A REFORMULATED APPROACH TO ATTRIBUTE-AWARE SAMPLING ON LARGE NETWORKS

BY

CHARLES SHANG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Associate Professor Hari Sundaram

ABSTRACT

Sampling has long been an important tool for extracting subsets of data for data mining tasks. As the scale of information produced has increased, efficient sampling is only becoming more important. Uniform sampling is often the preferred technique of choice, due to it's simplicity and speed. However, many network based data sources prevent random access, necessitating a different way to sample. Algorithms like Breadth first search, Random walk, Expansion sampling, or other related strategies fulfill this role currently. But these algorithms are focused mainly on ensuring properties based on the structure of the graph, without consideration for the attributes of each node.

In this study, we take an existing attribute aware sampler and propose a natural reformulation of the algorithm. We present a new surprise function that avoids some drawbacks of a previous work and take advantage of the submodularity property to reduce the computation that needs to be done when selecting a node and make some arguments about the efficiency and effectiveness of such a strategy. We test our algorithm on some real world data sets and found that our algorithm had increases in sample attribute coverage by up to 4 times when compared to techniques like random walk while still taking time approximately linear in the size of the sample.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Every day, a massive amount of data is being generated by the internet and individuals and organizations alike want access to this wealth of knowledge for various uses. One particular type of structure that can often be found on the internet is the network or graph. From social networks and hyperlinks on websites, to citations and papers, graphs are almost everywhere. Analysis on these data sets can be more generic, such as clustering[1], classification[2], or more network specific, like community[3] or outbreak[4] detection. Due to the sheer size of many networks (eg. Facebook has 2+ million nodes), processing all of the data is prohibitive, so suitable subsets need to be found.

The process of collecting these subsets, commonly referred to as sampling, is made complex by many different factors. The structure of the graph itself imposes constraints on which pieces of data can be accessed. Random access to arbitrary nodes, say people on a social network like Facebook, can be impossible on due to limits imposed by the source of the data. Facebook simply does not allow users access the data of random profiles. Many online services, like twitter, also rate limit API access, which can increase the time it takes to acquire each sample and thus reducing the number of samples it is possible to collect in a limited time.

Sampling on graphs is a well studied problem. Sampling on attributed graphs, however, is not. An attributed graph is a graph where each node has some additional data attached to it. In a social network, this can be information like age, height, or place of work. This is separate from graph structure data, which is things like degree or clustering coefficient. Getting a sample whose attributes are representative of the whole network is made difficult by a property called homophily. Homophily is a property commonly found in attributed graphs where nodes that are more similar are more likely to be connected to each other[5]. Thus, when using a simple graph traversal algorithm like BFS/DFS or random walk, samples are likely to be similar. In order to find a variety of attribute values, a sampler must find

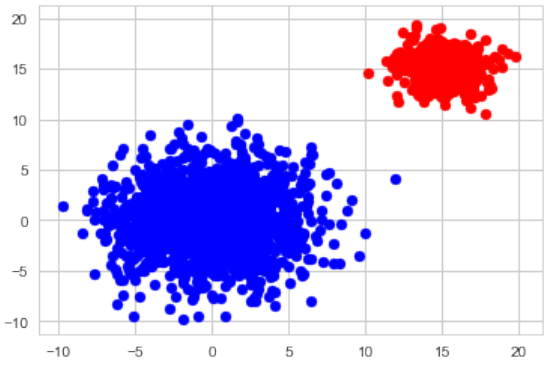
paths to potentially distant parts of the graph without having any information about how to get there.

Depending on the use of the sample, the data being representative may not be enough. For example, when training a SVM on attribute data, the most important data points would be the ones along the boundary of two classes, the support. A representative sample is likely to have many points in the densest parts of a distribution, which is often far from the boundaries. In cases like these, since the location of the boundary is unknown, coverage of the underlying attribute domain becomes more useful.

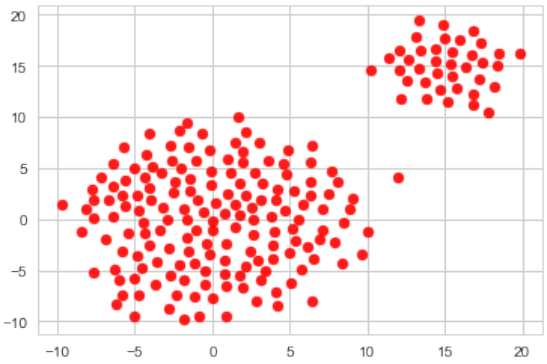
More formally, the problem we are trying to solve is as follows. Given a starting point on a graph and a budget of number of nodes, how to pick nodes to maximize coverage of an underlying distribution. Figure 1.1 is an example of a distribution and a few samples with different properties.

The focus of this work is to propose a sampler that uses attribute data of nodes to collect samples that give good coverage of those attributes while being efficient in terms of computation time and number of samples taken. Our sampler will be based on the idea of "surprise", or how dissimilar a candidate node is compared to our already sampled nodes. By picking surprising nodes to explore, we are hopefully able to quickly reach a wider range of attribute values. In addition, by picking a surprise function that satisfies submodularity, we are able to quickly identify which nodes should be explored even when the frontier is large. We also perform some analysis on the theoretical effectiveness of our algorithm.

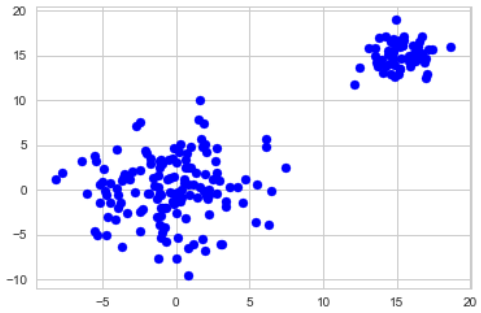
The rest of the thesis is organized as follows: In Chapter 2, related works are presented. In Chapter 3, a new attribute-aware graph sampling algorithm and analysis of its theoretical effectiveness are presented in details. In Chapter 4, experimental results using three real data sets are presented to show the improvement of the new method over previous methods. In Chapter 5, future work is discussed. Finally, in Chapter 6, the work is summarized and concluded.



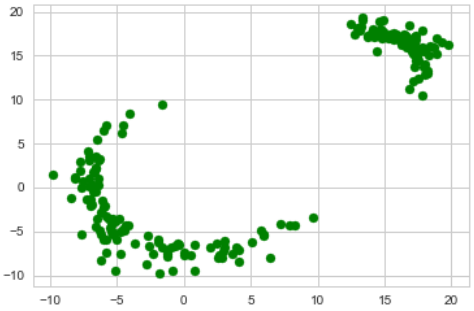
(a) A 2 center Gaussian distribution



(b) A sample with good coverage. This sample was produced by applying our algorithm assuming a complete graph.



(c) A uniform random sample



(d) An extreme point biased sample

Figure 1.1: An underlying distribution and some various samples

CHAPTER 2: RELATED WORKS

In this section, we discuss some general topics while building up to the problem of sampling on attributed graphs. The basis of our problem comes from one main direction. Attribute-independent samplers attempt to ensure some structural properties of the samples they give. Some examples of properties are degree distribution or probability of sampling a specific node. However, they are designed without consideration for the attributes of sampled nodes. We also discuss submodular optimization at the end as a way to improve the speed of our proposed sampler.

2.1 NOTATIONS AND DEFINITIONS

In this section, I will provide some common definitions used by this thesis. The network, or graph, $G =< V, E >$ is a set of vertices (or nodes) V and edges $E = \{(u, v) \mid u, v \in V\}$. The neighborhood of a vertex u is $N(u) = \{v \mid (u, v) \in E\}$ and the degree of u is $d(u) = |N(u)|$. The set of vertices that have already been sampled will be referred to as $S, S \subseteq V$. The frontier $F = \{v \mid (s, v) \in E, v \notin S, s \in S\}$ will be defined as the set of nodes neighboring the sample set that are not in the sample set. In addition to the above that describes standard graphs. Attributed networks are defined as a graph $G' =< V, E, X >$, where X is a set of $|V|$ W-dimensional feature vectors. Each feature vector x_i is paired with a corresponding vertices v_i and represents some characteristics of v_i. For a social network, this can be things like age or place of work.

2.2 ATTRIBUTE INDEPENDENT SAMPLING

For many purposes, acquiring a sample that is uniformly selected at random from an underlying distribution is a good starting point. However, this is made difficult by some networks preventing random access to arbitrary nodes. For example, to find a specific user

on Facebook, a name or other distinguishing feature is required to perform a search. Without knowing all possible attribute values, there will be users that are hard to find. As such, link-trace sampling is necessary. Link-trace sampling begins at a node and new samples are collected from neighboring nodes of existing samples, in the case of Facebook friend lists or group membership. This creates a frontier on nodes that slowly expands through the underlying network.

2.2.1 Basic Search

The basic forms of link-trace sampling are breadth/depth/random first sampling (BFS /DFS/RFS), names borrowed from their classic graph traversal counterparts. All three techniques start with some seed node as the initial sample and its neighbors as the frontier. While new samples are needed, a node is picked from the frontier its neighbors are added to the frontier. How this node is picked is what defines the differences between the three. BFS picks the first seen element, DFS picks the most recently seen element, and RFS picks one uniformly at random.

The sample sets created from these strategies are highly influenced by the structure of the graph. High degree nodes are more likely to be seen since arbitrary edges are more likely to be connected to nodes with more edges. It is then hard to give good relationships between sampled nodes and the total data set.

A randomized algorithm similar to BFS is Forest fire sampling (FFS)[6]. FFS is similar to BFS in that each time a new node is sampled, its neighboring unvisited nodes are considered. However, unlike BFS, not all neighbors are explored, instead each is visited with some "burning probability" p. In this way, FFS can be thought of as a fire that attempts to spread along edges. Similarly to the basic sampling algorithms, FFS suffers from being biased towards nodes of high degree due to each edge being considered separately.

2.2.2 Random Walk and its Variants

Random walk (RW) is another simple and commonly used technique. Instead of maintaining a frontier, a neighboring node is randomly chosen at each step to travel to. The probability assigned to each neighbor can vary depending on which of the many variants of RW is used. Of note, given enough time, RW will sample each edge with the same probability, though the same does not always hold for vertices.

A notable feature of random walk based samplers is the ability to revisit already visited nodes. If not accounted for, this causes most RW samplers to have a tendency to get trapped in local dense regions of a graph.

Some other modifications that are seen with RW include restarting the walk from the seed node after some amount of steps (Multiple independent random walkers), or jumping to a random other node in the graph with some probability (Random walk with escaping). Each of these modifications give slightly different structural properties to the nodes sampled.

One notable RW varient used when a representative sample is desired is the Metropolis-hastings random walk (MHRW)[7]. MHRW is a form of weighted random walk where after a candidate edge is picked with probability $1/d(u)$, whether or not it is taken is decided with probability

$$min\left(1, \frac{1/d(v)}{1/d(u)}\right)$$

Where u is the current node, v is the candidate node, and $d(x)$ is the degree of a node x.

As the length of the walk goes to infinity, the probability of visiting each node becomes the same. This means that given enough time, the samples collected by MHRW will become nearly the same as samples collected by a uniform random sampler. However, for most graphs where sampling is necessary (eg. Facebook with its 2+ billion users) the number of samples compared to the data set is small and MHRW will not reach the ideal of a uniformly random sample.

2.2.3 Other Algorithms

Expansion sampling (XS)[8] is an algorithm that purposefully tries to avoid being trapped in local communities by choosing a node v^* to sample in the following way:

$$v^* = argmax_{v \in S} |N(v) - S \cup N(S)|$$

In other words, XS will always pick nodes that give access to the most neighbors that are not shared by any nodes currently in the sample set. XS greedily adds nodes towards the largest unexplored region, which lets the sampler quickly escape local community structure.

2.3 ATTRIBUTE AWARE GRAPH SAMPLING

While there are very few general attribute aware graph samplers, one specialized form is the Snowball sampler. Snowball sampling is commonly used in the health community to perform surveys on very specific, historically hard to access, segments of the population (eg. drug users). This sampler asks survey participants for other people that they know that also have the specific population. The idea is that this sampler is able to quickly find hidden communities by asking to be referred to other members. For an automated system, this can be related to only exploring nodes that have a certain attributes or nodes that a classifier identifies as satisfying specific requirements. This doesn't translate well into attribute coverage however, due to desired samples often being distant attribute-wise.

The work my reformulations are based is done by Suhansanu[9], another student of Dr. Sundaram's. In his work, he proposes a task-independent algorithm called Surprising Information sampler (SI) that picks nodes based on the surprise of their neighborhood, or distance between a candidate's neighborhood and the existing sampled set. The surprise is defined by the following two functions, one for discrete attributes:

$$\text{Surprise}(v) = \sum_{A \in \mathbb{A}} \frac{d(p_{\Delta v}, p_{\mathbb{S}} \mid A)}{|\mathbb{A}|}. \tag{2.1}$$

$$d\left(p_{\Delta v}, p_{\mathbb{S}}\right) = \frac{-\sum_{i=1}^{r} p_{\Delta v}(i) \ln p_{\mathbb{S}}(i)}{|| \ln p_{\mathbb{S}}||}, \tag{2.2}$$

and one for continuous:

$$\text{Surprise}(v) = \min_{x \in \Delta v, y \in \mathbb{S}} d(x, y) \tag{2.3}$$

They concluded from their results that SI, though it did not preserve network structure very well, found samples that performed much better than attribute-agnostic samplers on data mining tasks like cluster preservation, with improvements averaging 45%.

They also discuss some limitations of their work, which we will be improving with our algorithms.

One limitation is the algorithms behavior with missing and new data. If a node lacks any attributes (eg. online social networks often have nodes with minimal attributes), they threw those nodes out of their data set. The equations could simply skip those attributes, but the behavior is not well investigated. In addition, the first time a certain discrete attribute value is seen, the surprise value for that node goes to infinity. While this simply causes a node to always be picked in the algorithm, it is an inelegant solution, and in situations where multiple infinite value nodes are found, tie-breaking is currently arbitrary.

Another limitation they address is that the algorithm as a whole can also be slow computation-wise. Each time a sample is added to S, the surprise of all frontier nodes have to be recalculated. Since the calculation for continuous variables involves comparing a frontier node's attributes to every single samples node, the number of computations grows quadratically with respect to the sample size and average degree. Compared to common random walk algorithms where the amount of work at each iteration is constant, this is very slow.

2.4 GRAPH MODELING

In order to reason about the effectiveness of our algorithm, we use synthetic graphs as a starting point. To keep things simple, we use the Erdos-Renyi model[10]. An Erdos-Renyi model graph $G(n, p)$ is a graph with n vertices where each edge exists with probability p (eg. $p = 1$ gives a complete graph). One important value of p is $\frac{\log n}{n}$, for all $p > \frac{\log n}{n}$, the probability of $G(n, p)$ being connect is basically guaranteed. This type of graph is easy to analyze due to the independence of edges.

2.5 SUBMODULARITY

One solution we investigate to speed up attribute aware samplers is the concept of submodularity.

In a work by Leskovec et al[4], they use submodularity to produce significant speedups in their outbreak detection algorithm. Our problem of attribute coverage is similar and uses similar techniques.

Outbreak detection, say in a water distribution network, is the problem of deciding where to put sensors in a network given a budget of k to best detect when an outbreak has occurred somewhere in the system. An outbreak is an event that occurs at some initial node and spreads along edges in the graph.

A submodular set function is a function $f : 2^V \to \mathbf{Z}$ that satisfies one of three equivalent conditions.

1. For every $X, Y \subseteq V$ with $X \subseteq Y$ and every $x \in V \backslash Y$ we have that $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$.

2. For every $S, T \subseteq V$ we have that $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$.

3. For every $X \subseteq V$ and $x_1, x_2 \in V \backslash X$ we have that $f(X \cup \{x_1\}) + f(X \cup \{x_2\}) \geq f(X \cup \{x_1, x_2\}) + f(X)$.

The equation most relevant to us is the first. In Leskovec's work, they formulate outbreak detection in networks as some submodular function $f(S)$ for set of nodes with sensors S and define a function $\delta(S, v) = f(S \cup \{v\}) - f(S)$ for some node v without a sensor. As $|S|$ increases, $\delta(S, v)$ can only decrease. They store the most recent value of $\delta(S, v)$ for each node v in the graph without a sensor. At each iteration, they mark all nodes without a sensor as invalid. Then, consider the node v with largest stored $\delta(S, v)$, if v is invalid, recalculate $\delta(S, v)$ with the most recent S and set v as valid. They continue to check the largest $\delta(S, v)$ until a node that is valid is found. In many cases, the nodes with highest $\delta(S, v)$ will only experience a small drop when recomputed, and are likely still at, or near, the top. This means that all the nodes with small marginal benefit don't need to be considered. With this optimization, they were able to reduce the number of computations by 700x.

CHAPTER 3: PROPOSED ATTRIBUTE-AWARE SAMPLER

3.1 PROBLEM FORMULATION

The problem we are trying to solve is as follows:

Given an attributed graph G and some set of arbitrary starting node(s) I, pick k samples to maximize familiarity of the underlying attribute distribution. Familiarity is the inverse of the surprise value for the most surprising unpicked node. Surprise is defined in the following sections. Again, this problem does not include any consideration for preserving any graph structural properties in the sample.

The number of possible solutions to this problem is $O(d^k)$, where d is the minimum degree of the graph. Since the search space is exponential in size, we employ a hill climbing using submodular maximization to produce an approximation.

3.2 ALGORITHM

The outline of the proposed algorithm is as follows in Algorithm 3.1.

Fast Surprise Sampler (FSS) selects the most surprising node in the frontier at each step using a priority queue. In fact $SurprisePrioQ$ is the frontier, just ordered in a specific manner. FSS reduces the number calls to $Surprise()$ that need to be made by using the property that the surprise value will never increase and thus only have to recompute the most surprising candidates.

Given a complete graph with attribute distribution seen in Figure 1.1a, an example of the nodes that FSS will attempt to pick, and their ordering can be seen in Figure 3.1. In the continuous case, FSS attempts to evenly cover the underlying distribution with samples and fill in gaps as it goes.

In the following sections, we discuss the process of choosing an good surprise function, as well as perform some analysis on the effectiveness and efficiency of FSS.

Algorithm 3.1 Algorithm outline

```
1: procedure FastSurpriseSampler(Graph G, Budget k)
2:     Sampled ← ∅
3:     seed = Random vertex v ∈ G
4:     SeenNodes ← {seed}
5:     SuprisePrioQ ← Empty max priority queue
6:     SuprisePrioQ.insert((0, seed))
7:     while |Sampled| < k do
8:         tested ← ∅
9:         while SurprisePrioQ.peek()[1] ∉ tested do
10:            update ← SurprisePrioQ.find-max()[1]
11:            tested.add(update)
12:            SurprisePrioQ.insert(Surprise(update), update)
13:        next_sample = find-max()[1]
14:        NewNeighbors ← Neighborhood(next_sample)/(Sampled ∪ SeenNodes)
15:        for n in NewNeighbors do
16:            SurprisePrioQ.insert((Surprise(n), n))
17:        Sampled ← Sampled ∪ {next_sample}
18:        SeenNodes ← SeenNodes ∪ NewNeighbors
19:    return Sampled
```

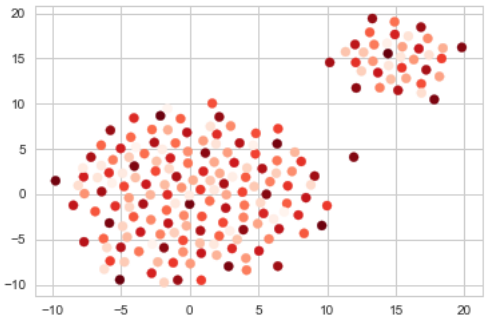


Figure 3.1: The order in which nodes a picked. Darker colored nodes are picked first.

3.2.1 Design of a surprise function

When designing our surprise function, we decided to interpret our sampled attributes and our candidates as distributions of their underlying attributes. This gives a natural definition of surprise as simply the difference between the distribution of our sample set and the distribution of our sampled set + candidate node.

In addition we need our surprise function to hold the property of being the difference between two submodular familiarity functions. This allows us to use a greedy hill climbing algorithm to approximate within a factor of $1 - 1/e$[see sampling paper] and to perform speedups in our algorithm.

$$\text{surprise}(S, v) = f(S \cup \{v\}) - f(S) \tag{3.1}$$

In more specific terms, our surprise function needs to satisfy equation 3.1 for some submodular function f. This equation means that the increase in utility gained by adding a node will only decrease as the size of the sample set increases. In other words, the surprise value of a node will only decrease over time. We refer to this as the submodular property in the rest of the thesis.

3.2.2 Interpretation of samples

We began by formulating a surprise function by measuring the distance between the attribute distribution of our sample set and a candidate node. How we do this depends on the type of attribute it is (i.e discrete vs continuous) and is described separately in the next two sections.

Discrete attributes

For the discrete case, we update the distributions in a Bayesian manner. Discrete attributes are well represented by multinomials. The conjugate prior of multinomial distri-

butions is the Dirichlet, which means we have an easy way of updating the underlying distribution of the sample set. By picking a good initial prior, we can avoid one problem that Suhansanu[9] faced by preventing the surprise function from diverging due to unseen attributes. The prior we used is a vector of one's with length equal to the number of attribute values. Let Θ_S be the distribution of set S.

To calculate distance between the sample set and the candidate, we use the negative of the probability of seeing a candidate c when drawn from S, defined by equation 3.2. α is the parameter vector of Θ_S with length k, β is the multivariate beta function.

$$\Pr[c|\Theta_S] = \int \cdots \int_{z_i \in Z; \sum_{i=1}^{k} z_i = 1} \prod_{i=1}^{k} z_i^{c_i} \Pr[Z] dZ \tag{3.2}$$

$$= \int \cdots \int_{z_i \in Z; \sum_{i=1}^{k} z_i = 1} \prod_{i=1}^{k} z_i^{c_i} \frac{\prod_{i=1}^{k} z_i^{\alpha_i - 1}}{\beta(\alpha)} dZ$$

Since our candidate is a single data point so each of it's attribute can only have one value, say c_h, we can simplify.

$$= \frac{1}{\beta(\alpha)} \int \cdots \int_{z_i \in Z; \sum_{i=1}^{k} z_i = 1} z_1^{\alpha_1 - 1} \cdots z_{h-1}^{\alpha_{h-1} - 1} z_h^{\alpha_h} z_{h+1}^{\alpha_{h+1} - 1} \cdots z_k^{\alpha_k - 1} dZ \tag{3.3}$$

By the definition of the β function, we collapse to

$$= \frac{\beta(\alpha_1, \cdots, \alpha_{h-1}, \alpha_h + 1, \alpha_{h+1}, \cdots, \alpha_k)}{\beta(\alpha)} \tag{3.4}$$

By using the relationship between β and Γ functions, we get.

$$= \frac{\Gamma(\alpha_1) \cdots \Gamma(\alpha_{h-1}) \Gamma(\alpha_h + 1) \Gamma(\alpha_{h+1}) \cdots \Gamma(\alpha_k)}{\Gamma(1 + \sum_{\alpha_i \in \alpha} \alpha_i)} \frac{1}{\beta(\alpha)} \tag{3.5}$$

Now we make the observation that in each iteration, when looking for the most surprising (or dissimilar) candidate node, all candidates share the same α. Also from the definition of Γ, we have $\Gamma(\alpha_h + 1) = \alpha_h \Gamma(\alpha_h)$ Thus we can drop the denominator and are left with

$$\Pr[c|\Theta_S] \propto \Gamma(\alpha_1)\cdots\Gamma(\alpha_{h-1})\Gamma(\alpha_h+1)\Gamma(\alpha_{h+1})\cdots\Gamma(\alpha_k) \tag{3.6}$$

$$= \alpha_h \prod_{i=1}^{k} \Gamma(\alpha_i)$$

Once again, this is proportional to

$$\Pr[c|\Theta_S] \propto \alpha_h \tag{3.7}$$

The surprise between a candidate with attribute value c_h is proportional to -1 times the number of times that attribute appears in the sample set. To keep surprise values small, we define

$$surprise_{discrete}(c, S) = -\ln(\alpha_i); c_i = 1 \tag{3.8}$$

This satisfies the submodularity property we need because α_i can only increase as nodes are added to S and ln is monotone for all positive values.

An extension of this surprise function for discrete attributes is instead of considering a single candidate c, consider the set $\Delta d = \{c\} \cup N(c)$. This would evaluate the surprise of a neighborhood of a candidate instead of just the candidate itself. Define Δd_i as the number of occurrences of attribute value i. Equation 3.4 becomes

$$= \frac{\beta(\alpha_1 + \Delta d_1, \cdots, \alpha_k + \Delta d_k)}{\beta(\alpha)} \tag{3.9}$$

Again using the relationship between β and Γ functions, we get

$$= \frac{\Gamma(\alpha_1 + \Delta d_1)\cdots\Gamma(\alpha_k + \Delta d_k)}{\Gamma(|\Delta d| + \sum_{\alpha_i \in \alpha} \alpha_i)} \frac{1}{\beta(\alpha)} \tag{3.10}$$

From here, the analysis begins to look different, since the denominator can be different for different sized neighborhoods. Let $|\alpha| = \sum_{\alpha_i \in \alpha} \alpha_i$

$$\propto \frac{(\alpha_1 + \Delta d_1 - 1)! \cdots (\alpha_k + \Delta d_k - 1)!}{(|\Delta d| + |\alpha| - 1)!} \tag{3.11}$$

Taking the log of the above and applying Stirling's approximation $\ln n! = n \ln n - n$ results in

$$\approx \frac{\sum_{i=1}^{k} (\alpha_i + \Delta d_i - 1) \log(\alpha_i + \Delta d_i - 1) + (\alpha_i + \Delta d_i - 1)}{(|\Delta d| + |\alpha| - 1) \log(|\Delta d| + |\alpha| - 1) + (|\Delta d| + |\alpha| - 1)} \tag{3.12}$$

$$= \frac{|\Delta d| + |\alpha| - 1 + \sum_{i=1}^{k} (\alpha_i + \Delta d_i - 1) \log(\alpha_i + \Delta d_i - 1)}{(|\Delta d| + |\alpha| - 1) \log(|\Delta d| + |\alpha| - 1) + (|\Delta d| + |\alpha| - 1)} \tag{3.13}$$

From here, the equation is possible to calculate quickly, it however is still much more complex than the candidate only (no neighbors) case.

Continuous attributes

Initially, we treated each continuous attribute as pulled from a Gaussian distribution. This didn't work as the sampler tended to prefer extreme samples, instead of samples that provide good coverage.

Instead, we decided to treat each sample in our sample set as the mean of a multivariate Gaussian, creating a mixture model $P(S)$ for sample S seen in equation 3.14. The variance of each multivariate Gaussian was set to $\frac{1}{|S|} I_{|A|}$, where $I_{|A|}$ is the identity matrix with size equal to the number of attributes, since as the number of samples grew, we expect the influence of each existing sample to fall.

$$P(S) = \sum_{s \in S} \frac{1}{|S|} \mathcal{N}\left(s, \frac{1}{|S|} I_{|A|}\right) \tag{3.14}$$

Where s denotes an attribute value in sample set S and $\mathcal{N}$ is the normal distribution.

A common way to measure the difference between two distributions P, Q, or information gained from using distribution P instead of Q, is the KL divergence, defined as follows:

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) \, dx \tag{3.15}$$

One problem, however, is calculating the KL divergence between two mixture models is computationally expensive[11]. Hershey and Olsen [11] have proposed several approximations, but the good ones still have a $O(|S|^2)$ running time. For our purposes this is still too computationally expensive.

Instead, we came up with an approximation specific for our purposes. The goal of using the KL divergence is to find the single sample that provides the largest value of D_{KL}. In one iteration of our algorithm, we compare many candidate nodes to the same sample set. This means that in equation 3.15, given any two samples, they will integrate to nearly the same value for all values of x except for the area around where the candidate means are. Thus, the biggest contribution to the difference between the KL divergences of two candidate nodes is from the Gaussian centered on the mean of the candidate and it's closest neighboring sample in the sample set. Thus we get the following equation:

$$\text{approx.}D_{\mathrm{KL}}(P(S \cup \{c\}) \parallel P(S)) = D_{\mathrm{KL}}(\mathcal{N}(c, \frac{1}{|S|+1} I_{|A|}) \parallel \mathcal{N}(\arg\min_{s \in S} d(c,s), \frac{1}{|S|} I_{|A|})) \tag{3.16}$$

Where the right hand side is

$$\begin{gathered} D_{\mathrm{KL}}(\mathcal{N}(c, \frac{1}{|S|+1} I_{|A|}) \parallel \mathcal{N}(\arg\min_{s \in S} d(c,s), \frac{1}{|S|} I_{|A|})) = \\ \frac{1}{2} \left(\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)^{\mathsf{T}} \Sigma_1^{-1} (\mu_1 - \mu_0) - |A| + \ln \left(\frac{\det \Sigma_1}{\det \Sigma_0} \right) \right) \\ \mu_0 = c, \Sigma_0 = \frac{1}{|S|+1} I_{|A|}, \mu_1 = \arg\min_{s \in S} d(c,s), \Sigma_1 = \frac{1}{|S|} I_{|A|} \end{gathered} \tag{3.17}$$

To further simplify, we notice that for all candidates in one iteration Σ_1, Σ_0 are constant, so by replacing them with the identity matrix, we can reduce the amount of computation

without affecting the results. To confirm this simplification does not affect nodes chosen, we tested on synthetic and real data sets.

Now, notice that within one iteration, the only difference between candidates is c, meaning the whole term is proportional to $(\mu_1 - \mu_0)^\mathsf{T} I_{|A|}(\mu_1 - \mu_0)$, which is just the square distance between the two points. Thus we can say

$$\text{approx.} D_{\mathrm{KL}}(P(S \cup \{c\}) \parallel P(S)) \propto \min_{s \in S} d(s, c)^2 \tag{3.18}$$

To make things simpler, we define surprise as just the distance, which is actually the same conclusion reached by Suhansanu in [9], though by a different processes.

$$surprise_{continuous}(c, S) = \min_{s \in S} d(s, c) \tag{3.19}$$

This satisfies the submodularity property since points can only be added to the sample set, the closest distance between a candidate and a sample can only decrease or stay the same.

3.2.3 Surprise function

We combine our surprise functions for the discrete and continuous case in the following way. For continuous attributes, they are interpreted as a single multidimensional vector for calculating the distance for surprise. The surprise of each discrete variable is calculated individually. All of these surprise values are summed and then divided by a normalizing factor equal to the number of attributes for the final surprise value of a candidate node. Missing attribute values do not contribute to the surprise and also do not contribute to the normalizing factor.

3.3 ANALYSIS

3.3.1 Complete vs Network constrained

Using a greedy hill climbing method on a submodular optimization problem can be a factor of $1 - 1/e$ off optimal[12], but this is assuming access to the full data set. In our problem we are constrained by a graph, which can make our results arbitrarily bad (consider a line graph where all nodes around a starting point are identical). However, in practice, this strategy works well.

As a proof of concept, we created an Erdos-Renyi model graph $G(n, p), n = 2000, p = \frac{\log n}{n}$, where p is chosen to ensure connectedness. 1500 nodes had attributes pulled from $\mathcal{N}((0,0), \left[\begin{smallmatrix} 10 & 0 \\ 0 & 10 \end{smallmatrix}\right])$ and the other 500 from $\mathcal{N}((15,15), \left[\begin{smallmatrix} 2 & 0 \\ 0 & 2 \end{smallmatrix}\right])$. When plotted, the underlying distribution looks like Figure 1.1a.

We then ran an experiment as follows. Take a graph and pick 400 samples starting from a random node. After each 20 samples, record the maximum distance an unpicked node was from the closest sampled node. This would be the quality of our sample, with smaller values being better.

$$\text{Sample Quality } = \max_{v \in V/S} \min_{s \in S} d(v, s) \tag{3.20}$$

We ran the experiment once assuming a complete graph and then 10 times according to the edges of the generated Erdos-Renyi model. We average the quality scores of the 20 trials and the results are in Figure 3.2

As you can see, limited node access to neighboring nodes does impact the quality, preventing quality of the graph constrained sampler from reaching the quality of the sampler on the complete graph. This is partially due to the constrained sampler being forced to pick non-optimal nodes, which creates many gaps that are uneven and larger than desired. Filling in those gaps is then difficult as we know no existing samples have edges to nodes in

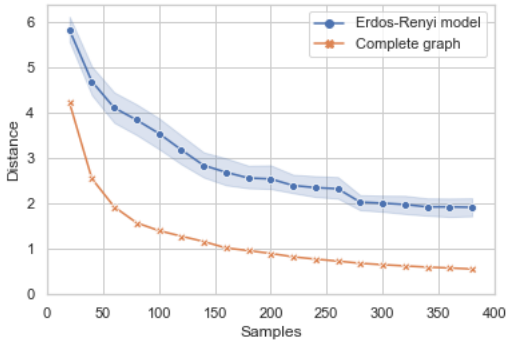
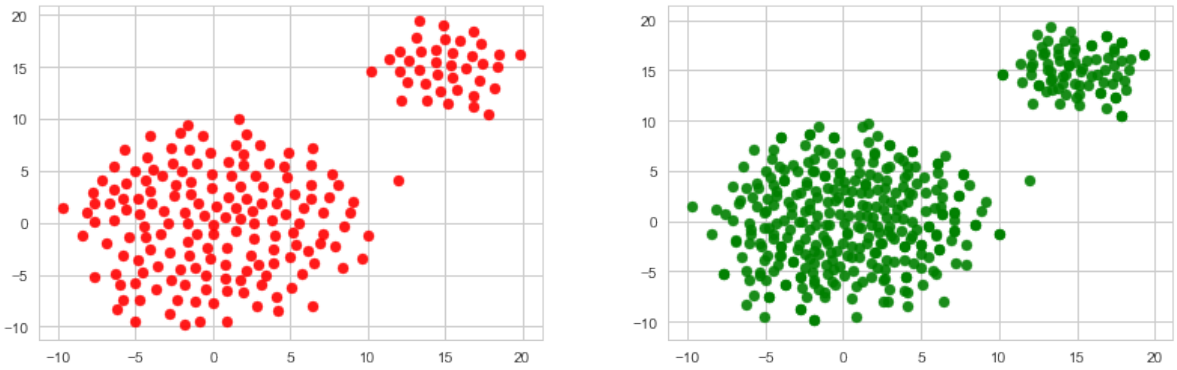


Figure 3.2: Distance of the furthest node from the sample over 400 samples. While the distance doesn't drop nearly as quickly when the graph structure is enforced, it drops at a comparable pace.

those gaps, so we have to rely on new nodes and their neighbors. Figure 3.3 is two graphs comparing the 400 samples taken during a trial.



(a) On a complete graph

(b) On an Erdos-Renyi model

Figure 3.3: The first 200 samples chosen. The underlying distribution can be found in Figure 1.1(a). Enforcing a graph constraints results a less even distribution of samples, but not by too much.

From appearances alone, the attribute distribution of the samples from the edge constrained graph are slightly closer to the underlying distribution. This is due to the fact that there are more nodes with those values and thus the sample set is more likely to have an edge to those nodes.

Of course, this model is not fully representative of real work attributed networks. It does not factor in homophily or features like communities. But those things should only benefit

our algorithm and improve sample quality.

3.3.2 Running Time

Having to calculate the surprise value of every frontier node for every sample taken is time consuming, which is why we used the submodular property of surprise and a priority queue to speed things up. But employing such a data structure also has costs. How much improvement does the priority queue really give us?

To find each sample, FSS will potentially iterate over all elements of the priority queue to find the next sample. If there are n nodes on the frontier, this results in a worst case $O(n \log n)$ time per iteration compared to $O(n)$ time per iteration to just recalculate surprise values. Assuming most nodes have similar degree, the running time become $O(n^2 \log n)$ and $O(n^2)$ respectively.

In practice, however, this is very unlikely, as the surprise value of nodes do not decrease significantly when distant node (attribute-wise) are picked. And as the space of the sampled attributes increases, the vast majority of frontier nodes are distant from any particular candidate. So as the sample size increases, fewer and fewer nodes will have significant updates, so FSS gets faster.

To test these claims, we performed an experiment to test exactly how much speed up we get. The experiment is as follows. Take a graph and pick k samples starting from a random node. Do this once with a priority queue and once recalculating all surprise values. After each 10% of k nodes were sampled, record the time elapsed since the start of the run. We performed this experiment three times, once on a complete generated graph, once on a generated Erdos-Renyi model ($G(n, p), n = 2000, p = \frac{\log n}{n}$), and once on a real world patent data set (described in more detail in section 4.1). Both generated graphs have 2000 nodes and $k = 400$. The patent graph has 92106 nodes and $k = 4000$. The results are in Figure 3.4

From the graphs, the running times of simply recalculating surprise at each iteration does

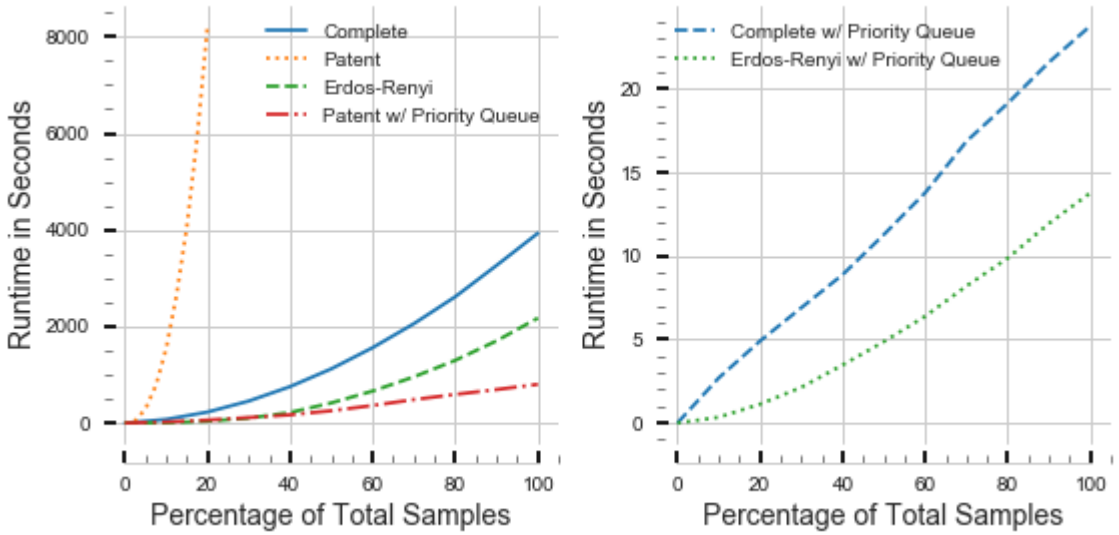


Figure 3.4: The number of seconds to collect various samples. The x axis is percentage of the sample taken. Note the difference in y axis scale of the two figures. When the priority queue is used, the running time is nearly linear.

seem to be $O(n^k)$ for $k \geq 2$. When using a priority queue, we end up with much shorter times, in fact, the time is almost linear. For the synthetic graphs, the time is 100 times shorter, while for the patent graph, the speed up is much larger.

CHAPTER 4: EXPERIMENTATION

4.1 DATASETS

To test our algorithm, we used three different data sets. In addition to the descriptions below, Table 4.1 goes into more detail about each attribute.

The first data set is collected from Facebook[13] and consists of anonymized user profiles as nodes and friendships as undirected edges. The original data set had 20+ different continuous and discrete attributes. However, due to most nodes missing values for most attributes, we only considered 3 discrete attributes: gender, local, and education type. We chose these attributes because they were the most complete.

The second data set is an Enron email network[14]. Nodes are individual users, attributes are various statistics about their email habits. Edges exist between two users if they have ever emailed each other. All attributes in this data set were continuous, so we selected a subset that we felt could be interesting.

The third and final data set is a citation network of patents in the US from 1963 to 1980[15]. Nodes are patents and edges represent citations. This data set had a good mix of continuous and discrete variables, so we selected several attributes that were relatively diverse.

4.2 METRICS

To evaluate the effectiveness of a sampler, I define three metrics. They are as follows.

4.2.1 Furthest unsampled node

For data sets with continuous attributes, this is the largest euclidian distance between a node in the data set and the nearest node in the sampled set. Another interpretation of this value is the minimum distance k such that all nodes are within k of a sample. The smaller

Attribute	Type	Cardinality
Facebook	4039 nodes	88234 edges
Gender	Discrete	3
Education Type	Discrete	2
Locale	Discrete	10
Enron	13533 nodes	176987 edges
AvgContentReplyCount	Continuous	[0, 19]
AvgNumberTo	Continuous	[0, 795]
AvgContentForwardingCount	Continuous	[0, 5]
AvgNumberCc	Continuous	[0, 457]
Patent	92106 nodes	125669 edges
Category	Discrete	36
Country	Discrete	69
Assignee Type	Discrete	7
Originality	Continuous	[0, 46]
% after 1963 citations made	Continuous	[0, 1]
Citations Made	Continuous	[0, 411]
Claims	Continuous	[0, 457]

Table 4.1: Attribute statistics for the three networks

this number, the more familiarity the sample has with the underlying distribution. This is a desirable feature in samples as it means no data point is too different from a sampled node.

4.2.2 Coverage

For discrete attributes, coverage is the count of how many unique tuples are sampled out of the total number of unique tuples. If a data set only has discrete attributes, full coverage would mean every single combination of attribute values was sampled. The higher this number, the more familiarity the sample has with the underlying distribution. Higher coverage is desirable since it means a data point is more likely to have a similar, or identical, counterpart in the sample set.

It is possible to include continuous variables in coverage by first binning them into and treating each bin as a discrete value.

4.2.3 Observed Distribution

This is less of a quantitative value, and more of a qualitative description. FSS was designed to take samples with an even mixture of attribute values, which naturally means the attributes will have a different distribution when compared to the ground truth. Thus, visualizations of the distributions of the whole data set and the sample are useful to better understand what is going on.

4.3 RESULTS

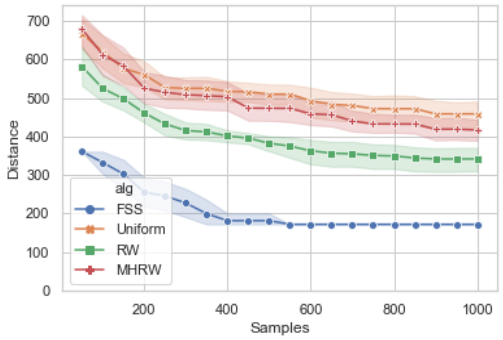
To evaluate the effectiveness of FSS, we compared it to a uniform, random walk, and metropolis-hastings random walk sampler. For each dataset, sampler pair, we start from a random seed node and sample 1000 points. The three link trace samplers (FSS, RW, MHRW) were limited by the graph structures while the uniform sampler had random access to the whole dataset. This process was repeated 20 times and the results averaged.

4.3.1 Furthest unsampled node

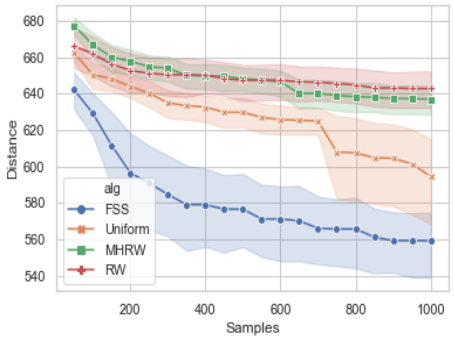
Since only the Enron and Patent data sets have continuous variables, Figure 4.1 only has two graphs.

For both of these data sets, FSS performs the best. This is expected, as FSS will purposefully pick the candidate node on it's frontier that is the furthest away. The uniform sampler especially struggles on the Enron data set due to the existence of large outliers, which can be seen in Figure 4.4. Another interesting quirk of the Enron results is variance between runs of FSS dropping to zero. The likely reason for this is the existence of a distant (attribute-wise) node that has very few edges to unsurprising nodes. Thus the sampler is very unlikely to ever come across that single surprising node.

In the patent data set, the sudden drops are due to a few scattered outlier points. These outliers generally have a very large number of claims. They are rare, 99.7% of data points



(a) Enron



(b) Patent

Figure 4.1: The average distance of the furthest unsampled node. Note the y axis scale of Subfigure (b). FSS performs better on both data sets. On Enron, it does nearly twice as good, while on Patent, it only does about 10% better. This can be partially attributed to how large the Patent data set is. The variance of the FSS sample for Enron drops to zero, possibly due to a distant node that has few edges.

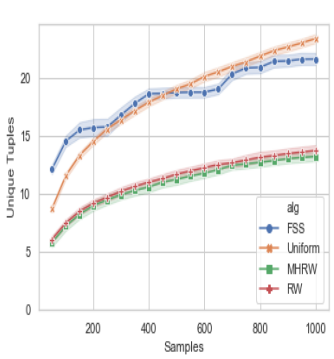
have < 40 claims and only 9 have > 80. With a total of more that 90,000 nodes, whether or not a sampler finds one of these points is mostly dependent on starting seed. In fact, these outliers cause the difference between FSS and MHRW on the patent data set to be only around 13% ($\sim$640 vs $\sim$560).

4.3.2 Coverage

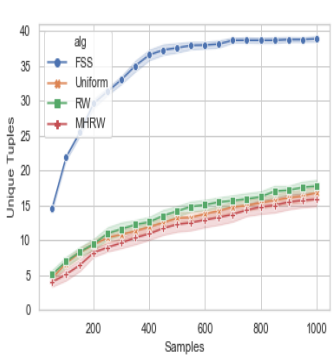
Since both discrete and continuous attributes can be compared using coverage, all three data sets have figures in Figure 4.2. The continous variables of the Enron dataset were binned into 10 bins, each representing 10% of the range of attribute values.

Uniform and FSS both show good results for the Facebook and Patent data sets. The number of unique tuples for Patent may seem small compared to the total unique tuples, but this is due to the sample size being only 1000. For the Enron dataset, again, Uniform struggles due to how skewed the attributes are.

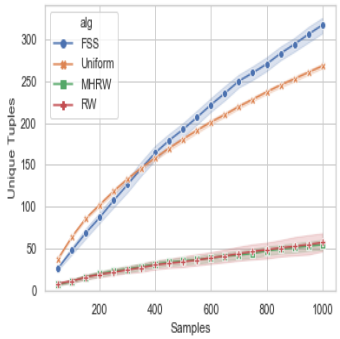
FSS has a very bumpy line for the Facebook data set. I believe this is due to different trials finding a communities of unique nodes at different times in the run, increasing the number of new tuples quickly before having to search for new communities.



(a) Facebook (35)



(b) Enron, Binned (47)



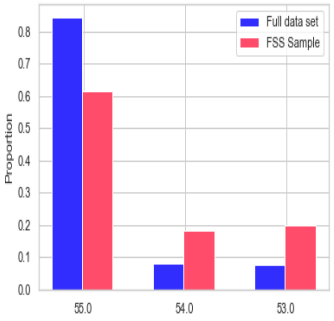
(c) Patent, discrete only (1958)

Figure 4.2: Average number of unique tuples sampled. Total unique tuples are noted in the parenthesis. Uniform and FSS perform comparatively on Facebook and Patent. Uniform struggles with Enron due to high skew in the attributes. RW and MHRW both do not do very well.

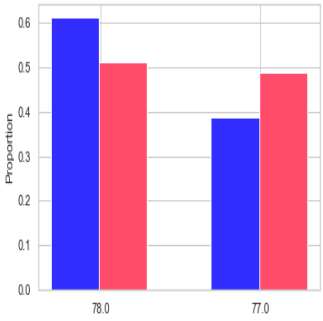
RW and MHRW both struggle here, most likely because they get caught in local structures and are not able to cover enough of the graph to reach new tuples.

4.3.3 Distribution

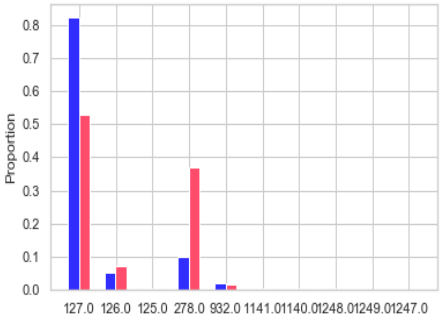
While the distribution is a qualitative judgement of how well a sampler is functioning, it does reveal interesting insights. Each data set has been given a separate figure.



(a) Education Type



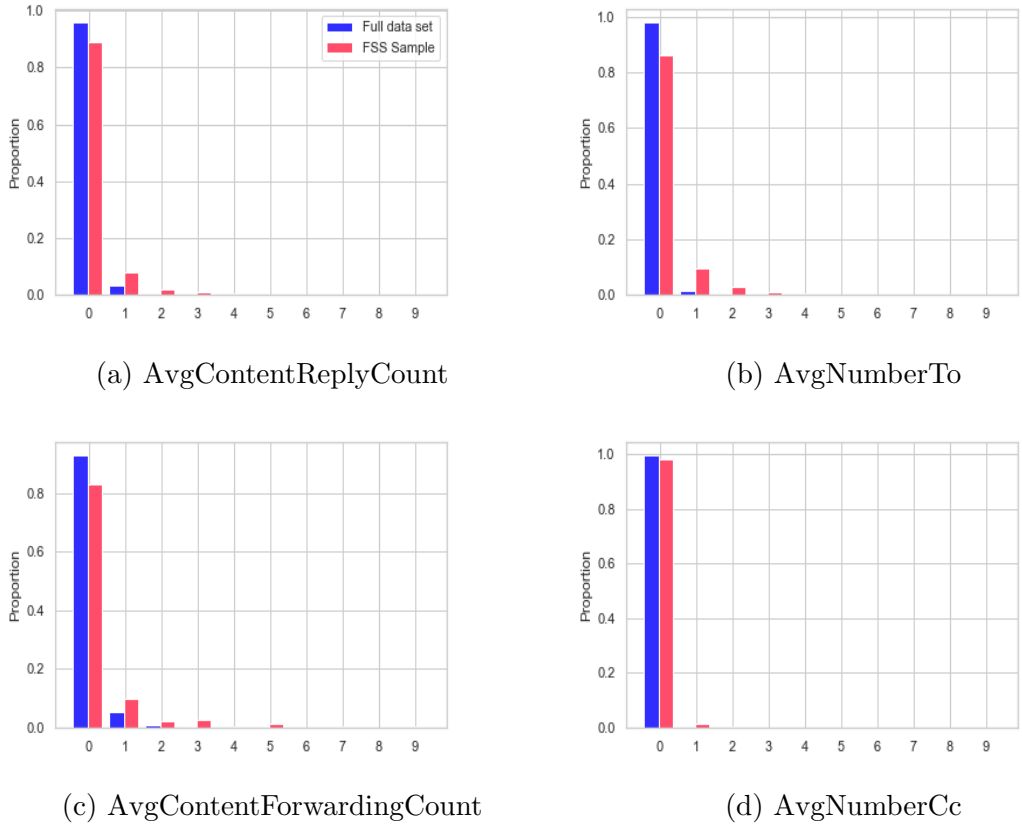
(b) Gender



(c) Locale

Figure 4.3: Distribution of attributes from the Facebook data set. FSS consistently produces closer to uniform distributions.

For the Facebook data set (Figure 4.3) the FSS samples consistently have a closer to uniform distribution compared to the full data set. Very common values in the full data set are a little more rare in the sample, and rare values in the full data set are a little more common in the sample. From the locale figure, we can see that locale 278 has a significantly higher proportion in the sample when compared to the full data set. I believe this is due to 278 being common enough that FSS would generally pick those nodes instead of the very common locale 127.



(a) AvgContentReplyCount

(b) AvgNumberTo

(c) AvgContentForwardingCount

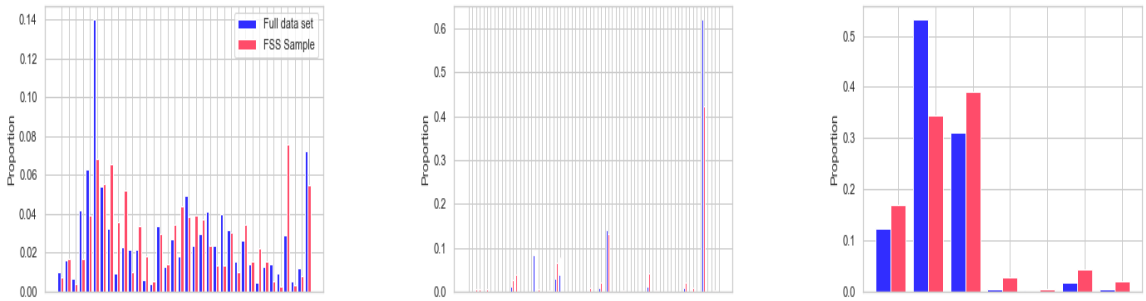
(d) AvgNumberCc

Figure 4.4: Distribution of binned attributes from the enron data set. Again, FSS consistently produces samples that are closer to uniform. Notice how most of the data set have small attribute values, this causes the Uniform sampler to struggle with coverage.

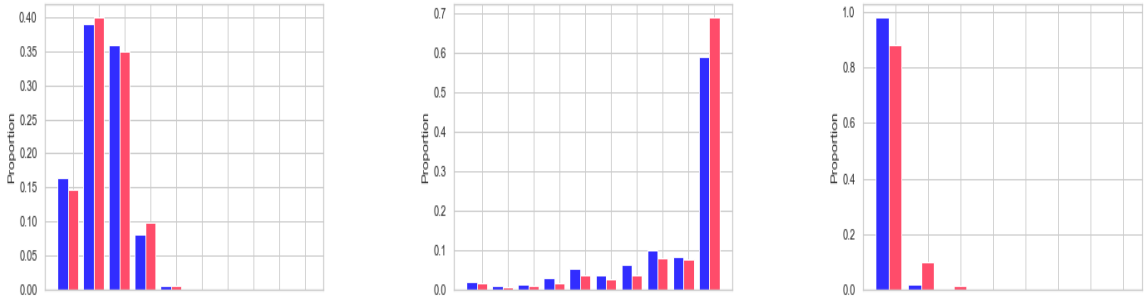
The tendency for FSS to favor rare values is more obvious in Figure 4.4. For all four of these attributes, FSS has flattened out the distribution slightly. For some bins, the full data set bar is so small it wasn't even drawn, while the FSS bin is still visible. These figures

also make more obvious the skewed nature of this data set. Most attribute values are less than 10% of the full range. This shows that FSS can function well even in these extreme environments.

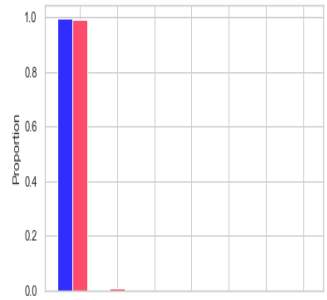
The results from the patent data are more mixed. Specifically, Figure 4.5 (e) has the full data set being closer to uniform. I believe this is due to the range of that attribute being $[0, 1]$. This range is small, which causes FSS to weight that attribute much less that the other attributes, since it is not doing any weighting. The difference in the calculations for continuous and discrete values could also be causing some of the problems. These things would probably the biggest weakness of FSS.



(a) Category (b) Country (c) Assignee type



(d) Originality (e) % after 1963 citations made (f) Citations made



(g) Claims

Figure 4.5: Distribution of attributes from the patent data set. Continuous variables are binned. Due to the number of attribute values, labels have been omitted. The FSS samples in (e) are further from the uniform distribution when compared to the base data set. This is caused by attribute (e) having small range, causing FSS to be biased against it.

CHAPTER 5: FUTURE WORK

Sampling on attributed graphs is still relatively unexplored and thus there are still many directions where work can still be done.

In our work, we focused on one strategy of designing a surprise function. However, as long as it satisfies the submodular property, there could be other ways to go about it. Our surprise function for continuous variables is dependent on the range of the underlying attribute, having some way to weight attributes to keep one from dominating the others could be useful to get more evenly spread samples. Another possible direction would be to combine knowledge about structural and non-structural attributes to give to the sampler to more rapidly find surprising data points.

One assumption we make in FSS is the independence of attributes. It is possible that there are better strategies to sampling that can take advantage of two attributes being strongly correlated. If two attributes are strongly correlated, it is possible that data points that don't follow the correlation could be more valuable. Also, we decided to evaluate surprise for discrete and continuous attributes in two separate ways. Having one unified surprise function for the two type would be a more elegant solution and would allow the sampler to possibly take advantage of any relationships between those variables. Another drawback of FSS is the effect of the difference in magnitude of different attributes on which attributes are more heavily weighted. A way to normalize without knowing the full range could solve this problem.

Another problem found in real world data sets is the reliability of node attributes. Alternatively, some attributes could be set by a classifier that doesn't have full accuracy. How does this error interact with the sampling process? If different attributes have different inaccuracies, how should a sampler take this into account? Intuitively, the sampler would just need more samples to reach the same level of use-fullness, but how many more, or which samples are better, could be an interesting study. It might also be possible for a sampler to

identify samples that are more likely to be correct and use that to it's advantage.

This work focused on collecting better samples for an arbitrary task to save time, but another way to increase sampling efficiency would be to investigate how many samples are needed for various tasks. Given then complexity of a task (eg. clustering, classification), say by VC dimension, is it possible to give a relationship between the number of samples taken by a specific sampler and how well the task is completed.

CHAPTER 6: CONCLUSION

In this work, we study sampling on attributed networks.

After briefly going over attribute agnostic sampling methods, we improve an existing surprise based sampling algorithm by reformulating the surprise function so it no longer diverges on unseen attribute and using submodularity to avoid recalculating the surprise of all frontier nodes. We argued that even though the graph structure defined by the problem could result to arbitrarily bad results, in practice, this is rarely the case.

From our results, we can see that our algorithm generally produces samples that are biased towards more uniform attribute distributions, and that it covers the underlying data set better than random walk, both the standard version and metropolis hastings. On the Patent data set, we saw over 4 times more coverage from FSS. For a highly skewed data set, it also outperforms uniform sampling in terms of coverage. The results also showed that our algorithm still has draw backs and there is still much work that can be done in this area. As this research direction becomes more developed, hopefully it will lead to newer and better ways of collected data, especially as advances in data mining and machine learning continue to drive the need for good data.

REFERENCES

[1] M. S. Handcock, A. E. Raftery, and J. M. Tantrum, "Model-based clustering for social networks," *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, vol. 170, no. 2, pp. 301–354, 2007.

[2] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*. Springer, 2011, pp. 115–148.

[3] J. Yang, J. McAuley, and J. Leskovec, "Community detection in networks with node attributes," in *2013 IEEE 13th International Conference on Data Mining*. IEEE, 2013, pp. 1151–1156.

[4] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281239 pp. 420–429.

[5] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[6] 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.

[7] C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani, "Metropolis algorithms for representative subgraph sampling," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 283–292.

[8] A. S. Maiya and T. Y. Berger-Wolf, "Benefits of bias: Towards better characterization of network sampling," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 105–113.

[9] S. Kumar and H. Sundaram, "Task-driven sampling of attributed networks," in *Paper in preparation*, 2018.

[10] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[11] J. R. Hershey and P. A. Olsen, "Approximating the kullback leibler divergence between gaussian mixture models," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, vol. 4, April 2007, pp. IV–317–IV–320.

[12] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functionsi," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[13] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in neural information processing systems*, 2012, pp. 539–547.

[14] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[15] B. H. Hall, A. B. Jaffe, and M. Trajtenberg, "The nber patent citation data file: Lessons, insights and methodological tools," National Bureau of Economic Research, Tech. Rep., 2001.