Efficient Sampling of Information in Social Networks

Gautam Das
University of Texas at Arlington
gdas@uta.edu

Nick Koudas
University of Toronto
koudas@cs.toronto.edu

Manos Papagelis
University of Toronto
papaggel@cs.toronto.edu

Sushruth Puttaswamy
University of Texas at Arlington
sushruthp@gmail.com

ABSTRACT

As online social networking emerges, there has been increased interest to utilize the underlying social structure as well as the available social information to improve search. In this paper, we focus on improving the performance of information collection from the neighborhood of a user in a dynamic social network. To this end, we introduce sampling based algorithms to quickly approximate quantities of interest from the vicinity of a user's social graph. We then introduce and analyze variants of this basic scheme exploring correlations across our samples. Models of centralized and distributed social networks are considered. We show that our algorithms can be utilized to rank items in the neighborhood of a user, assuming that information for each user in the network is available. Using real and synthetic data sets, we validate the results of our analysis and demonstrate the efficiency of our algorithms in approximating quantities of interest. The methods we describe are general and can probably be easily adopted in a variety of strategies aiming to efficiently collect information from a social graph.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Information networks*; H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Performance, Human Factors

Keywords

Sampling Social Networks, Social Search, Personalization

1. INTRODUCTION

The widespread adoption of Web2.0 is evident by the multitude of activity related to social networks and web collaboration. Web sites like Myspace, Facebook, LinkedIn to name a few attract millions of users that interact, share and collaborate. At the same time the desire to connect and interact evolves far beyond centralized social networking sites and takes the form of ad hoc social networks formed by instant messaging clients, VoIP software, etc; although interactions with people beyond one's contact list is not currently possible (e.g., via query capabilities), the implicit social networking structure is in place.

Given the large adoption of these networks, there has been increased interest to explore the underlying social structure and data towards social search. The main idea of social search is to use information collected from a user's social network to improve the accuracy of search results. Social search has recently gained attention as an approach towards personalized search. The utility of social search has been established via experimental user studies [16]. However, for large online social networks, usually consisting of millions of users, a complete crawl of a user's extended neighborhood is infeasible. Therefore, efficient methods are required.

We build on these observations and focus on improving the performance of information collection from the neighborhood of a user in a social network. We model environments in which social peers participate in a centralized social network (where knowledge of the network structure is assumed) or distributed (where network structure is unknown or limited). The rate of change of the structure of these networks is also an important factor. We make cases for static and dynamic networks. In each case we assume access to user log data (e.g., clickthrough logs, as is frequently the case for research related to search personalization) that are frequently updated.

Given such an environment we make the following contributions:

- We introduce sampling based algorithms for quickly approximating the number of users in the neighborhood of v that have actually endorsed an item.
- We introduce and analyze variants of this basic sampling scheme in which we aim to minimize the total number of nodes in the network visited by exploring correlations across samples.
- We evaluate our sampling based algorithms in terms of accuracy and efficiency using real and synthetic data and demonstrate the utility of our approach.
- We show that our sampling schemes can be utilized for a variety of strategies aiming to rank items in a network, assuming that information for each user in the network is available.

The rest of the paper is organized as follows. In Section 2 we review related work. Section 3 formally defines the problem and introduces appropriate notation. The foundational ideas of our methodology are presented in Section 4 and Section 5 introduces our algorithms. Section 6 presents our experimental evaluation. We conclude in Section 7.

CIKM/SSM 2008 Napa Valley, California, USA

2. RELATED WORK

The need for efficient information collection in a network appears in many contexts ranging from peer-to-peer systems to social networks. Our work, is mostly related to work on sampling large graphs via random walks and on personalized search.

Generating a uniform random subset of nodes of a graph via random walks is a well studied problem; it frequently arises in the analysis of convergence properties of Markov chains (e.g., see [11, 4, 14, 10]) or the problem of sampling a search engine's index [6, 5]. The basic idea is to start from any specific node, say v, and initiate a random walk by proceeding to neighbors selected at random at every iteration. Let the probability of reaching any node u after k steps of this walk be $p(u)$. It is known that if k is suitably large (the value of k depends on the topological properties of the graph, more specifically on the second eigenvalue of the adjacency matrix of the graph), this probability distribution is *stationary*, i.e that it does not depend on the starting node. However, this stationary distribution is not uniform; the probability associated with each node is inversely related to its degree in the graph. This final stationary distribution can be made uniform using techniques such as the Metropolis Hastings algorithm (see [12]), or using rejection sampling (where, after reaching a final node, the node is selected to be included in the sample with probability inversely proportional to its degree). This process can be repeated to obtain random samples of a desired size. Our research presents ways to improve upon these generic random walk methods on graphs by leveraging the fact that we need to sample from the neighborhood of a node v with a small depth (i.e., just a few links away from v).

Personalization of web search has been an open research topic for years [13]. The premise there is that by tailoring search to the individual improved result accuracy may be brought off. In a sense, utilizing information from one's digital social environment to improve on user satisfaction is a form of "extended" personalization, with the extent being defined as a function of the social neighborhood of an individual in the network. In [17] the CubeSVD approach was developed to improve Web search by taking into account clickthrough information of the type "user, query, url" and trying to automatically capture the latent factors that govern the relations among users, queries and urls. Further studies showed that taking into account such clickthrough information and building simple statistical models for user behavior can improve the result ranking quality significantly [2, 18]. Moreover, recent work [1, 17] presents strong evidence that result ranking can be greatly improved by taking into account the interactions of multiple users with a search engine. Many other approaches exist that utilize some notion of relevance feedback in order to re-rank web search results. Our research is complementary as we aim to offer performance improvements, via sampling, to the process of collecting information from clickthrough logs by exploring the underlying graph structure offered by a social network.

3. PROBLEM DEFINITION

Let G be a graph depicting connections between users in a social network. Depending on the degree of a priori knowledge of the network structure, such a graph can be centralized, that is its structure is fully known or distributed, in a sense that each node in the graph is aware of its adjacent nodes only. Centralized graphs are typical in social networking sites in which complete knowledge of users's network is maintained (e.g., del.icio.us, flickr, etc.). Distributed graphs, where a user is aware only of its immediate connections, are more common. Consider for example the case in ad hoc social networks formed by typical instant messaging or VoIP protocols (e.g., MSN, Skype). There are also cases that the model of the graph is between the two extremes allowing limited knowledge of a node's neighborhood. Our methods apply to these models as well, with potentially slight modifications.

Moreover the rate of change in these graphs is also an important factor. The most typical case is for such networks to change rapidly as users join and depart from the graph by forming or destroying social connections. Although one can make a case for relatively static social networks (in which the graph structure changes less frequently) in general such graphs are expected to be highly dynamic. We focus on dynamic networks (either centralized or distributed) but also treat the relatively easier case of static networks.

Let nodes in the graph represent users. For each user we assume availability of a log accumulated over time. The log, in its most simple form, at node v, has the form $(x, count_x^v)$ where x is an item and $count_x^v$ is the number of times x has been endorsed by user v (or a numeric value that represents the endorsement of user v to item x). Endorsement of an item is defined in a generic sense and it may have various instantiations, for example clicking on a url, renting a DVD, rating a movie, buying from a seller on an e-market, etc. Endorsements of items by users in the neighborhood of v consist valuable social information that may be utilized to form personalized rankings of items to v.

Using G and starting at v we can obtain the total count of the number of times that an item x has been endorsed by consulting the neighborhood of v at some specific depth (number of hops) d. Formally, if we define y_v as the quantity $count_x^v$, then for an item x we may obtain its exact aggregate value $Y = \sum_{i \in D_d(v)} y_i$ by visiting and querying the log at every node in the specified vicinity of v, $D_d(v)$.

However, crawling the entire neighborhood $D_d(v)$ and computing the exact aggregate value Y for an item x at runtime may be prohibitively slow, especially as the size of the neighborhood increases in number of nodes. Therefore, we have to resort to efficient approximation methods such as sampling. By sampling we avoid visiting all nodes in the vicinity of v and thus attain improved performance. We formally define the following problem:

PROBLEM 1. *Let a graph G and a user $v \in G$. Let $D_d(v)$ a user specified vicinity of v at depth d. For a specific item x, obtain through sampling nodes of G in $D_d(v)$ an approximate value of $Y = \sum_{i \in D_d(v)} y_i$.*

Note here that the sampling process operates on a node v and should respect the underlying network structure of v's neighborhood, in a sense that all v's neighbors in depth d should have the same chance to be selected in the sample.

We are now ready to formally define the problem of interest in this paper:

PROBLEM 2. *Let a graph G and a user $v \in G$. Let $D_d(v)$ a user specified vicinity of v at depth d. Let X be a set of items. Obtain through sampling nodes of G in $D_d(v)$ an approximate ordering of the items in set X.*

To obtain a solution to Problem 2 we have to repeat the process and seek solution to Problem 1 for each item $x \in X$. The ordering, incorporates the behavior of the users with which v has some social relationship.

Note that the objective of our work is to efficiently collect information in social networks and is orthogonal to any re-ranking strategy. Once the social information has been collected, a number of strategies are possible to re-rank the items in X taking into account the item counts and possibly the distance of a sampled user to user v. Designing and evaluating a re-ranking algorithm that increases the user satisfaction is out of the scope of this paper.

4. METHODOLOGY

In this section we discuss the foundational ideas behind our sampling based approaches to solve Problem 1. We first describe an idealized approach in which we assume it is possible to efficiently obtain a uniform random sample of $D_d(v)$. Let y_1, y_2, ..., y_N be the values of the nodes in $D(v)$. Suppose we could obtain a uniform random sample S of size $n \ll N$ with $S \subset D_d(v)$ and values y_1, y_2, ..., y_n. Let y be the sample sum, i.e. $y = \sum_{i \in S} y_i$. Then it is well known that the quantity $Y' = y \cdot (N/n)$, i.e., the sample sum scaled by the inverse of the sampling fraction, is an approximation for Y. In fact, Y' is a random variable whose mean and standard deviation can be approximated (for large N) by the following sampling theorem [7].

THEOREM 1.

$$E[Y'] = Y$$

$$sd[Y'] = N \cdot \sigma / \sqrt{n}$$

Remind that in Problem 2, that is of interest in this paper, we seek for an approximate ordering of the items in a set X. This ordering can be obtained directly by the estimated sample sums without the need to scale them by the inverse of the sampling fraction (i.e., N/n). Practically, the total number of nodes in $D(v)$ (i.e. N) from which we form the sample does not need to be known. Given this basic framework, the main challenge confronting us is how to obtain a uniform or near-uniform random sample of the nodes in $D_d(v)$. We discuss this issue under assumption of *static* and *dynamic* network topologies.

4.1 Assuming Static Networks

We first consider the case where the topology of the social network is static, or changes only slowly over time (although the click-through logs, i.e., the "data" stored at each node are rapidly changing). For this case a straightforward solution exists where each node, in a precomputation phase, performs a complete crawl of its neighborhood $D_d(v)$ and selects a uniform random sample S of n nodes, whose addresses (or access paths) are then stored at the initiating node. At runtime, the value stored at each sample node is retrieved and aggregated. Clearly, such a precomputation phase is computationally intensive. However, this phase needs to be recomputed infrequently; once the social network topology has undergone significant changes.

4.2 Assuming Dynamic Networks

We next consider the case where the topology of the network is dynamic, i.e, where nodes and links are being added/ deleted to the network rapidly in addition to the data changes at each node. In such a case, it makes little sense to precompute samples of $D_d(v)$ as such samples go stale very quickly. Thus, the task of sampling from $D_d(v)$ has to be deferred to runtime. This problem is interesting because we cannot crawl the entire neighborhood $D_d(v)$ at runtime (this will be prohibitively slow). It becomes even more interesting by the fact that we are constrained to simulate random walks by only following edges of the social network.

We first make a simplifying assumption, that the graph structure of the neighborhood $D(v)$ resembles a *tree* rooted at v. The solution that we first present will consist of random walks that are initiated from the root of this tree v and follow edges towards the leaves of the tree. Later, we shall describe how to generalize this basic approach for more general graph structures that are not trees - essentially by constraining our random walks to only follow edges of a spanning tree of $D_d(v)$ rooted at v.

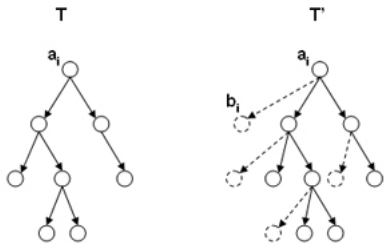


Figure 1: Tranformation of T to T'.

4.2.1 Assuming that $D_d(v)$ is a Tree

Assume that the subgraph of the social network induced by the nodes in $D_d(v)$ is a tree T with N nodes ($a_1, \ldots, a_N$), where each node is a member of $D_d(v)$. Assume that $v = a_1$ is the root and that all edges are directed downwards, i.e., from root to leaf. The maximum depth of this tree is d. Recall that each node a_i in T contains a value y_i which we wish to aggregate. To allow for better conceptualization, let us first convert the tree T to another tree T', such that the values are only at the leaf nodes, and not at internal nodes. We do this as follows: for each internal node a_i we add a leaf b_i and connect a_i to b_i via a new edge. We then move the value of a_i to b_i (see Figure 1).

To motivate our approach, let us first make the (unrealistic) assumption that for each node a_i, we know $size(a_i)$ the number of leaves in the subtree rooted at a_i. Let us also assume that each edge of the tree is weighted as follows: Let the set of children of node a_i be A_i. Consider any child node a_j in A_i. Then, weight(a_i, a_j) is defined as $\frac{size(a_j)}{\sum_{a \in A_i} size(a)}$. It is easy to see that each weight is in [0, 1] and for each node, the sum of the outgoing edge weights add up to 1.

Once T has been transformed to T', we shall perform random walks on T'. A random walk starts from the root and ends at a leaf. At every internal node, it picks an outgoing edge with probability equal to its weight. Once the walk has ended, the leaf node is returned by the random walk. The main thing to note is that any specific leaf node b_i is picked with the same probability (i.e., the product of the weights of all edges encountered along the walk). This random walk procedure can be repeated n times to obtain a uniform random sample of the nodes in $D_d(v)$ of size n.

Of course, for the above scheme to work, we have to know the sizes of each node and the weights of each edge of the tree. Clearly, computing these at runtime will be prohibitive as it will require a full traversal of the tree. Therefore, without knowing these quantities in advance, we are left with no choice but to select each outgoing edge with equal probability, i.e., $\frac{1}{|A_i|}$. But if we perform the random walk this way, we shall pick leaf nodes in a biased manner, because some leaves are more likely to be destinations of random walks than other leaves.

Correcting for the Bias: One way to correct for this bias is to let the random walk reach a leaf, but instead of accepting it into the sample, we toss a biased coin and only accept it if the coin turns up as heads. So, we have to determine what should the bias (i.e., the *acceptance probability*) of the coin be. Let the probability of reaching the leaf b_i be $p(b_i)$. Let $maxDeg$ be the maximum out-degree of the tree. The following lemma suggests how to set the acceptance probability of a leaf b_i.

LEMMA 1. *The acceptance probability of a leaf b_i is set to $C/p(b_i)$ where $C \leq 1/maxDeg^d$.*

Proof: The probability that a random walk returns a leaf b_i is equal to the probability of reaching that leaf, multiplied by its acceptance probability, which is equal to $p(b_i) \cdot C/p(b_i) = C$, which is constant for all leaves. To ensure uniform sampling we require that $C \leq min(p(b_i))$ but since $1/maxDeg^d$ is a lower bound of $p(b_i)$, it is $C \leq 1/maxDeg^d$. Note that C also represents a probability, hence it has to be at most 1. This is guaranteed since $p(b_i) \geq 1/maxDeg^d \geq C$.
end

Note that unlike the previous case where each random walk returns a random node, here we are not always guaranteed that a random node will be returned. In fact, often a random walk fails to return a node.

We refer to the maximum value of C that ensures a near-uniform random sample as C_{opt} (i.e., $C_{opt} = \frac{1}{maxDeg^d}$). However, this approach has two problems. One is of course the necessity of having to know $maxDeg$ in advance. This problem is perhaps not that crucial; after all, the maximum degree $maxDeg$ of the tree can be bounded if one has a reasonable idea of the maximum degree of the entire social network. However, the second problem is that setting such a conservative value of C (i.e., $C = C_{opt} = 1/maxDeg^d$) results in an extremely inefficient process for collecting samples. This is because a very small C, such as C_{opt}, while ensuring near-uniform random samples, almost always rejects a leaf node from being included in the sample, and consequently, numerous random walks may have to be undertaken before a leaf node is eventually accepted into the sample. To help alleviate this problem of inefficiency we propose setting a larger C.

Setting a larger C: We observe that setting C to be larger than C_{opt} would result in a larger acceptance probability per node (i.e., $\frac{C}{p(b_i)}$), which would eventually result in fewer random walks needed to generate a sample of desired size n. However, a larger C is likely to introduce non-uniformity, or bias into the sample. This is because for all leaves b_i since $C > C_{opt}$ it will be $\frac{C}{p(b_i)} > \frac{C_{opt}}{p(b_i)}$. What this means is that once leaves are reached they are more likely to be accepted into the sample and that are therefore going to be unduly over-represented in the sample. Thus, the parameter C can serve to illustrate an interesting tradeoff between ease of collecting sample nodes and the bias of the sample obtained. We investigate the effect of the parameter C in the accuracy and efficacy of our sampling methods in the experimental evaluation section.

4.2.2 Generalizing when $D_d(v)$ is not a Tree

For purposes of exposition we have been assuming that the induced subgraph of the social network over $D_d(v)$ is a tree; most induced subgraphs are not trees, but graphs with higher connectivity. However, we can adopt our solution of sampling from trees to this specific scenario by ensuring that the union of all random walks made in collecting a sample always resembles a tree. To do so, we have to keep a history of all random walks processed in response to this query, and make sure that at any point in time, their union has no cycles (see Figure 2).

More precisely, for each fresh random walk we have to ensure that it can be partitioned into two parts; the first part is a prefix of a previous random walk, while the second part is a random walk that does not visit a single node that has been visited by earlier random walks. To comply with the above constraints, when a random walk is progressing, state information can be maintained as to whether it is still a prefix of a previous random walk, or whether it has moved on into the unvisited region of $D_d(v)$. Thus, if the last node a_j along the random walk is a previously visited node, then the set of neighboring nodes that are candidates for the next random step

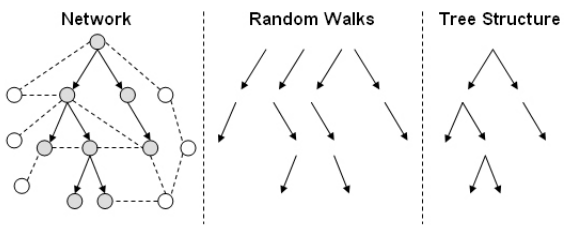


Figure 2: Random walks that obey structural properties of Tree

will be the neighbors of a_j *minus* the nodes that have been visited earlier. It is not hard to see that such an effort will ensure that the union of all random walks is a tree which is a subset of the graph induced by $D_d(v)$.

5. ALGORITHMS

In this section we present algorithmic details of our proposed methods. First, we describe $SampleDyn$, an algorithm that is able to compute a near-uniform sample of users in dynamic social networks. Then, we introduce two algorithms, $EvalSingle$ and $EvalBatch$, that make use of $SampleDyn$ in order to estimate approximate counts in a user's vicinity for a set of items.

5.1 Sampling Dynamic Social Networks

Let $D_d(v)$ be the vicinity of a user v at depth d. We introduce the algorithm $SampleDyn$ that takes as input the user v, the size of the sample n, the network depth d, and a constant value for parameter C and obtains a near-uniform random sample of users by performing random walks on the nodes of $D_d(v)$. The pseudocode is given in Algorithm 1.

Let $children(u)$ denote the nodes that are directly connected to the current node u and are either nodes that have not been visited by any of the previous random walks (unseen nodes) or nodes that extend on the prefix random walk that has been followed so far. Then, $children(u) \cup u$ represents the set of candidate nodes for the next step of the walk (line 12). Each of the candidate nodes is selected with the same probability. Thus, a random walk starts at user v and ends either when a self-link is followed, a link that connects a node with itself (line 16) or when a node in depth d has been reached (line 11). Note that, as the random walk progresses, state information is maintained regarding previous walks and visited nodes that ensures the random walk obeys structural properties of a tree. We require that $T \cup v\ has\ no\ cycle$ to represent this information (line 13). Once a node has been reached it is selected to the sample with probability equal to the acceptance probability C/p (line 17).

5.2 Estimating Item Counts

Recall that our goal, as defined in Problem 2, is to compute the counts of items in a set X, which are then used to assume an ordering. We present two approaches to estimate the ordering of a set of items in $D_d(v)$ using sampling.

5.2.1 Using Separate Samples

A first approach is to draw a separate independent sample from $D(v)$ and estimate the aggregate counts for each item. Formally, we introduce an algorithm that for each $x \in X$, obtains an approximate value of $\sum_{i \in D_d(v)} count_x^i$ through sampling nodes of $D_d(v)$. The algorithm takes as input v, d, C, n and X and returns an array of the approximate counts. We refer to this algorithm as

Algorithm 1 Sampling in Dynamic Social Networks

```
1: procedure SAMPLEDYN(u, n, d, C)
2:     T = NULL, samples = 0, Sample array of size n
3:     while samples <= n do
4:         if (v = randomWalk(u, d, C, T)) != 0 then
5:             Sample[samples++] = v
6:         end if
7:     end while
8: end procedure
9: procedure RANDOMWALK(u, d, C, T)
10:    depth = 0, p_s = 1
11:    while depth < d do
12:        pick v ∈ children(u) ∪ u with p_v = 1/(degree(u)+1)
13:        if T ∪ v has no cycle then
14:            add v to T
15:            p_s = p_s * p_v
16:            if v = u then
17:                accept with probability C/p_s
18:                if accepted then
19:                    return v
20:                else
21:                    return 0
22:                end if
23:            else
24:                u = v, depth++
25:            end if
26:        end if
27:    end while
28:    return 0
29: end procedure
```

$EvalSingle$ because it evaluates a single item at each visit to a sampled node. The pseudocode is given in Algorithm 2. While such an approach is statistically sound, the drawback is *efficiency* - this approach is unlikely to allow us to complete the re-ranking process fast enough to satisfy end users.

5.2.2 Using the Same Sample

An alternate approach is to draw a sample S only once, and reuse the same sample to estimate the aggregate counts for each item $x \in X$. We refer to this algorithm as $EvalBatch$ because it evaluates a batch of items at each visit to a sampled node. Algorithm 3 presents the pseudocode for this case.

Clearly this approach will be much faster, since we need to compute only one sample. Note however, that though practical, this process is flawed since the same sample is reused for a set of items, which are likely to exhibit strong correlations, i.e., a bad sample can affect the counts of *all* $|X|$ items.

This phenomenon is well studied in statistics, and is known as *simultaneous statistical inference* (see [15]). The classical solution proposed in [15] is to make Bonferroni corrections to ensure that the estimated counts of the items fall within their confidence intervals. A similar problem also arises in sampling-based approximate query answering techniques. For example, popular approaches in approximate query answering is to pre-compute a sample and use the same sample to answer a stream of aggregation queries (see [9]). Likewise, due to practical considerations, our proposed approach is to also reuse the same drawn sample for estimating the counts of all returned items. We experimentally evaluate the impact of such correlations and results indicate that in practice, the errors in the approximations are not unduly severe.

Algorithm 2 Counts Estimation - Separate Samples

```
1: procedure EVALSINGLE(v, d, C, n, X)
2:     S array of size n
3:     Count array of size |X|
4:     for all x ∈ X do
5:         S = SampleDyn(v, n, d, C)
6:         for all i ∈ S do
7:             Count[x] = Count[x] + count_x^i
8:         end for
9:     end for
10:    return Count
11: end procedure
```

Algorithm 3 Counts Estimation - Same Sample

```
1: procedure EVALBATCH(v, d, C, n, X)
2:     S array of size n
3:     Count array of size |X|
4:     S = SampleDyn(v, n, d, C)
5:     for all i ∈ S do
6:         for all x ∈ X do
7:             Count[x] = Count[x] + count_x^i
8:         end for
9:     end for
10:    return Count
11: end procedure
```

6. EXPERIMENTAL EVALUATION

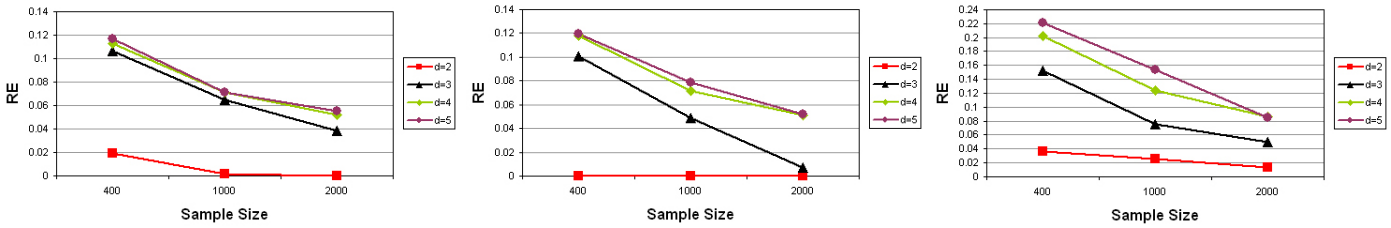
Having presented our sampling methods and algorithms we now turn to evaluation. For the needs of our experiments we consider the application area of social search. Let G be a graph depicting connections between users in a social network, where each node in the graph represents a user. For each user we assume availability of a clickthrough log accumulated over time via browsing. The log, in its most simple form, at node v, has the form $(q, url_q, count^v_{url_q})$ where q is a query, url_q is the url clicked as a result of q and $count^v_{url_q}$ is the number of times url_q has been clicked by v.

Now, consider the scenario where a query q is submitted to a popular search engine by a user v and a set of urls r_{q_v} is returned. A social search algorithm would try to personalize this result. Intuitively, an algorithm might collect information from v's social network and use this information to re-rank the results according to a re-ranking strategy. Using G and starting at v we can obtain the total count of the number of times that each url $url_q \in r_{q_v}$ has been clicked by consulting the neighborhood of v at some specific depth (number of hops) d. Then a re-ranking r'_{q_v} of r_{q_v} is possible that incorporates the behavior of the users with which v has some social relationship.

6.1 Description of Datasets

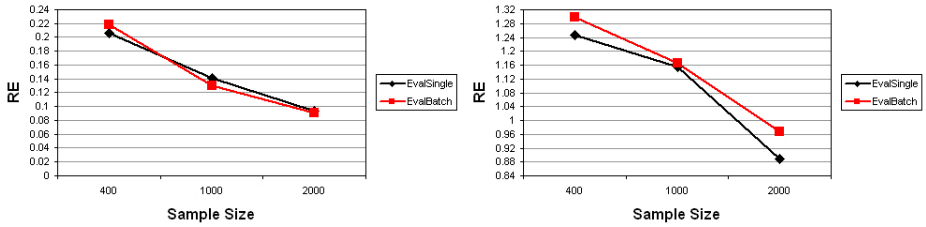
In our experiments we consider one *real* and two *synthetic* network topologies. The real network topology, *epinions-net*, is an instance of the Epinions' real graph, consisting of 75888 nodes and 450740 edges. The first synthetic topology, *uniform-net*, simulates a uniform random network topology and the second, *prefatt-net*, simulates a preferential attachment network topology.

The synthetic networks were generated so that they have similar number of vertices and edges to the real network of Epinions. Note however that since they are not based on the same model they depict different topology characteristics, such as average path length, clustering coefficient and degree distribution. Our objective is to study the application of our sampling based algorithms under di-



(a) *prefatt-net, synth_log* (b) *uniform-net, synth_log* (c) *epinions-net, real_log*

Figure 3: Sampling Accuracy



(a) *prefatt-net, synth_log* (b) *epinions-net, real_log*

Figure 4: Batch Sampling Effect

verse assumptions of network connectivity and stress any interesting differentiation on performance due to network topology.

We experiment with *real* and *synthetic* user search history logs. For the needs of our experiments we create *real_log* by randomly selecting 75888 users from the AOL dataset along with their search history logs (about 4M queries, 3M urls) [8]. The synthetic log, *synth_log*, consists of the same users as the *real_log* but we populate user's history logs with high numbers of queries and url counts for the urls of interest.

In order to generate suitable final data sets for our experiments we randomly map each of the 75888 AOL users in a search history log to the 75888 nodes of a network topology to resemble data of a *social search network*. Note that since the focus of our work is on performance we do not require that "similar" users are placed in adjacent network nodes.

6.2 Evaluation Metrics

We assess the performance of our algorithms according to *accuracy* and *efficiency* measures.

Accuracy concerns how well our sampling framework estimates the exact count of a url and how well it estimates the ordering of a set of urls in the vicinity of a user. To assess the accuracy in the first case we use the *Relative Error* (RE) metric, which is usually employed to express accuracy of an estimate. Formally, the relative error between an exact value y and an estimated value $\hat{y}$ is given by:

$$RE = |\frac{y - \hat{y}}{y}|$$

To assess the accuracy in the second case we employ two metrics that are usually considered for comparison of ranked lists, the *Normalized Spearman's Footrule Distance* and the *Precision at k*. *Spearman's Footrule Distance* measures the distance between two ranked lists. Formally, given two full lists r' and r that rank items of the set r, their Spearman Footrule Distance is given by

$$F(r, r') = \sum_{e \in r} |r'(e) - r(e)|$$

After dividing this number by the maximum value $(\frac{1}{2})|r|^2$, one can obtain a normalized value of the footrule distance, which is always between 0 and 1. *Precision at k* (*P@K*) measures the precision at a fixed number of retrieved items (i.e., top k) of the ordered list r' and the ordered list r. Assume $TopK$ and $TopK'$ are the retrieved items of r and r' respectively, then the *precision at k* is defined as

$$P@K = \frac{|TopK' \cap TopK|}{k}$$

Efficiency concerns the cost of our sampling framework. To assess the efficiency of our sampling algorithms we use as a surrogate for cost the number of random walks performed to obtain n samples from the network. Formally, the cost of the sampling is

$$Cost = \#Random\ Walks$$

6.3 Experimental Results

6.3.1 Sampling Accuracy

In our first set of experiments, we evaluate the sampling accuracy of *EvalSingle* under assumption of different network topologies. More specifically, we first determine the *top* url when submitting q to Google. Then, we apply *EvalSingle* to quickly compute an estimate of its count in $D_d(v)$. The process is repeated many times for different queries and users and at each iteration the relative error of the estimated url count to the exact url count is computed. We experiment for variable network depth d and sample size n. Figure 3 presents the results for the different topologies.

In all topologies for a fixed network depth d the sampling accuracy increases with the sample size n (i.e., the average relative error decreases for larger sample sizes). Furthermore, for a fixed sample size n the sampling accuracy decreases as d increases. The

observed behavior is in accordance with theory. The total population N (from which we sample) increases with the network depth d ($d \sim N$) and from the Theorem 1 is true that the sampling standard error is proportional to the total population N ($sd \sim N$) and inversely proportional to the number of samples ($sd \sim \frac{1}{n}$).

In the case of the synthetic network topologies, for a fixed depth d and a fixed sample size S the sampling accuracy in the *uniform-net* is better than the one in the *prefatt-net*. This is explained by the fact that the preferential model has a systematically shorter average path length than the random graph model [3]. As a result, for a fixed depth d the average total population N of the *prefatt-net* is larger than the one of the *uniform-net*. Since N is larger in *prefatt-net* than in *uniform-net* for a fixed d the standard error will also be larger according to Theorem 1 ($sd \sim N$).

In the case of the real network topology (*epinions-net* with *real_log*) the sampling accuracy results are not directly compared to the results in the synthetic data. This is due to the fact that the exact url counts in the real data are much smaller and leverage the sampling performance. As a result, even if trends are identical to the ones observed in the synthetic data, slightly larger absolute errors are demonstrated.

For the rest of the experimental evaluation we set the parameter $d = 4$. This is a reasonable choice for our data set. For $d = 1$ and $d = 2$ the network populations are small and sampling is not needed. For $d = 4$ we are able to reach almost all nodes in the network, thus, there is no need to consider $d > 4$. Between $d = 3$ and $d = 4$ we choose $d = 4$ that will increase the network population and therefore make the approximation problem harder.

6.3.2 Batch Sampling Effect

In Section 5 we introduced *EvalBatch*, which is much faster than *EvalSingle* as it needs to compute only one sample. However, it is also flawed since the same sample is reused for a set of urls, which are likely to exhibit strong correlations. In this set of experiments we evaluate the impact of such correlations.

Accuracy results reported are average relative errors over a number of runs for random users and queries. At each run the set of the top-k urls is determined by submitting the query q to Google. Then, *EvalSingle* and *EvalBatch* compute the estimates of the url counts, which are then compared to the exact counts.

Figure 4(a) presents the results of the experiment in the case of *prefatt-net* and *synth_log* for variable network depth d and sample size n. Results indicate that using the same sample for evaluating url counts has a low effect in the sampling accuracy. Note that when we generated the synthetic data we tried to avoid any correlation by assigning queries and urls to users randomly. In the absence of any correlations in the data, it was expected that in the case of synthetic data, the two algorithms should perform equally. From now on, whenever similar trends are demonstrated between the two synthetic network topologies, we only present the results for *prefatt-net* and omit the details for *uniform-net*. The preferential attachment model is favored since it widely exists in the social information networks of interest and is also more challenging for our sampling methods.

In the case of real data (*epinions-net* with *real_log*), we expect that data exhibit some sort of correlation in the urls. Indeed, this effect is depicted in Figure 4(b) where the difference between *EvalSingle* and *EvalBatch* is more obvious than it was in the synthetic data. However, their accuracy performance is still comparable.

Since the errors in the computation of estimates that are due to the use of the same sample are not unduly severe, our proposed approach is to use *EvalBatch* for estimating the counts of urls which is more efficient.

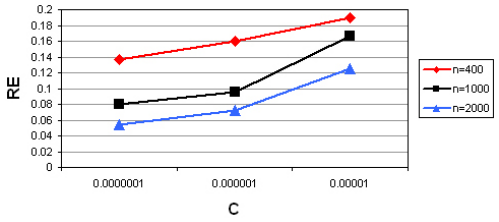


Figure 5: Correcting for Bias

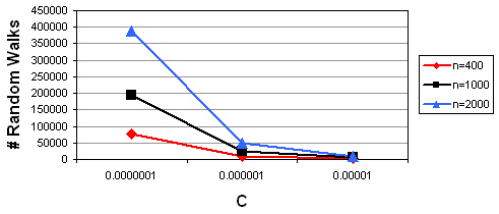


Figure 6: C vs. #Random Walks

6.3.3 Ordering Accuracy

The end objective of our method is to approximate the ordering of a set of urls in a user's neighborhood and not necessarily their exact counts. In this set of experiments we assess the ordering accuracy of our sampling algorithms. Formally, for each query q we retrieve the top-k urls returned by a search engine, such as Google. Let this set of urls be r_q. Then, for a given user v and network depth d we compute the ordering r_{q_v} of these urls in the $D_d(v)$ of v according to their exact counts. On the other hand, *EvalBatch* estimates the url counts of each $url_q \in r_q$ and comes up with another ordering r'_{q_v}. The two lists are then compared using the *Normalized Spearman's Footrule Distance* and the *Precision at k* metrics. Results are averaged over a number of random users and queries.

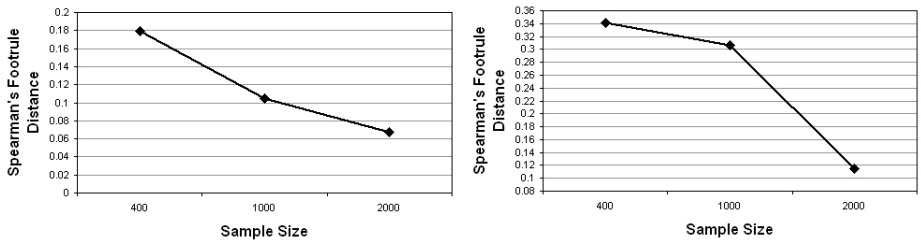
Figure 7(a) reports on the Normalized Spearman's Footrule Distance of the two orderings in the case of *prefatt-net* and *synth_log* for $d = 4$ and variable sample size n. The distance decreases as the sample size increases signifying that our estimated counts become increasingly more accurate. The same trend becomes evident in the case of *epinions* and *real_log* as shown in Figure 7(b).

Figure 8(a) reports on the *Precision at k* (P@K) of the two orderings in the case of *prefatt-net* and *synth_log* for $d = 4$ and variable sample size n. Precision at top k urls is high and increases with the sample size. The same trend becomes evident in the case of *epinions* and *real_log* as shown in Figure 8(b).

In the case of the real network topology (*epinions-net* with *real_log*) the ordering accuracy results are not directly compared to the results in the synthetic data. This is due to the fact that the exact url counts in the real data are much smaller and leverage the sampling performance. As a result, even if trends are identical to the ones observed in the synthetic data, slightly worse performance of the ordering accuracy is demonstrated in the course of both metrics.

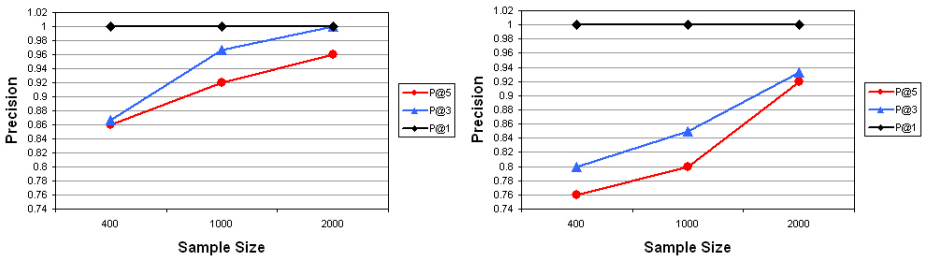
6.3.4 Effect of C

As discussed in Section 4, parameter C can serve to illustrate an interesting tradeoff between ease of collecting sample nodes from a tree and the bias of the sample obtained. This set of experiments aims to demonstrate this tradeoff. We run experiments on the synthetic data, *prefatt-net* with *synth_log*, for network depth d=4 and for variable values of the parameter C and the sample size n. We report on the sampling accuracy in terms of relative error RE and



(a) *prefatt-net, synth_log* (b) *epinions-net, real_log*

Figure 7: Ranking Accuracy - Spearman's Footrule Distance



(a) *prefatt-net, synth_log* (b) *epinions-net, real_log*

Figure 8: Ranking Accuracy - Precision at k

the sampling cost in terms of the number of random walks needed to form the sample. The values of C were arbitrarily selected to better exhibit the tradeoff between accuracy and efficiency. Figure 5 clearly demonstrates the effect of the parameter C in the sampling accuracy, where as C gets larger the relative error increases. Meanwhile, Figure 6 demonstrates the effect of C in the sampling cost, where as C gets smaller the number of random walks needed to form the sample increases and eventually renders sampling inefficient. Depending on the application area, one would need to adjust this parameter to balance time and accuracy performance.

7. CONCLUSIONS

In this paper, we focused on improving the performance of information collection from the neighborhood of a user in a social network. Our approach is to use sampling-based methods to quickly approximate quantities of interest. We demonstrated the utility of our approach by running experiments on real and synthetic data sets and showed that our algorithms are able to efficiently estimate the ordering of a set of items in a user's network giving rise to any search re-ranking strategy. Our sampling schemes are general and can be utilized in a variety of strategies aiming to efficiently compute interesting quantities in a dynamic social network.

8. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, 2006.

[2] E. Agichtein, E. Brill, S. T. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *SIGIR*, pages 3–10, 2006.

[3] R. Albert and I. Barabasi. Statistical Mechanics of Complex Networks. *Modern Physics Reviews*, pages 47–97, 2002.

[4] D. Aldous. On the markov chain simulation method for uniform combinatorial distributions and simulated annealing. *Probab. Engrg. Inform. Sci.*, 1(2):33–46, 1987.

[5] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. In *VLDB*, 2000.

[6] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. In *WWW*, 2006.

[7] W. G. Cochran. *Sampling Techniques, 3rd Edition.* John Wiley, 1977.

[8] C. T. G. Pass, A. Chowdhury. A Picture of Search. *InfoScale*, 2006.

[9] M. N. Garofalakis and G. P. B. Approximate query processing: Taming the terabytes. *VLDB Tutorial*, 2001.

[10] D. Gillman. A chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27(4), 1998.

[11] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Perform. Eval.*, 63(3):241–263, 2006.

[12] W. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1), 1970.

[13] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, 2002.

[14] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *FOCS*, 1989.

[15] R. G. Miller. *Simultaneous Statistical Inference*. Springer Verlag, Heidelberger, Berlin, 1981.

[16] A. Mislove, K. P. Gummadi, and P. Druschel. Exploiting Social Networks for Internet Search. *HotNets*, 2006.

[17] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen. Cubesvd: a novel approach to personalized web search. In *WWW*, 2005.

[18] J. Teevan, S. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. *SIGIR'05*.