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.**

***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.

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 trade-offs 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 show 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 four real-world graph datasets: (Citeseer, Power Grid Topology, Internet P2P, Microsoft Academic Graph). We compare our approach with existing sampling methods such as Random, Sample and Hold, and Forest Fire.

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

Table I
Graph Metrics for Sampling

Several approaches to graph sampling have been proposed in the literature. Al Hasan et al. [2] propose a generic sampling framework that is based on the Metropolis-Hastings algorithm to sample the output space of frequent subgraphs [2]. Leskovec et al. [3] perform a thorough experimentation on several diverse graph datasets and provide insight about required sample size, sampling method to use, and novel

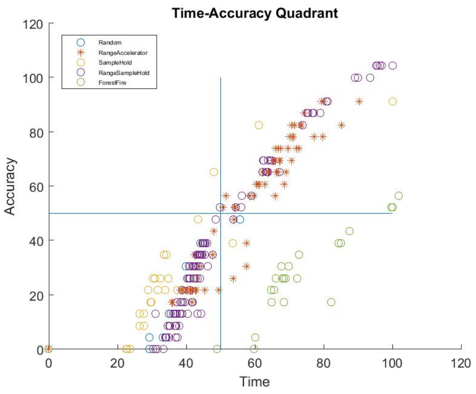


Figure 1. Performance Quadrants

metrics to measure the goodness of the sampling method. They also present relations between properties of the original and sampled graph. Zou et al. [4] 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. [5] 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. [6] 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. Lin et al. [7] and Wang et al. [8] published a survey of various sampling and summarization approaches for social networks. There has been some recent work in *Task-Driven Sampling* that performs sampling for a specific task. Maiya et al. [9] propose a stratified sampling approach to identify community structure in the graph. Relational Classification is used in social, biological, and information networks to understand relationships between entities. Ahmed et al. [10] propose a study to demonstrate the impact of sampling algorithms on the Relational Classification problem. Venu et al. [11] present a graph *sparsification* process for graph clustering. Sparsification is defined as the process that preserves all the nodes and reduces the edgeset only. Our approach focuses on the graph properties and heterogeneity of the graph to identify the best sampling strategy.

II. Problem Formulation

This section defines *Labeled Graph*, and *Sampled Graph*. It also describes the problems of Frequent Sub-Graph Mining (FSM), and Community Detection (CD) and provides their corresponding mathematical formulation. We use a directed graph without self-loops for the purpose of this research.

A. Definitions

Definition 1: Labeled Graph: A labeled graph is a directed graph that can be represented as 4-tuple. G = (V, E, L, l), where V is set of vertices, E is set of Edges such that E $\subseteq$ V x V, L is set of labels on vertices and edges, and I is a function V $\cup$ E $\rightarrow$ L that assigns labels to vertices and edges.

A *Sampled Graph* is an induced subgraph from the original graph intended to exhibit similar graph properties to the original graph. Any good sampling approach insures that the sampled graph has predictable performance metrics.

B. Frequent Sub-Graph Mining

The Frequent Subgraph Mining (FSM) application concentrates on identification of frequent subgraphs within a given graph data set [12] [13] [14] [15]. Different FSM approaches have been proposed that deal with different kinds of graphs and input setup. Algorithms such as gSpan [14], FSG [16], and Gaston [17] treat input graphs in a transactional setting. Holder et al. proposed SUBDUE [13] that uses the Minimum Description Length (MDL) principle to discover structures that compress the data and represent structural concepts in the data. Arabesque [18] is a distributed FSM implementation using MapReduce. NOUS [19] is another distributed subgraph mining implementation for dynamic knowledge graph streams. In many applications we need to find frequent subgraphs in a single large graph whereas some recent applications attempt to mine a continuous graph stream and find frequent subgraphs in it.

Definition 2: Frequent Subgraph Mining: Given a graph G = (V, E, L, l) and a scaler value *min_support*, the FSM problem is to find a set of unique subgraphs G_f where every subgraph has frequency greater than *min_support*.

For a given labeled graph, *min_support* can be defined in two ways. The first approach is to consider two subgraphs different if they have at-least one edge different. This leads to higher *min_support* value than the second approach. The second approach does not allow sharing any edge between two subgraphs, although the nodes can be shared.

Any definition of *min_support* must exhibit the downward closure property [18] [4]. This property restricts the selection of *min_support* to be a monotonic function. The monotonic property states that the frequency (i.e., support) of the super-graph must be a strictly increasing or non-decreasing function. This property is critical to prune the search space as the bigger frequent patterns are discovered.

Minimum Image-based Support (MIS) [20] [18] is one such metric, which defines the support of a pattern as the minimum number of distinct mappings for any vertex over all the pattern instances in the graph. As the pattern size grows, the MIS of larger patterns are never greater than the MIS of smaller sub-patterns. This attribute guarantees the required monotonic property.

Multiple metrics can be used to compare the performance of a sampled graph with respect to the original graph. *Accuracy* is one such metric that compares the number of frequent subgraphs identified in both the original and sampled graph. It is an easy to use, approximate but sufficient metric as it does not guarantee to identify the same frequent subgraphs in the sampled graph. *Run Time* is another such metric that gives information about how much time it takes to mine the entire graph and produce results. Accuracy and Run Time can be used as user-provided constraints in approximate graph mining of a single huge graph. We also looked at the *interestingness* of the identified frequent sub-graphs.

C. Community Detection

The community detection (CD) application identifies cohesive substructures in the input graph. The substructure is defined as an organization of the vertices and edges in the graph such that the vertices and the edges joining them form a cluster or community [21]. Given a graph, the problem of community detection is to compute a partitioning of vertices into communities that are closely related within and weakly related across communities. *Modularity* is used to measure the quality of communities detected [22] [23].

Definition 3 Community Detection: Given a graph G = (V, E, L, l), a *community* within the graph represents a subset of V. The problem of *community detection* is to compute a *disjoint* partition P of the graph that maximizes the modularity of all the communities.

A *partition* is a separation in the vertex space using a similarity function. Vertices are closely related in the same partition and weakly related across partitions.

Modularity is a statistical measure for assessing the quality of a given community-wise partitioning (or equivalently, *clustering*). Newman et al. [23] proposed a formal definition of modularity which is also used by Louvain's method [24], one of the most used methods for identifying communities in large networks. Modularity is a function of a set of all communities, and all edge-weight values. A *good* clustering method is one that clusters closely related elements (vertices) as part of the same community (or *cluster*) while separating weakly related elements into different clusters [22].

III. 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 [3] are some of the parameters influencing sampling validity and performance. This section introduces various sampling algorithms and their core sampling strategies.

A. General Purpose Approaches

Some of the general purpose sampling approaches are applicable to a wide range of applications and datasets. Random Sampling [25], Random Walk [26], and Forest Fire [27] samplings provide different ways of sampling a huge graph. Random Nodes selection sampling creates a sample graph by randomly selecting a set of nodes N with uniform probabilities. The resultant sampled graph is created by using only the sampled N nodes and edges around those nodes [4]. Random edges selection sampling approach randomly selects a subset of edges and the resulting sampled graph is constructed from the set of sampled edges. Random Walk Sampling (RWS) randomly picks a starting node and performs a random walk on the graph. New nodes are selected randomly on every step, keeping track of already selected nodes so that loops are avoided. Random Walk Sampling is biased towards high degree nodes. There are different variations of RWS and each returns a connected graph. *Random Walk With Restart* performs random walk and starts a new walk with a given probability. *Random Jump* uses a probabilistic jump to reach to any node in the network[26].

B. Application Specific Approaches

All of the commonly used 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 we can expect the same frequent subgraphs for a smaller graph 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. Various other applications such as link prediction, node/edge classification and regression are also expected to exhibit similar trends.

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.

C. 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 reads a graph and calculates the degree distribution for all the nodes. It can be configured to use *indegree*, *outdegree*, or both. For this research we used the combined degree of a node as its representative *degree* value. 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. For a given support value, it is possible to remove 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. *Accelerator* 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. Line 1 of Algorithm 1 computes the degree of all the vertices. n_d a list of all graph nodes where Each element of the list has (node_id, degree) entry. Line 2 gets a subset of vertices that have node frequency in the given range. The range is a value between 1 and 50. Line 3 creates the resulting graph by calling auxiliary function *SampleGraph*.

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 $G_s$
```

In Algorithm 2, Line 1 initializes the output graph with no edges. Line 2 initiates a linear iteration of the graph edges. The loop visits every edge of the graph and calculates node probability of the source and destination of the edge. The probability determines whether to skip or add the current edge in output graph. Line 4 uses α which is the accelerator that increases the rate at which nodes are excluded (or included). The default accelerator value is 1.

Algorithm 2 SampleGraph(G, V_s, α)

```
1: $G_{out}$ = Empty Graph
2: for each $e(u, v) \in G$ do
3:     if u in $V_s$ OR v in $V_s$ then
4:         $P = \alpha * max(d(u), d(v))/|E(G)|$
5:         if random(0,1) ¿ P then
6:             add $e$ to $G_{out}$
7:         else
8:             skip $e$
```

D. Sample And Hold Algorithm

We also introduce a variant of NFS that uses the graph sample and hold technique [6]. 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 in the graph stream processing. 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, the edge is selected with a probability *p*.

E. 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 RangeWithSampleHold(G, R, p, q)

```
1: $V_{vis} = \phi$
2: $G_{out} = \phi$
3: for each $e(u, v) \in E(G)$ do
4:     if v in $V(G)$ s.t. $(d(v)$ ¡ R ) $\vee$ $(d(v)$ ¿ (100-R)) then
5:         if u in $V_{vis}$ OR v in $V_{vis}$ then
6:             Add edge to $G_{out}$ with probability $q$
7:             Add u,v to $V_{vis}$
8:         else
9:             Add edge to $G_{out}$ with probability $p$
10:            Add u,v to $V_{vis}$
```

Lines 1 and 2 of Algorithm 3 initialize G_{out} and V_{vis}. Line 3 onward, each edge of the graph is visited by the algorithm and it is determined whether to add or skip that edge in the output graph. RangeWithSampleHold uses range R in addition to sample and hold probabilities p and q to make that decision.

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. By focusing on only the tail-end of the degree distribution, this algorithm shows preferential bias towards the subgraphs that form any pattern. As a result, this algorithm outputs a graph with higher numbers of frequent patterns of the given sample size.

F. Forest Fire Sampling

The Forest Fire (FF) sampling [3] is inspired by the work on temporal graph evolution. It is based on Forest Fire Model [27] that is a graph generation model. It randomly picks a *seed* vertex and picks edges the way fire progresses in forest. It starts with immediate neighbors and recursively picks their neighbors. It works on two probabilities p and r. p is *forward burning probability* and r is *backward burning probability*. Graph vertex v is selected randomly and we sample the subgraph originating from this vertex as explained in [3], [27].

ForestFire algorithm requires knowledge of the vertex neighborhood. Edges are selected one hop at a time. In a distributed graph processing framework, it is a challenge to optimally partition the graph such that vertex neighborhood information is available locally. All the other sampling approaches presented do not require extra neighborhood knowledge. They compute graph level properties based on vertex and edge distribution. This makes them more suitable for distributed graph processing as the vertex partitioning and neighborhood processing are expensive operations.

G. Non-degree based Graph Sampling

All the methods above use degree distribution of the graph to identify a vertex or edge to remove from the output graph. 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 [28] 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.

IV. Experiments

This section presents the experimental setup used to execute different sampling algorithms using various datasets. It also presents the empirical results and research findings. Experimentation is done on a 3.1 GHz Intel Core i5 machine with 16GB 1333 Mz DDR3 RAM and 4 Cores. It runs OS X Yosemite Version 10.10.3 with AMD Radeon HD 1024 MB Graphics. Apache Spark is used to load the graph and compute graph statistics such as degree, page rank, diversity, and number of triangles [29]. GraMi [15] and NOUS [19] 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 [24]. Original Citeseer dataset is converted to an *edge list* graph where each line contains a pair *"s_id d_id"*, where s_id is the id of the source vertex and d_id is the id of destination vertex. *SampleHoldRange* sampling is applied to create different sampled datasets of the original graph and the community detection algorithm is applied.

A. Citeseer Dataset

The Citeseer dataset [30] represents the citation network among research papers from different research domains. This network categorizes 3,312 papers into six domains and the interaction is represented by 4,782 edges.

These papers are classified into one of the following six classes: Agent based System, Artificial Intelligence, Database, Information Retrieval, Machine Learning, and Human Computer Interaction.

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 [15]. GraMi [15] is a framework for frequent subgraph mining in a single large graph that uses MIS.

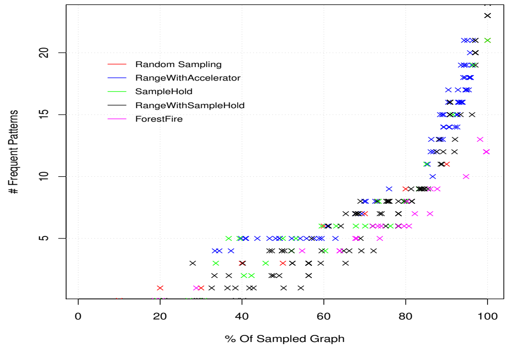


Figure 2. Citeseer Dataset Sampling Results

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.

Subsequently we also experimented with different values for the p and q and we observe 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

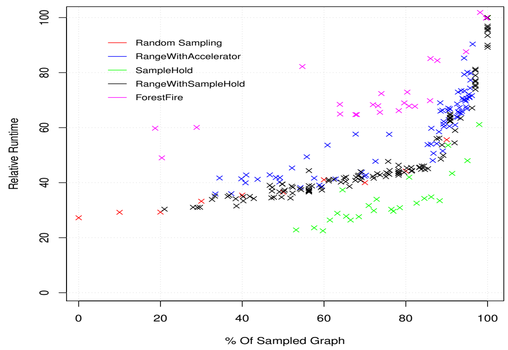


Figure 3. Citeseer Dataset Runtime Results

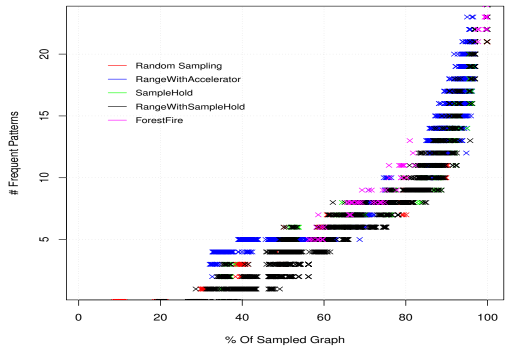


Figure 4. Performance over 30 Iterations

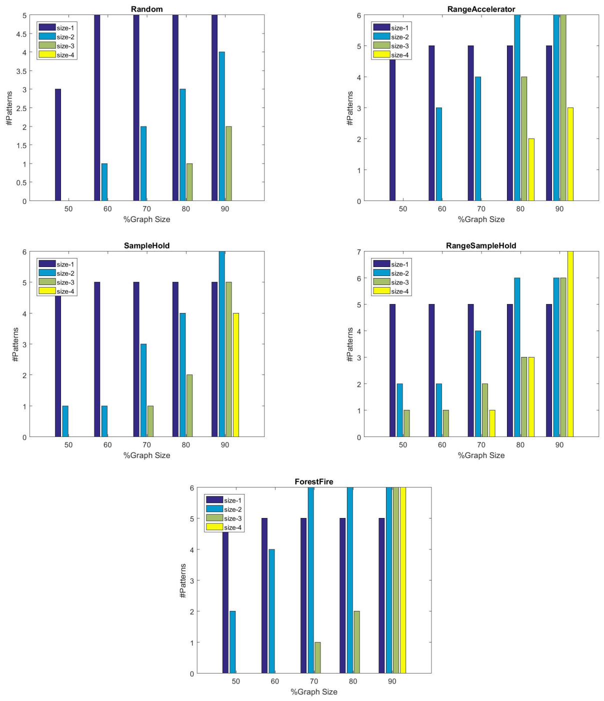


Figure 5. Citeseer Dataset Quality Results

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. *SampleHoldRange* can also be configured using different parameters such as p, q, R, α.
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 also used to generate sampled graphs in *edge list* format. Multiple such graphs are used to generate communities using the Louvain 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.
For all the pairs of vertices v_n and v_{n+1}, the quality Q is defined over all such pairs as: $Q = (s + s')/N$ where

- s = # of instances the vertex pair shares a community in the sampled graph if they share a community in the original graph

- s' = # of instances the vertex pair does not share a community in the sampled graph if they do share a community in the original graph
- N = # of total vertex pairs

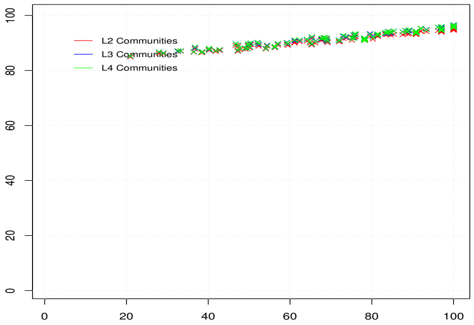


Figure 6. Community Detection Quality using RangeWithSampleHold

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.

B. *Power Grid Topology Dataset*

Another dataset used in out experimentation is an undirected, unweighted network representing the topology of the Western States Power Grid of the United States. Data was compiled by D. Watts and S. Strogatz and made available on the web [31]. The same algorithms are executed against the power grid data using GraMi. Results in Figure 7 show that there is a sharp decline in the accuracy even for a small amount of sampling. Although for any given sampling size the *RangeWithSampleHold* algorithm produces slightly better results.The run-time analysis of the FSM application for the Powergrid dataset in Figure 8 also confirms similar trends as observed in Figure 3.

The majority of real-world graph datasets are found to be sparse in nature such as the above mentioned two datasets. 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.

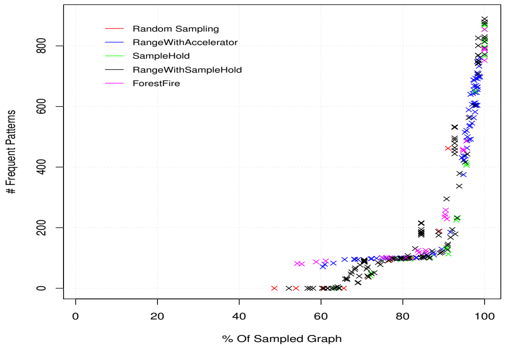


Figure 7. Power Grid Dataset Sampling Results

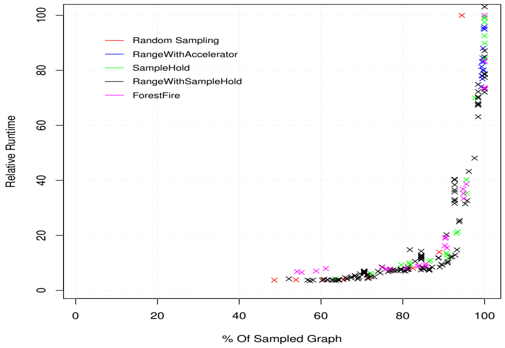


Figure 8. Power Grid Dataset Runtime Results

C. *Internet peer-to-peer network*

The Gnutella peer-to-peer network from August 5, 2002 [32] is a dense graph which is a sequence of snapshots of the Gnutella peer-to-peer file sharing network from August 2002. 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 [19] 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 9 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 10 shows the performance trends for the original Gnutella p2p network of average degree 10. Different support values are 50,100,150,and 200.

Corresponding trends are labeled as $S50, S100, S150$, and $S200$.

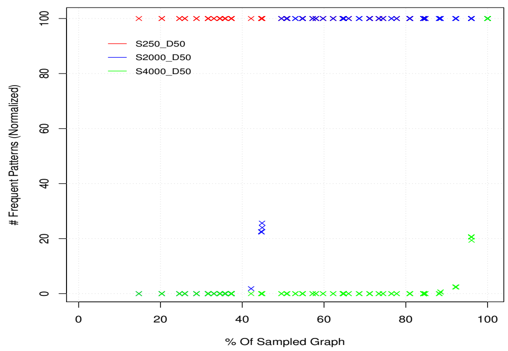


Figure 9. Modified P2P Dense Network Dataset Sampling Results

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 9 and 10, it can be interpreted as three such zones. In Figure 9, plots $S250_D S50$ and $S2000_D S5$ 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.

The rate of decrease in the resulting quality of a graph mining operation also depends on the support value. Graph mining at a higher support value is more sensitive to the sampling size. Whereas the lower support values have an advantage of using space-efficient samples without compromising the result quality as shown in plot $S250_D 50$ of Figure 9 and plot $S50$ of Figure 10. Intuitively, efficient sampling algorithms such as *RangeWithSampleHold* are more useful at higher support values where it is important to not miss frequent patterns as show in Figure 10. As stated earlier, the number of frequent patterns is an approximate performance metric. The total number of frequent sub-graphs found varies even in the original graph as we increase the support threshold. To show the relative trends at different support values, Figures 9 and 10 normalize the number of frequent patterns found in the sampled graph by dividing it by the maximum number of patterns found in the original graph.

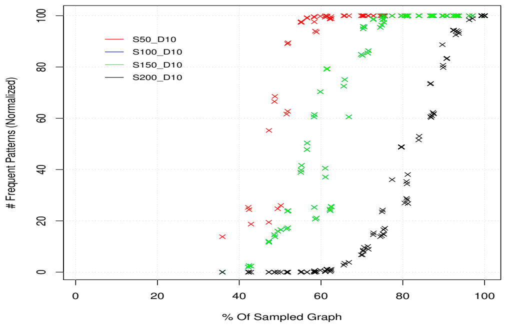


Figure 10. P2P Network Dataset Sampling Results at Different Support Values

D. Microsoft Academic Graph (MAG)

The Microsoft Academic Graph [28] 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 shows that diversity and triangle based sampling perform better than the degree based sampling consistently as shown in Figure 11. Triangle based sampling is also found to be more sensitive towards star shape patterns than multi-hop patterns.

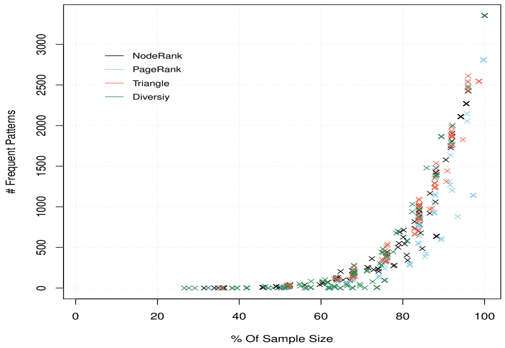


Figure 11. Degree and Non-degree based Sampling

E. 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 for the FSM application, 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. For the FSM application a predictive model $\mathcal{H}$ is defined as a set of functions over Graph Size, Density, and Support.

Definition 4: For the FSM application a predictive model $\mathcal{H}$ is defined as $\mathcal{H}$ = f($\mathcal{N}, \mathcal{D}, \mathcal{S}$) where:
$\mathcal{N}$: Size of the Graph
$\mathcal{D}$: Graph Density
$\mathcal{S}$: *min_support*

Empirical results suggest that a logistic function best fits the observations. As shown in Figure 12 the logistic function of the form:

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

best fits the empirical results. Coefficients of the logistic function also exhibit a trend over $\mathcal{N}, \mathcal{D}, \mathcal{S}$. 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 attempt to find different functions to predict values of *a*,*b*, and *c*.

V. 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 [1] for the training. We will also improve upon existing sampling algorithms.

VI. 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] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 985–1042, 2010.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] S.-D. Lin, M.-Y. Yeh, and C.-T. Li, "Sampling and summarization for social networks."

[8] 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.

[9] 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.

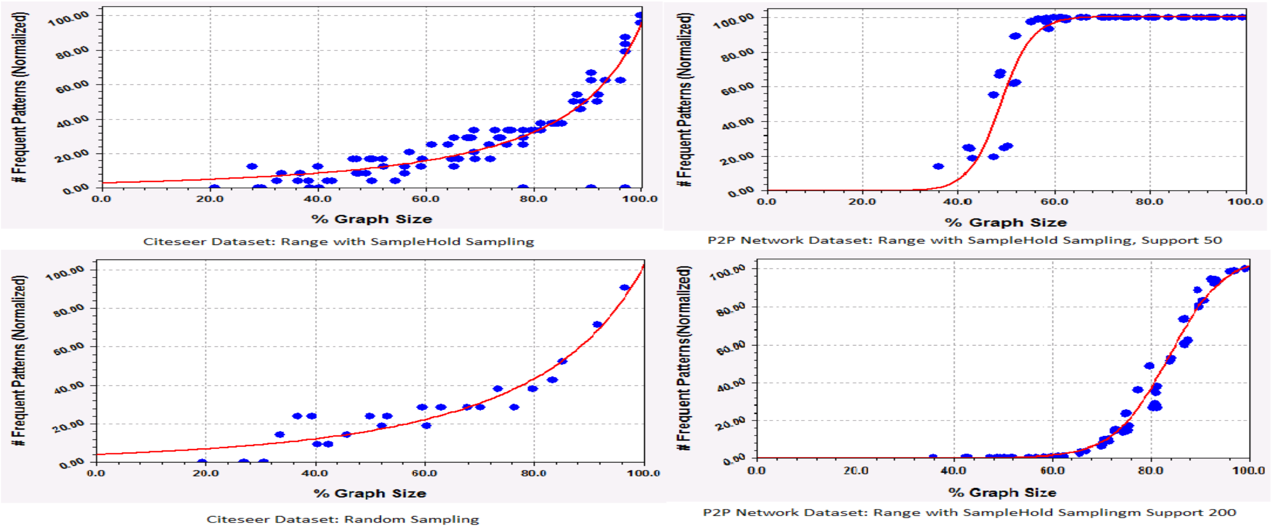
[10] N. K. Ahmed, J. Neville, and R. R. Kompella, "Network sampling designs for relational classification." in *ICWSM*, 2012.

[11] 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.

[12] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *The Knowledge Engineering Review*, vol. 28, no. 01, pp. 75–105, 2013.

[13] L. B. Holder, D. J. Cook, S. Djoko *et al.*, "Substucture discovery in the subdue system." in *KDD workshop*, 1994, pp. 169–180.

[14] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE, 2002, pp. 721–724.



Citeseer Dataset: Range with SampleHold Sampling

P2P Network Dataset: Range with SampleHold Sampling, Support 50

Citeseer Dataset: Random Sampling

P2P Network Dataset: Range with SampleHold Samplingm Support 200

Figure 12. Best fit Logistic Function

[15] 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.

[16] M. Kuramochi and G. Karypis, “Frequent subgraph discovery,” in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001, pp. 313–320.

[17] S. Nijssen and J. N. Kok, “The gaston tool for frequent subgraph mining,” *Electronic Notes in Theoretical Computer Science*, vol. 127, no. 1, pp. 77–87, 2005.

[18] C. H. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, “Arabesque: a system for distributed graph mining,” in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015, pp. 425–440.

[19] S. Choudhury, K. Agarwal, S. Purohit, B. Zhang, M. Pirrung, W. Smith, and M. Thomas, “Nous: Construction and querying of dynamic knowledge graphs,” *arXiv preprint arXiv:1606.02314*, 2016.

[20] B. Bringmann and S. Nijssen, “What is frequent in a single graph?” in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2008, pp. 858–863.

[21] S. Fortunato, “Community detection in graphs,” *Physics reports*, vol. 486, no. 3, pp. 75–174, 2010.

[22] H. Lu, M. Halappanavar, and A. Kalyanaraman, “Parallel heuristics for scalable community detection,” *Parallel Computing*, vol. 47, pp. 19–37, 2015.

[23] M. E. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, “Graphx: Graph processing in a distributed dataflow framework,” in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 599–613.

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

[31] D. J. Watts and S. H. Strogatz, “Collective dynamics of small-worldnetworks,” *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

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