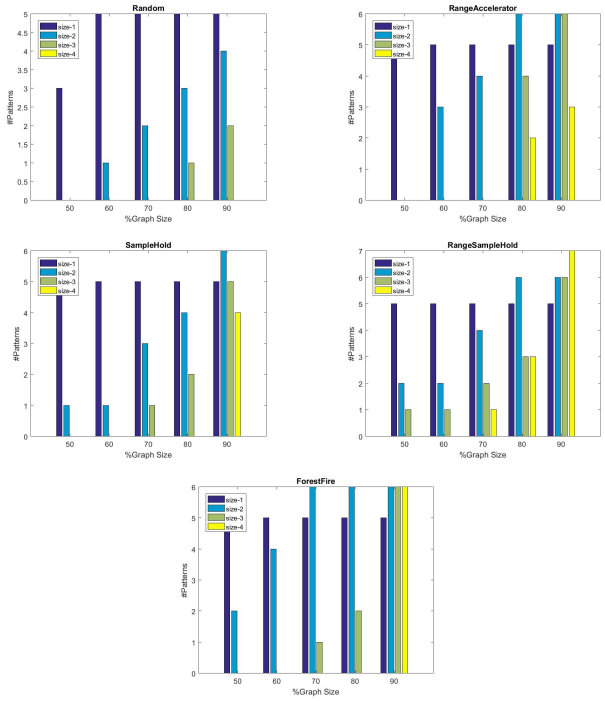

Figure 5. Citeseer Dataset Quality Results

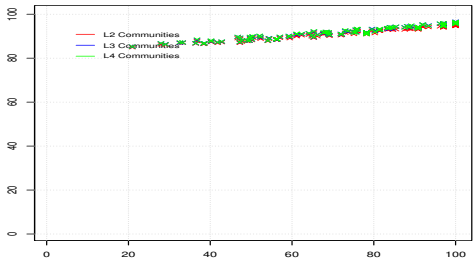

Figure 6. Community Detection Quality using RangeWithSampleHold

method. Louvain is an iterative method and it generates communities at the end of each iteration. These iterations are called "Levels" where *Level 1* represents each vertex in its individual community. In every subsequent iteration, each vertex is either re-assigned to a neighbor community or kept in its original community.

Pairwise comparison of the community IDs of neighboring vertices is used to quantify the sampling effect on the quality of the community detection. A one-pass algorithm iterates over all the vertices sequentially and the quality is defined as a fraction of preserved community pairs.

Normalize Accuracy is used as the performance metric. It is calculated as a ratio of the number of discovered communities to the total number of communities in the original graph at a given level. Figure 6 shows that community detection algorithms show a linear reduction in the accuracy with sample size. The number of communities formed by lesser numbers of vertices is less sensitive to the input graph size.

The majority of real-world graph datasets are found to be sparse in nature such as the above mentioned dataset. In absence of a *perfect* sampling algorithm a sudden drop in the frequent subgraphs is observed as more and more edges are excluded from the sampled graph. Empirical results show that *RangeWithSampleHold* performs better than various other sampling algorithms. For a given sample size, it identifies more frequent patterns. The following subsection shows that the *RangeWithSampleHold* algorithm can also be applied to dense graphs.

B. Internet peer-to-peer network

The Gnutella peer-to-peer network from August 5, 2002 [16] is a dense graph which is a sequence of snapshots of the Gnutella peer-to-peer file sharing network. Nodes represent hosts in the Gnutella network topology and edges represent connections between the Gnutella hosts. This graph contains 8846 nodes and 31839 edges with an average degree close to ten. *RangeWithSampleHold* algorithm is used to create multiple sampled graphs. NOUS [13] is used to identify frequent patterns as a function of sampled graph size. To observe the impact of graph density on sampling performance, the original dense graph is inflated further by adding random edges to the graph. All such induced graphs are also mined by NOUS to identify frequent sub-graphs. The experiment is repeated at different support values. Figure 7 shows the performance trends for the Gnutella p2p graph with average degree as 50. Different support values used are 250, 2000, and 4500. Corresponding trends are labeled as $S250\_D50, S200\_D50$, and $S4500\_D50$, respectively. Similarly Figure 8 shows the performance trends for the original Gnutella p2p network of average degree 10.

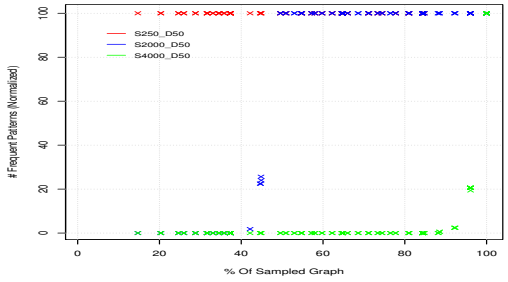

Figure 7. Modified P2P Dense Network Dataset Sampling Results

For a given dense graph the performance trends can be categorized in different zones. These trends have a boundary where the sample graph size is either closer to original graph size or an empty graph. As shown in Figures 7 and 8, it can be interpreted as three such zones. In Figure 7, plots $S250\_DS50$ and $S2000\_DS5$ show that the effective correctness of the mining operations exhibit very little change when the sampled graph size is either closer to original graph size or an empty graph. For a given *appropriate* support value the correctness does not degrade as the sample size is reduced because the exclusion of many edges does not force a frequent pattern to be an infrequent one. Here an *appropriate* support value is a dataset-specific integer number that produces many interesting patterns. For a very low support value, this behavior extends to an even lower range of sampled dataset size. For a higher support value, a sudden drop is expected because every excluded edge may force a specific frequent pattern to become infrequent.

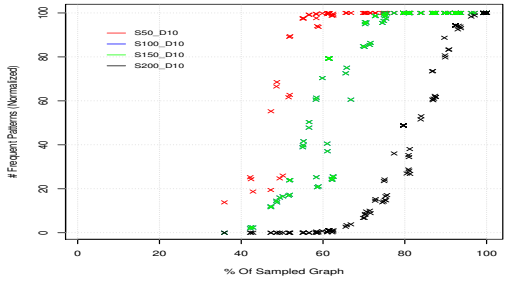

Figure 8. P2P Network Sampling Results at Different Support Values

C. Microsoft Academic Graph (MAG)

The Microsoft Academic Graph [11] is a heterogeneous graph containing scientific publication records, citation relationships between those publications, as well as authors, institutions, journals, conferences, and fields of study. We constructed a graph of all the papers published in VLDB, SIGKDD, and CIKM conference in the year 2010. It is a dense graph of 10K nodes and 40K edges. We picked the best performing degree based sampling algorithm RangeSampleHold and swap different graph metrics with degree distribution of the graph used in line 4 of the algorithm *RangeWithSampleHold*. Results show that diversity and triangle based sampling perform better than the degree based sampling consistently as shown in Figure 9. Triangle based sampling is also found to be more sensitive towards star shape patterns than multi-hop patterns.

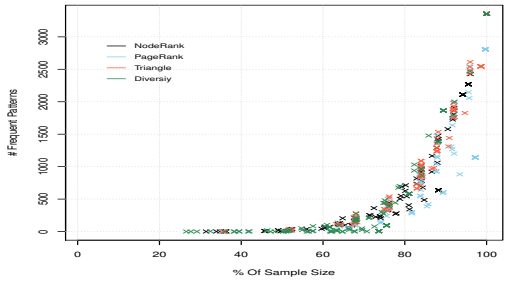

Figure 9. Degree and Non-degree based Sampling

D. Predictive Modeling for Expected Correctness

Experiments above corroborate the expected trend that the application specific sampling methods perform better than general purpose methods. It is also evident that the dense graphs can be sampled more aggressively than a sparse graph. To generalize the observations it is important to identify a multi-dimensional function that can predict the expected correctness of the output. We observe that the

logistic function of the form:

$$f(x) = a/(1 + be^{-cx}) \quad (1)$$

best fits the empirical results. Coefficients of the logistic function also exhibit a trend over graph properties. Sampling a dense graph shows a slow rate of decrease in the correctness, and this leads to higher values of a, b, and lower values of c. In preliminary results all the coefficients show a linearly decreasing trend as the graph density decreases. Future work will find different functions to predict values of a, b, and c.

IV. Conclusions

Graph Mining is a complex and challenging topic given the huge volume of available structured data. Many domains deal with high volume, high velocity, heterogeneous data. These domains can benefit from the extra knowledge about the underlying graph mining algorithm and its data requirements. This research provides extra knowledge that is useful for domain-users to make intelligent trade-offs about scalability and accuracy. This research shows that for some dense graphs it is possible to reduce problem size by 20%-30% without incurring more than 5% loss in accuracy. This guidance is valuable when combinatorial complexities of graph mining operations make it difficult to perform the analysis at scale. The graph properties also influence the sampled output and they can be factored into the trade-off function. This research finds a logistic function best represents the trend and is valuable for the users in deciding how to interpret their results and how to extrapolate improvements in results if problem size is increased.

Future work will use graph metrics to develop a machine learning model to predict optimal sample size. We will use billion scale stochastic Kronecker Graphs for the training. We will also improve upon existing sampling algorithms.

V. Acknowledgment

This material is based on work supported by the National Science Foundation under Grant No. 1646640. The research is also partially supported by the Analysis in Motion Initiative at Pacific Northwest National Laboratory. We thank Khushbu Agarwal at PNNL for her help in preparing the MAG Dataset.

References

[1] M. Al Hasan and M. J. Zaki, "Output space sampling for graph patterns," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 730–741, 2009.

[2] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 631–636.

[3] R. Zou and L. B. Holder, "Frequent subgraph mining on a single large graph using sampling techniques," in *Proceedings of the eighth workshop on mining and learning with graphs*. ACM, 2010, pp. 171–178.

[4] R. Gao, H. Xu, P. Hu, and W. C. Lau, "Accelerating graph mining algorithms via uniform random edge sampling."

[5] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella, "Graph sample and hold: A framework for big-graph analytics," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1446–1455.

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

[7] V. Satuluri, S. Parthasarathy, and Y. Ruan, "Local graph sparsification for scalable clustering," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 721–732.

[8] M. P. Stumpf, C. Wiuf, and R. M. May, "Subnets of scale-free networks are not scale-free: sampling properties of networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 12, pp. 4221–4224, 2005.

[9] W. Wei, J. Erenrich, and B. Selman, "Towards efficient sampling: Exploiting random walk strategies," 2004.

[10] J. Leskovec, J. Kleinberg, and C. 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*. ACM, 2005, pp. 177–187.

[11] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-j. P. Hsu, and K. Wang, "An overview of microsoft academic service (mas) and applications," in *Proceedings of the 24th international conference on world wide web*. ACM, 2015, pp. 243–246.

[12] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 517–528, 2014.

[13] S. Choudhury, K. Agarwal, S. Purohit, B. Zhang, M. Pirrung, W. Smith, and M. Thomas, "Nous: Construction and querying of dynamic knowledge graphs," in *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 2017.

[14] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[15] "Citeseer Dataset," https://linqs.soe.ucsc.edu/node/236, accessed: 2017-08-18.

[16] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.