Fast Stochastic Block Partitioning via Sampling

Frank Wanye
Dept. of Computer Science
Virginia Tech
Blacksburg, VA, USA
wanyef@vt.edu

Vitaliy Gleyzer
MIT Lincoln Laboratory
Lexington, MA, USA
vgleyzer@ll.mit.edu

Wu-chun Feng
Dept. of Computer Science
Virginia Tech
Blacksburg, VA, USA
wfeng@vt.edu

***Abstract*—Community detection in graphs, also known as graph partitioning, is a well-studied NP-hard problem. Various heuristic approaches have been adopted to tackle this problem in polynomial time. One such approach, as outlined in the IEEE HPEC Graph Challenge, is Bayesian statistics-based stochastic block partitioning. This method delivers high-quality partitions in sub-quadratic runtime, but it fails to scale to very large graphs. In this paper, we present sampling as an avenue for speeding up the algorithm on large graphs. We first show that existing sampling techniques can preserve a graph's community structure. We then show that sampling for stochastic block partitioning can be used to produce a speedup of between $2.18\times$ and $7.26\times$ for graph sizes between $5,000$ and $50,000$ vertices without a significant loss in the accuracy of community detection.**

***Index Terms*—community detection, GraphChallenge, sampling, stochastic block partitioning**

I. Introduction

Many real-world datasets from sources like social media, the world wide web, communication networks, and biological systems can be represented as graphs [1], where related items in the datasets are connected through edges. In such datasets, items can generally be grouped, where items in a group are more strongly connected to each other than to items in other groups [2]. The process of identifying such groups is known as community detection, graph partitioning, or graph clustering.

Graph partitioning has a multitude of applications in the real world. For example, clustering graphs of web clients can help inform the placement of servers [3]. Identifying communities of customers with similar purchasing habits is used in product recommendation systems [4]. Analyzing the community structure of a graph has also been shown to benefit classification tasks [5], [6]. In parallel computing, graph partitioning is used to minimize the communication overhead when assigning workloads to computing resources [7].

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

This work was supported in part by NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC).

Optimal graph partitioning is an NP-hard problem [2], making it feasible for only small graphs. In order to perform graph partitioning on real-world graphs, which often contain millions of vertices and edges, several sub-optimal approaches have been developed. These approaches are based on heuristics such as the minimum-cut, similarity measures, modularity maximization, and statistical inference [2].

Recently, much of the work involving community detection has used modularity maximization. However, modularity maximization has a resolution limit on the size of detectable communities and requires prior knowledge of the number of communities in the graph [8]. To address these drawbacks, we focus on stochastic block partitioning, as proposed in [8]–[10]. This approach is based on statistical inference, rather than modularity maximization, and incorporates a model selection algorithm. Thus, it does not suffer from the drawbacks of modularity maximization; but its algorithmic complexity of $O(E\log^2 E)$, where E is the number of edges [8], makes it infeasible for use in processing huge graphs.

Several approaches have been developed to improve the runtime of stochastic block partitioning, including streaming the graph [11] and parallelizing it on multi-core and multi-node clusters [12]. However, the first approach requires that the graph be stored in multiple parts, while the second requires computing resources that are not always available to researchers and analysts.

In this paper, we introduce *sampling* as an alternative means to improve the runtime of stochastic block partitioning on multi-core (and multi-node) clusters; furthermore, it does not require modification to the graph data structure and layout. Sampling has been previously shown to preserve multiple graph parameters, including degree distribution and clustering coefficient [13]. It has also been applied to other graph partitioning methods to decrease the algorithmic runtime in [14].

Our research contributions are as follows:

- We show that a randomly sampled subgraph can preserve much of the original graph's useful community structure, as evidenced by the minimal difference in resulting accuracy of partitioning and number of blocks found.
- We show that, with a reasonable sample size, we can combine sampling with stochastic block partitioning to partition a full graph with a speedup of up to $7.26\times$ on a graph with $50,000$ vertices.

II. Approach

In this section, we provide background information on stochastic block partitioning (SBP) and sampling from graphs, followed by our sampling method, as applied to SBP.

A. Background

Stochastic Block Partitioning (SBP): SBP is a community detection method that is based on the degree-corrected stochastic blockmodel, as introduced in [15]. A stochastic blockmodel is a representation of graph structure in the form of connections between groups of vertices called blocks, which correspond to communities in the community detection context. The model assumes that all vertices in a given community are homogeneous in terms of their degree; however, this assumption does not hold for many real-world graphs, whose vertex degrees are very heterogeneous. The degree-corrected stochastic blockmodel addresses this shortcoming by incorporating this variation into the blockmodel.

The algorithm for stochastic block partitioning (SBP) performs inference and model selection on the degree-corrected stochastic blockmodel. The blockmodel is initialized with every vertex as a separate block. A Fibonacci search is then performed to find the optimal number of blocks in the blockmodel, based on the overall description length of the blockmodel [8], [16].

Each step of the Fibonacci search consists of two phases: merging blocks together and fine tuning of the block membership for individual vertices. Both phases of the Fibonacci search leverage the Metropolis-Hastings algorithm, wherein a new block membership/block merge is proposed for every vertex/block and is probabilistically accepted or rejected based on the change in entropy of the degree-corrected blockmodel upon application of the proposal [16]. There are three key differences between the block merge and fine-tuning phases, as presented in [8].

1) *Granularity of the proposals.* The block-merge phase operates on the block level, where each proposal could change the block membership of all the vertices in a block. The fine-tuning phase operates on the vertex level, where each proposal affects a single vertex.
2) *Time of application of accepted proposals.* In the block-merge phase, the change in entropy is calculated for all proposals made and then the block merges are applied in a greedy manner. In the fine-tuning phase, each proposal is considered individually and applied immediately upon acceptance.
3) *Nature of the iterations.* In the block-merge phase, a fixed number of proposals are made per block, and a fixed number of merges are performed in every step of the Fibonacci search, resulting in a fairly deterministic runtime for the block-merge phase. In the fine-tuning phase, the number of iterations and the number of accepted proposals in a given Fibonacci search step are non-deterministic, though there are general trends that can be observed.

To better understand the bottlenecks in SBP, we analyzed the algorithm's runtime on a single core of an Intel Xeon CPU with 63 GB of RAM, using the official GraphChallenge datasets with 1k, 5k, 20k and 50k, vertices. Fig. 1 shows that the initial three Fibonacci search steps of the algorithm encompass a majority of the algorithm's runtime and that the percentage of the runtime that they encompass grows with the graph size. This is due to the initial expensive block-merge phases, where block-merge proposals are made and evaluated for a total of $1.75V$ blocks in the first three block-merge phases, where V is the number of vertices in the graph.

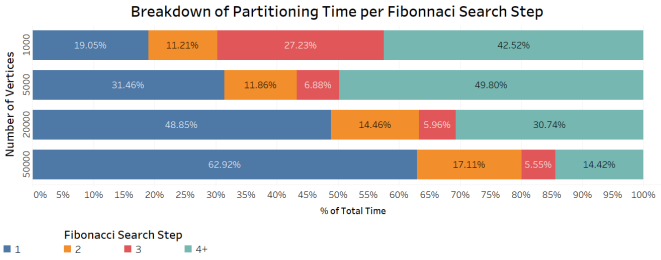

Fig. 1. Comparison of the time spent in each of the first three Fibonacci search steps and the remaining Fibonacci search steps for graph sizes between 1,000 and 50,000 vertices.

Sampling from Graphs: Data reduction through sampling is a well-known technique in data analytics. Considering the size of real-world graphs, which can consist of billions of edges and take up terabytes of disk space, the need for data reduction is evident. The challenge of sampling graphs is that graphs have many structural properties (e.g., vertex degree distribution, clustering coefficient, diameter, and weakly and strongly connected components), and the ability of sampling techniques to preserve each of these properties varies significantly, with no single technique able to preserve them all [13], [17].

In this paper, we study the following sampling algorithms, selected from those used in [13], [17] for their relatively short runtime and/or good overall performance and for their ability to retain the clustering coefficient of the full graph:

1) *Uniform Random (UR).* The sampled vertices are chosen uniformly at random and without replacement. This algorithm requires the least computation and space.
2) *Degree Weighted (DW).* This algorithm is like UR, but the vertices are *not* sampled with uniform probabilities. Instead, the probability of a vertex being sampled is proportional to the degree (i.e., number of incoming and outgoing edges) of the vertex. The algorithm requires $O(\text{number of vertices})$ additional space to store the sampling probabilities of each vertex.
3) *Random Node Neighbor (RNN).* This algorithm is like UR, but with the key difference being that vertices are sampled together with all of their outgoing neighbors.
4) *Random Walk (RW).* This exploration-based sampling technique begins by choosing a starting vertex v_{start} and follows its outgoing edges at random, adding the vertices encountered to the sample. If the vertex has no outgoing edges or probabilistically with a probability of 15% (for comparison with [13]), we restart the sampling from v_{start}.

When the number of vertices visited equals the desired sample size, we restart sampling from a new starting vertex. This algorithm requires $O(\text{number of vertices})$ additional space to prevent duplicate vertices from being added to the sample.

5) *Random Jump (RJ).* This is nearly identical to RW, but instead of restarting from v_{start} with a probability of 15%, RJ restarts sampling from a new starting vertex.
6) *Forest Fire (FF).* This sampling algorithm combines RW and RNN. We start with a vertex v_{start}. The outgoing edges of v_{start} are chosen probabilistically using a geometric distribution with a probability $p = 0.7$ (for comparison with [13]). All vertices on the ends of edges that are not selected are marked as visited. All other vertices are added to the sample. The algorithm then proceeds recursively for each of the newly sampled vertices until the sample size is reached, there are no newly selected vertices (the algorithm is stuck), or there are no vertices that have not been visited yet. When there are no newly selected vertices, the algorithm restarts from a new randomly selected starting vertex. When all the vertices have been visited, then in addition to restarting from a randomly selected vertex, the list of visited vertices is reset, to allow the algorithm to reach the desired sample size.

Other algorithms, like edge-based sampling, are not considered here because they insufficiently capture most graph properties [13], while algorithms like PageRank-weighted random sampling and ones based on the Metropolis-Hastings algorithm are iterative, and therefore more computationally expensive.

B. Sampling Method

In our proposed sampling approach, we implement each of the six (6) chosen sampling algorithms with sample sizes between 10% and 50%, inclusive, in increments of 5%. For each combination of sampling algorithm and sample size, we create a subgraph containing the sampled vertices as well as the edges between the chosen vertices. We then run the stochastic block partitioning (SBP) algorithm on the subgraph and report the results.

We argue that, intuitively, the best sampling algorithm for community detection can do the following:

1) Capture vertices from every community present in the graph.
2) Capture a proportionate number of vertices from the graph communities. For example, a sample size of 15% should capture approximately 15% of the vertices from every community.
3) Result in accuracy that is similar to that of the full graph (when community detection is performed on the sampled subgraph).

Thus, for the purposes of verifying our approach, we use datasets that include the true community membership for at least a subset of vertices in the graph.

C. Community Propagation

Our goal is to perform community detection on the entire graph, not just on the subset. To that end, the inferred community membership of the vertices in the subgraph needs to be extended to the remaining vertices in the full graph. To do so, we employ stochastic block partitioning (SBP) without the block-merge step. That is, we assume that the SBP performed on the subgraph identified the correct number of communities in the full graph, and therefore, perform only the fine-tuning phase of SBP.

Before the fine-tuning phase can be conducted, all the vertices in the graph must have an initial community membership label. In order to seed the fine-tuning phase, we assign to each of the remaining vertices, the community label of the community to which they are most strongly connected, in terms of the number of edges between the vertex and the identified communities. This approach is computationally cheap, and intuitively, provides a reasonably good estimate of the true community membership. A disadvantage of this approach is that the seeding heuristic has access to a limited view of the model, and future membership assignments can change the optimal assignment for previous vertices.

We evaluate our approach through comparisons of the runtime and F1 score of our approach versus running SBP on the entire graph.

III. Experiments

Here we describe our experimental approach and the metrics used to evaluate it, followed by the datasets and infrastructure on which we perform our experiments.

A. Experimental Approach

Due to the randomness inherent to the stochastic block partitioning (SBP) algorithm and the chosen sampling methods, the community detection results obtained vary each time the algorithm is run. As such, we run our sampling-based SBP algorithm 10 times for every combination of dataset $\{\text{A..L}\}$, sample size $\{10, 15, \ldots, 50\}$, and sampling algorithm $\{\text{DW}, \text{FF}, \text{RJ}, \text{RNN}, \text{RW}, \text{UR}\}$, resulting in $6,480$ experiments. The average values over the 10 algorithm runs for each combination of dataset, sample size, and sampling algorithm are then reported for each of the collected metrics.

B. Metrics for Evaluation of Samples

To quantify the efficacy of our sampling, we propose three new evaluation metrics.

1) *Percentage of captured communities.* This refers to the percentage of communities from which at least one vertex is included in the sample. Intuitively, the best sampling algorithms for community detection will capture all the communities present in the graph. However, this does not guarantee that the vertices sampled are useful for the community detection algorithm or even that there will be enough of them for the algorithm to detect them as a separate community.

2) *Difference from the exact uniform sample.* This is the average difference between an "exact uniform" sample and the sample actually obtained using each sampling method. The number of vertices in an exact uniform sample for a given community c, denoted as $V_c^{\text{exact uniform}}$, is defined as follows: given a graph where each community c is made up of v_c vertices and a sample size of $s\%$,

$$V_c^{\text{exact uniform}} = s\% \cdot v_c. \tag{1}$$

The actual number of vertices sampled from each community, V_c^{actual} is obtained by counting the true community membership of the vertices in the subgraph. The difference between the actual and ideal number of vertices, V^{diff} is then calculated as follows:

$$V^{\text{diff}} = \Sigma_c \frac{|V_c^{\text{exact uniform}} - V_c^{\text{actual}}|}{S}, \tag{2}$$

where S is the size of the subgraph in terms of the number of vertices and $V_c^{\text{exact uniform}}$ is the expected number of vertices in community c if the sample was exactly uniform. V^{diff} is invariant to scale, which is useful for comparing sampling performance across graphs of various sizes. Intuitively, the closer V^{diff} is to 0, the more representative the sample is of the communities in the full graph.

3) *F1 score on the subgraph.* Precision p and recall r are two of the pairwise metrics suggested in [8] for evaluating the performance of the SBP algorithm. The proposed F1 score is a function of both the precision and recall, where F1 score $= \frac{2pr}{p+r}$, and allows us to summarize both pairwise metrics at once. This is the most direct measure of how well a sampling algorithm captures the community structure of a graph, but it does not explain why the sampling algorithm does or does not do so.

C. Metrics for Evaluation of Sampling for Stochastic Block Partitioning on the Full Graph

Here we propose two metrics to evaluate sampling for stochastic block partitioning (SBP) on the full graph.

1) *Speedup of partitioning.* This defines the speedup of partitioning the full graph using our sampling approach over running SBP on the full graph. It is measured as $\frac{\text{average partitioning time without sampling}}{\text{average partitioning time with sampling}}$.
2) *F1 score on the full graph.* This measures the performance of our sampling approach on the full graph, after the community labels have been propagated from the sample to the full graph and then fine tuned. If a sampling algorithm captures enough of the graph's community structure, then the F1 score on the full graph with our sampling approach should be similar to the F1 score on the full graph when SBP is performed without sampling.

D. Datasets and Infrastructure

The Streaming Graph Challenge [8] provides a set of official synthetic graph datasets to evaluate the results of the SBP algorithm, with true community membership included for every vertex in the graphs. The graphs are unweighted and generated based on the degree-corrected stochastic blockmodel shown in [15]. The generative model allows the amount of overlap between communities and the size variation of said communities to be controlled via hyperparameter selection. Table I outlines the key characteristics of the chosen datasets.

TABLE I
DATASETS USED FOR EVALUATION

ID	Number of Vertices	Number of Edges	Community Overlap	Community Size Variation	Number of Communities
A	5000	50850	low	low	19
B	5000	50544	low	high	19
C	5000	51091	high	low	19
D	5000	51157	high	high	19
E	20000	473914	low	low	32
F	20000	476386	low	high	32
G	20000	475421	high	low	32
H	20000	473329	high	high	32
I	50000	1189382	low	low	44
J	50000	1193994	low	high	44
K	50000	1183975	high	low	44
L	50000	1187682	high	high	44

We conducted our performance evaluations on a 64-bit Intel Xeon CPU with a clock frequency of 3.50 GHz and 256 GB of memory, running the Debian GNU/Linux version 8 operating system.

IV. RESULTS

In this section, we present the results of our experiments, first in terms of evaluating the capacity of the various sampling techniques to capture the community structure of the provided graphs and then in terms of evaluating the speedup and accuracy of community detection when our sampling approach is used to process the full graph.

A. Evaluation of Samples

1) *Percentage of communities captured.* We found the performance of all six algorithms to be comparable with respect to this metric. As expected, the harder the graph is to partition, the larger the sample size needs to be in order to capture all the communities in the full graph. The lowest average percentage of communities found was 90.0%, using RW sampling on dataset D. On dataset A, which is similar in size to D, but with more distinct communities, the lowest average percentage of communities found was 97.89%, using RW sampling. For all algorithms, a large enough sample almost always captured all the communities in the full graph. Additionally, most of the time, as the graph size grew, the sample size needed to capture all the communities decreased for all the sampling algorithms. Fig. 2 shows how well the selected algorithms retain communities from the full graph across datasets A-L, for sample sizes between 10% and 50%.
2) *Difference from exact uniform sample.* Our experimental results, summarized in Fig. 3, show that almost universally,

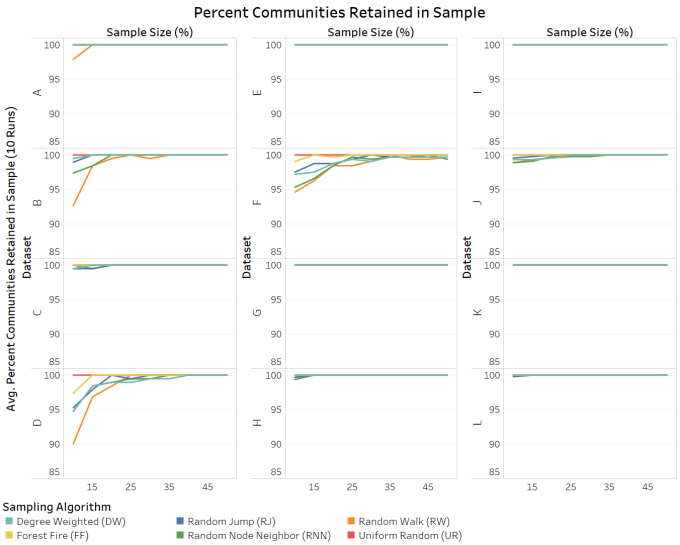

Fig. 2. The average percentage of blocks retained by each of the chosen sampling algorithms over 10 runs, for each of the datasets A-L, at different sample sizes.

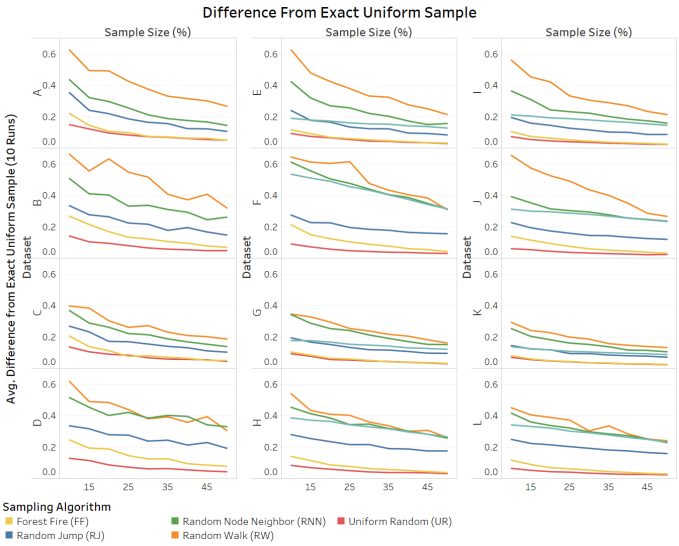

Fig. 3. The average difference from the exact uniform sample for each of the chosen sampling algorithms over 10 runs, for each of the datasets A-L, at different sample sizes.

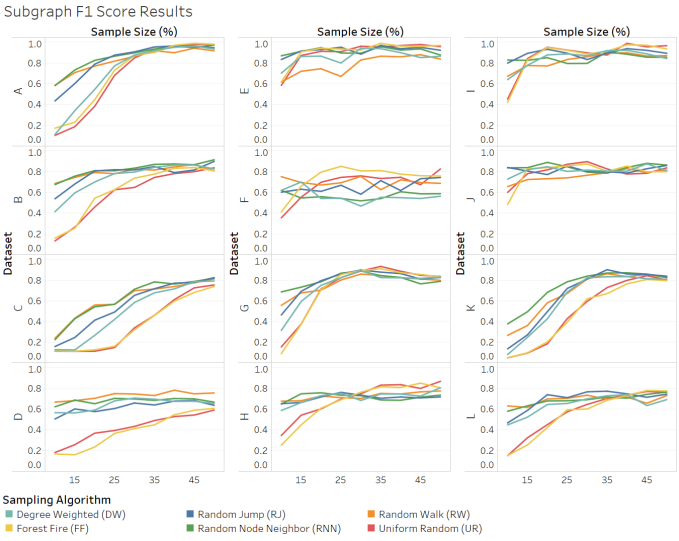

Fig. 4. The average F1 score obtained after running the SBP algorithm on the sampled subgraph over 10 runs, for each of the chosen sampling algorithms, for each of the datasets A-L, at different sample sizes.

the UR sampling algorithm produced samples that were closest to the exact uniform sample. This is to be expected, since a fully random algorithm is unbiased towards any of the graph's properties, while most of the other algorithms are either biased towards high-degree vertices (DW, RNN) or towards exploring larger, connected communities (FF, RW, RJ). FF is found to be the second-best performer, likely because vertices are added to the sample with a constant probability and the fact that previously visited vertices are not considered for addition to the sample, which forces the algorithm to explore a wider area of the graph than RW and RNN.

3) *F1 score on the subgraph.* Fig. 4 summarizes the F1 score of stochastic block partitioning (SBP) of the subgraph on graphs A-L. Our results show that, for smaller sample sizes of less than 30%, algorithms like RW, DW and RJ outperform UR and FF sampling. As the sample size increases, however, UR and FF begin to outperform all other sampling algorithms. The exceptions to this rule are datasets A-D, where UR and FF are consistently the worst-performing algorithms, and dataset K, where RNN is consistently the best-performing sampling algorithm in terms of F1 score on the subgraph. These results suggest that UR and FF perform better on larger graph sizes. Additionally, the performance of all five sampling algorithms on the subgraph generally stabilizes around sample sizes between 30% and 40%.

B. Evaluation of Sampling for Stochastic Block Partitioning on the Full Graph

1) *Speedup of partitioning.* The speedups obtained did not vary greatly with the community overlap or community size variation of the graphs, so we grouped our results (see Fig. 5) by the size of the graph in terms of the number of vertices in it. Our results show that speedup does not vary greatly with respect to the sampling algorithm, except in the case of the graphs with $5,000$ vertices. The average speedups obtained range from $2.18\times$ on the $5,000$ vertex graphs with a sample size of 50%, to $34.04\times$ on the $50,000$ vertex graphs with a sample size of 10%.

2) *F1 score on the full graph.* The F1 scores on the full graph, shown in Fig. 6, varied greatly based on the community overlap and community size variation configurations of the graph, the sampling algorithm, the sample size, and the size of the graph.

All six sampling algorithms performed poorly on datasets C and D, showing a marked decrease in F1

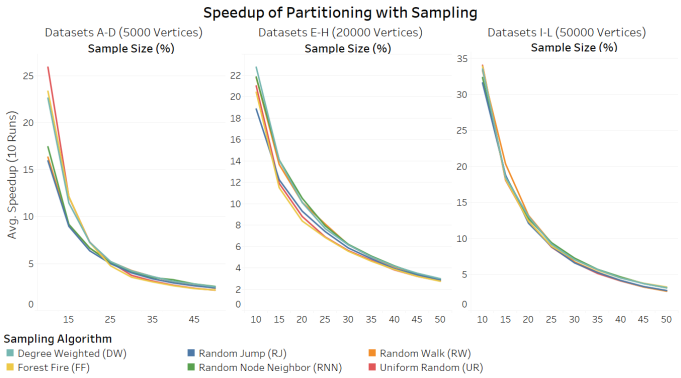

Fig. 5. The average speedup obtained using our sampling for SBP approach over 10 runs, for each of the chosen sampling algorithms, for datasets of size $5,000$, $20,000$ and $50,000$ vertices, at different sample sizes.

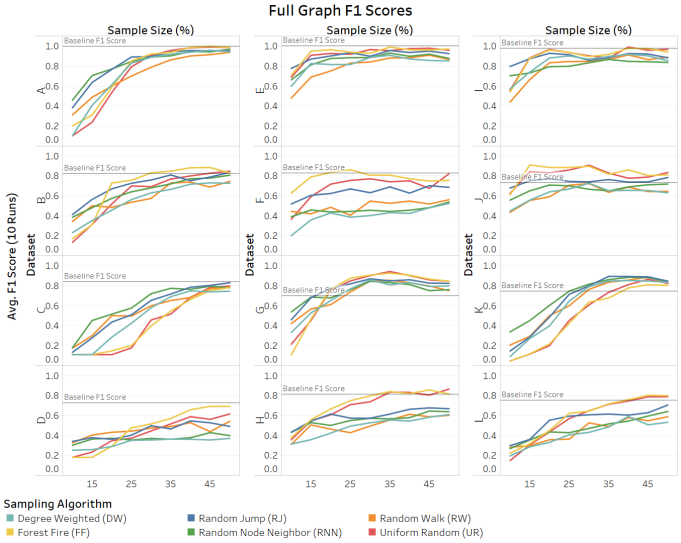

Fig. 6. The average F1 score obtained after using our sampling for SBP approach over 10 runs, for each of the chosen sampling algorithms, for datasets A-L, at different sample sizes. The horizontal lines represent the average baseline F1 scores over 10 runs, obtained by running the SBP algorithm without sampling on each dataset.

score even for large sample sizes. Datasets A and B show very little difference between F1 scores with and without sampling after sample sizes of 30% and 40%, respectively.

For the larger datasets E-L, the F1 score with UR and FF sampling is often higher than without sampling. The threshold at which this occurs varies with the community overlap and community size variation and the graph size. While this is not always the case, our sampling approach is always at least comparable to the baseline F1 score for samples of size 35% and above on datasets E-L.

Furthermore, our results show that UR and FF tend to outperform the other sampling algorithms on the larger sample sizes. FF tends to outperform UR, especially on dataset F, but in most cases the difference is not significant. For smaller sample sizes of less than 20%, RW, RJ, and in some cases, RNN, generally have better F1 scores than UR and FF. However, at such low sample sizes, none of the sampling techniques produce F1 scores comparable to the baseline.

V. Conclusion

Community detection is a graph analysis task with applications in areas ranging from cybersecurity to product recommendation systems. Due to the size of modern graphs and the inherent algorithmic complexity of the task, performing community detection on modern graphs without resorting to over-simplistic heuristics can be time consuming. In this paper, we focus on sampling as a means to bypass the computationally expensive initial block-merge phases of the stochastic block partitioning (SBP) algorithm, to perform community detection on the full graph. We perform our experiments on 12 synthetic datasets with varying characteristics and compare six sampling algorithms at sample sizes ranging from 10% to 50%.

We first show that sampling can preserve the community structure of large graphs. Our results show that uniform random (UR) sampling and forest fire (FF) sampling deliver the best performance out of the tested sampling algorithms with respect to preserving community structure, based on three different evaluation metrics. For smaller sample sizes, however, random node neighbor (RNN) sampling outperforms the UR and FF algorithms.

We then show that, using the Metropolis-Hastings algorithm and a simple seeding heuristic, we can propagate the community detection results from the sample to the full graph. Our approach leads to comparable, and in some cases, better community detection results than simply performing stochastic block partitioning (SBP) on the full graph. Furthermore, we identify the UR and FF sampling algorithms as the best performers out of the algorithms tested, further confirming our evaluations of the sampling algorithms. Our experiments suggest that a sample size of 30% is typically enough to achieve near-optimal community detection results, with a speedup between $5.56\times$ and $6.20\times$ on $20,000$ vertex graphs, and a speedup between $6.67\times$ and $7.28\times$ on $50,000$ vertex graphs. This further suggests that the speedup obtained grows with the graph size.

Future work involves applying our sampling approach to larger synthetic and real-world graphs to study the relationship between sample size and final community detection results at scale. We also seek to study the relationship between various sample evaluation metrics and the final community detection results to arrive at a definitive measure for evaluating samples for community detection purposes. Finally, more expensive sampling techniques like the Markov Chain Monte Carlo (MCMC) approach used in [18] could be tested, as an attempt to decrease the sample size needed to preserve community structure.

Acknowledgment

We thank Mohammed Hassan of the Synergy Lab at Virginia Tech for the many insightful discussions on this work.

REFERENCES

[1] M. Newman, "The structure and function of networks," *Computer Physics Communications*, vol. 147, no. 1-2, pp. 40–45, 2003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0010465502002011

[2] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010. [Online]. Available: https://doi.org/10.1016/j.physrep.2009.11.002

[3] B. Krishnamurthy, J. Wang, B. Krishnamurthy, and J. Wang, "On network-aware clustering of Web clients," in *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4. Stockholm, Sweden: ACM, 2000, pp. 97–110. [Online]. Available: http://portal.acm.org/citation.cfm?doid=347057.347412

[4] P. Krishna Reddy, M. Kitsuregawa, P. Sreekanth, and S. Srinivasa Rao, "A Graph Based Approach to Extract a Neighborhood Customer Community for Collaborative Filtering," in *Databases in Networked Information Systems*. Springer, Berlin, Heidelberg, 2002, pp. 188–200. [Online]. Available: http://link.springer.com/10.1007/3-540-36233-9_15

[5] G. Rizos, S. Papadopoulos, and Y. Kompatsiaris, "Multilabel user classification using the community structure of online networks," *PLOS ONE*, vol. 12, no. 3, p. e0173347, 2017. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0173347

[6] G. Li, D. Zhang, and Y. Li, "Packet Classification Using Community Detection," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. Guangzhou, China: IEEE, 2017, pp. 94–100. [Online]. Available: https://ieeexplore.ieee.org/document/8367253/

[7] G. M. Levchuk and J. Colonna-Romano, "Optimizing collaborative computations for scalable distributed inference in large graphs," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVII*, I. Kadar, Ed., vol. 10646. SPIE, 2018, p. 23. [Online]. Available: https://doi.org/10.1117/12.2305872

[8] E. Kao, V. Gadepally, M. Hurley, M. Jones, J. Kepner, S. Mohindra, P. Monticciolo, A. Reuther, S. Samsi, W. Song, D. Staheli, and S. Smith, "Streaming graph challenge: Stochastic block partition," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–12. [Online]. Available: http://ieeexplore.ieee.org/document/8091040/

[9] T. P. Peixoto, "Parsimonious Module Inference in Large Networks," *Physical Review Letters*, vol. 110, no. 14, 2013. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.110.148701

[10] ——, "Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models," *Physical Review E*, vol. 89, no. 1, p. 012804, 2014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.89.012804

[11] A. J. Uppal and H. H. Huang, "Fast Stochastic Block Partition for Streaming Graphs," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8547523/

[12] A. J. Uppal, G. Swope, and H. H. Huang, "Scalable stochastic block partition," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/8091050/

[13] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*. New York, New York, USA: ACM Press, 2006, p. 631. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1150402.1150479

[14] S. Yun and A. Proutière, "Community Detection via Random and Adaptive Sampling," in *Conference on Learning Theory*. Barcelona, Spain: Association for Computational Learning, 2014. [Online]. Available: http://proceedings.mlr.press/v35/yun14.pdf

[15] B. Karrer and M. E. J. Newman, "Stochastic blockmodels and community structure in networks," *Physical Review E*, vol. 83, no. 1, p. 016107, 2011. [Online]. Available: http://arxiv.org/abs/1008.3926

[16] T. P. Peixoto, "Parsimonious module inference in large networks," *Physical Review Letters*, vol. 110, no. 14, 2013. [Online]. Available: http://arxiv.org/abs/1212.4794

[17] T. Wang, Y. Chen, Z. Zhang, T. Xu, L. Jin, P. Hui, B. Deng, and X. Li, "Understanding Graph Sampling Algorithms for Social Network Analysis," in *2011 31st International Conference on Distributed Computing Systems Workshops*. IEEE, 2011, pp. 123–128. [Online]. Available: http://ieeexplore.ieee.org/document/5961350/

[18] A. S. Maiya and T. Y. Berger-Wolf, "Sampling community structure," in *Proceedings of the 19th international conference on World wide web - WWW '10*. New York, New York, USA: ACM Press, 2010, p. 701. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1772690.1772762