Sampling Online Social Networks

Manos Papagelis, Gautam Das, *Member*, *IEEE*, and Nick Koudas, *Member*, *IEEE*

Abstract—As online social networking emerges, there has been increased interest to utilize the underlying network structure as well as the available information on social peers to improve the information needs of a user. In this paper, we focus on improving the performance of information collection from the neighborhood of a user in a dynamic social network. We introduce sampling-based algorithms to efficiently explore a user's social network respecting its structure and to quickly approximate quantities of interest. We introduce and analyze variants of the basic sampling 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.

Index Terms—Information networks, search process, query processing, performance evaluation of algorithms and systems

1 Introduction

The changing trends in the use of web technology that aims to enhance interconnectivity, self-expression, and information sharing on the web have led to the emergence of online social networking services. This is evident by the multitude of activity and social interaction that takes place in web sites like Facebook, Myspace, and Twitter to name a few. 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, or mobile geosocial networks. Although interactions with people beyond one's contact list is currently not 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 information in order to improve on information retrieval tasks of social peers. Such tasks are in the core of many application domains. To further motivate our research, we discuss in more detail the case of *social search*. Social search or a social search engine is a type of search method that tries to determine the relevance of search results by considering interactions or contributions of users. The premise is that by collecting and analyzing information from a user's explicit or implicit social network we can improve the accuracy of search results. The most common social search scenario is the following:

1. A user v in a network submits a query to a search engine.
2. The search engine computes an ordered list L of the most relevant results using a global ranking algorithm.
3. The search engine collects information that lies in the neighborhood of v and relates to the results in L.
4. The search engine utilizes this information to reorder the list L to a new list L' that is presented to v.

The utility of social search has been established via experimental user studies. For example, in [1], Mislove et al. report improved result accuracy for web search when urls for a query are not ranked based on some global ranking criteria, but based on the number of times people in the same social environment endorsed them. Many ideas have been suggested to realize online social search; from entirely human search engines that utilize humans to filter the search results and assist users in clarifying their search requests to social-influenced algorithms that exploit a user's web history log to influence result rankings, so that pages that she visits more often are ranked higher. In any case, the goal is to provide end users with a limited number of relevant results informed by human judgement, as opposed to traditional search engines that often return a large number of results that may not be relevant. These are all examples of tasks that require to visit and probe a large number of peers in the extended network of an individual for information that lies locally in their logs, and then use this information to improve the quality of search experience.

Despite the fact that many algorithms and tools exist for analysis of networks, in general, these mainly focus on analysis of the properties of the network structure and not on the content of the nodes. They also typically not operate on user specific graphs (i.e, users' neighborhoods), but on the whole graph. Instead, for many modern applications, it would be beneficial to design algorithms that operate on a single node. For example, in the case of social search, it would be beneficial to design algorithms that starting from a specific user in the network, crawl its

- *M. Papagelis and N. Koudas are with the Department of Computer Science, Bahen Center for Information Technology, University of Toronto, 40 St. George Street, Toronto, ON M5S 2E4, Canada. E-mail: {papaggel, koudas}@cs.toronto.edu.*
- *G. Das is with the Computer Science and Engineering Department, University of Texas, 626 Engineering Research Building, 500 UTA Blvd., Arlington, TX 76019. E-mail: gdas@uta.edu.*

Manuscript received 16 Feb. 2011; revised 7 Nov. 2011; accepted 9 Nov. 2011; published online 8 Dec. 2011.
Recommended for acceptance by B.C. Ooi.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-02-0073.
Digital Object Identifier no. 10.1109/TKDE.2011.254.

(extended) neighborhood and collect information that lies on their close peers. Such networks may consist of thousands of users and their structure may not be static, thus a complete crawl of all social peers is infeasible. Therefore, efficient methods are required. Based on these observations we focus on improving the performance of information collection from the neighborhood of a user in a social network and make the following contributions:

- We introduce sampling-based algorithms that given a user in a social network quickly obtain a near-uniform random sample of nodes in its neighborhood. We employ these algorithms to quickly approximate the number of users in a user's neighborhood that have endorsed an item.
- We introduce and analyze variants of these basic sampling schemes 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 basic 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.

Our research aims to offer performance improvements, via sampling, to the process of (almost) uniformly collecting information from user logs by exploring the underlying graph structure of a social network. Note that our work does not make any assumption about the semantics of neighboring nodes, thus we do not assume that users that are closer in the network may exhibit some behavioral similarity, have similar interests or so. On the contrary, in designing our algorithms we assume that any node in the extended neighborhood of a user is by definition equally important, independently of how close or far it lies from the initiator node. How one can utilize this information to increase user satisfaction is orthogonal to our work and out of the scope of this paper.

The rest of the paper is organized as follows: Section 2 formally defines the problems of interest in the paper and introduces notation. Foundational ideas of our sampling methods are presented in Section 3, and Section 4 describes the algorithmic details that implement them. We experimentally evaluate our algorithms in Section 5. In Section 6, we review related work and conclude in Section 7.

2 Problem Definition

The social network structure can be modeled as a graph G with individuals representing nodes and relationships among them representing edges. 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). 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 and Skype). There are also cases that the model of the graph is between the centralized and the fully distributed allowing limited knowledge of a node's neighborhood typically controlled by the node in terms of privacy settings (e.g., LinkedIn and Facebook). Our methods apply to these models as well, with slight modifications. The *rate of change of the structure* of these networks 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. In any case, we assume that the *rate of change of the content* in these networks is high. Given such an environment we define the following two problems of interest in this paper.

2.1 Sampling Nodes in Social Networks

Using G and starting at v we would like to be able to obtain the set of nodes in the neighborhood $D_d(v)$ of v at some specific depth d (i.e., at most d hops away of v). However, crawling the entire $D_d(v)$ 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 $D_d(v)$ and thus attain improved performance. We formally define the following problem:

Problem 1. *Let a graph G and a user $v \in G$. Let $D_d(v)$ be a user specified vicinity of v at depth d. Using G and starting at v, obtain a uniform random sample of the nodes in $D_d(v)$.*

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 users in $D_d(v)$ should have the same chance to be selected in the sample. Thus, we assume (by definition of our problem) that any node in $D_d(v)$ should be equally important for the information task, and ignore other semantics, such as the distance of a node from v.

2.2 Sampling Information in Social Networks

For each user in G we assume that a log accumulated over time is available. Let the log at node v have 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, rating a movie, etc. Endorsements of items by users in the neighborhood of v comprise valuable social information that may be utilized to provide personalized rankings of items to v. In many social information tasks (such as in social search) we are interested in the *relative order* or *ranking* of a set of items X in the social network of v. Using G and starting at v we would like to obtain the total count of the number of times that each item $x \in X$ has been endorsed by

TABLE 1
Sampling Notation

Notation	Explanation
N	Number of nodes in $D(v)$
S	Set of nodes in Sample
n with $n \ll N$	Number of nodes in Sample
$y_1, y_2, ..., y_N$	Values of the nodes in $D(v)$
$y_1, y_2, ..., y_n$	Values of the nodes in S
σ	Standard deviation of values in $D(v)$

consulting the neighborhood of v at some specific depth d (i.e., at most d hops away of v). Formally, if we define y_v as the quantity $count_x^v$, then for an item $x \in X$ 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, visiting any node in $D_d(v)$ and computing the exact aggregate value Y for any item $x \in X$ 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. We formally define the following problem:

Problem 2. *Let a graph G and a user $v \in G$. Let $D_d(v)$ be 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 estimate of the ordering of the items X in $D_d(v)$.*

Once the social information has been collected, a number of personalization strategies are possible to rerank the items in X taking into account semantics of the collected information, such as the item counts or the distance of a sampled user to user v. Designing and evaluating a reranking algorithm that increases the user satisfaction is out of the scope of this paper.

3 Sampling 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 $D_d(v)$. The sampling notation we use is shown in Table 1 for reference. Let $y_1, y_2, \ldots, y_N$ 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 $y_1, y_2, \ldots, y_n$. 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 well-known sampling theorem [2].

Theorem 1.

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

The standard deviation $sd[Y']$ provides an estimate of the error in estimating Y by Y'. Since σ, the standard deviation of values in $D(v)$, is usually not known in advance, it can be itself estimated by computing the standard deviation σ' of the sample; thus $sd[Y']$ is estimated as $N\sigma'/\sqrt{n}$. More fine-grained error estimations such as *confidence intervals*are also possible; see [2] for details. Remind that in Problem 2, 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. We will only assume a priori knowledge of this number when evaluating the accuracy of our sampling methods, in order to compare estimated with actual aggregate values. Given this 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.

3.1 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 clickthrough 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.

3.2 Assuming Dynamic Networks

We next consider the case where the topology of the network is dynamic, i.e., where the network structure changes frequently 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 challenging because we cannot crawl the entire neighborhood $D_d(v)$ at runtime (this will be prohibitively slow). It becomes even more challenging by the fact that we are constrained to simulate random walks by only following edges of the social network. As we discuss in Section 6, there are methods to generate a uniform random subset of nodes of a large graph via random walks. However, in our case, we can improve upon generic random walk methods on graphs as we can leverage the fact that we need to only sample from the neighborhood $D_d(v)$ of a node v with a small depth d (i.e., just a few links away from v). Consequently, we are able to develop even more efficient random walk procedures. 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 toward the leaves of the tree. Later, we 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.

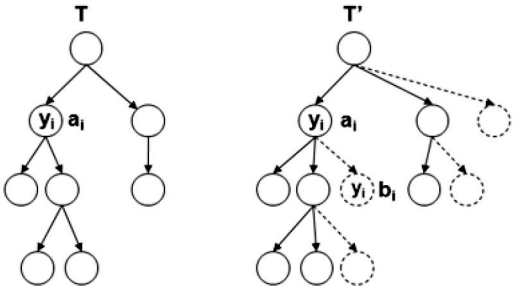

Fig. 1. Transformation of T to T'.

3.2.1 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 downward, 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 Fig. 1). Note that for each internal node in the original tree, the *degree* in the new tree has been increased by one, and that now the number of leaves is N. 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 adds 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 following lemma determines the probability of returning any specific leaf node.

Lemma 1. *The probability of the random walk returning any specific leaf node b_i is $p(b_i) = 1/N$*

Proof. (*Sketch*) The proof hinges on the way the edge weights of the tree have been defined. Note that each leaf is picked with probability equal to 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 (with replacement) of the nodes in $D_d(v)$ of size n. Thus, after we have done n independent random walks, we will have collected n leaves (may contain duplicates), i.e., the set of n leaves is a near-uniform random sample of the entire set of N nodes. 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. We explore the effect of this bias in the sampling accuracy in the experimental section.

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 $p(b_i)$ (i.e., the product of $1/outdegree$ of all nodes along the path from root to leaf). 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 to achieve near-uniform random sample.

Lemma 2. *If the acceptance probability of a leaf b_i (i.e., the bias of the coin) is set to $C/p(b_i)$ where $C \leq 1/maxDeg^d$, then a random walk performs near-uniform sampling.*

Proof. Let C be a constant that controls the probability with which a leaf node is returned by a random walk. We need to find a value of C that ensures the probability of the random walk returning any specific leaf node b_i is the same for all leaves. It is easy to see that an appropriate value for C depends on the structure of the tree and the number of leaves in it, which we cannot assume to be known. Without any prior knowledge of the tree structure we can, at the worst case, assume that it is a full k-ary tree, a tree where each node has either 0 or k children, depending on whether it is a leaf or an internal node. It is known that for a full k-ary tree with height h, the upper bound for the maximum number of leaves is k^h. In our case $k = maxDegree$ and $h = d$. Therefore, the maximum number of leaves in a tree on which our random walks operate would be $maxDegree^d$. Thus, to ensure near-uniform sampling of any leaf node we require that $C \leq 1/maxDegree^d$. Note also that since $C/p(b_i)$ represents a probability, it has to be at most 1. This is guaranteed since $p(b_i) \geq 1/maxDeg^d$. □

Assuming that $maxDeg$ is known in advance 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. 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. The probability of *success* p_s of a random walk returning nonempty is described by the following lemma.

Lemma 3. *The probability of success of the random walk returning any specific leaf node b_i is $p_s = C$.*

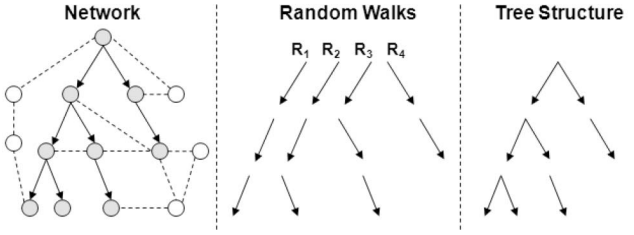

Fig. 2. Random walks that obey tree structure.

Proof. The probability of success p_s that a random walk returns a leaf b_i is equal to the probability of reaching that leaf ($p(b_i)$) multiplied by its acceptance probability ($C/p(b_i)$). Thus, $p_s = p(b_i) \cdot C/p(b_i) \Rightarrow p_s = C$. □

Thus if we iterate this random walk several times and collect all returned nodes, we will be able to get a near-uniform random sample of any desired size. The following lemma quantifies the expected number of random walks needed to generate a near-uniform sample of size n.

Lemma 4. *The expected number of random walks required to collect a uniform random sample of size n is n/p_s.*

Proof. The proof hinges on the probability of success p_s. Since the probability that a random walk returns a node is C, we need n/p_s random walks to collect n nodes. □

3.2.2 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 Fig. 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 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 $D_d(v)$.

3.3 Tuning C

Despite its neatness, our sampling approach suffers one inherent drawback. Setting such a conservative value of C (i.e., $C \leq 1/maxDeg^d$) results in an extremely inefficient process for collecting samples. This is because a very small C, 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. Let us refer to the maximum value of C that ensures a near-uniform random sample as C_{opt} (i.e., $C_{opt} = \frac{1}{maxDeg^d}$). Note 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 nonuniformity, or bias into the sample. This is because for all leaves b_i since $C > C_{opt}$ it will be $\frac{C}{p(b_i)} > \frac{C_{opt}}{p(b_i)}$. This means that once leaves are reached they are more likely to be accepted into the sample and that are therefore going to be unduly overrepresented 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 further investigate the effect of adjusting the parameter C and demonstrate this tradeoff between accuracy and efficacy by running experiments on a synthetic network of 75k nodes and 450k edges, for network depth $d = 4$ and for variable values of C and sample size $n = \{400, 1000, 2000\}$. We report on the sampling accuracy in terms of relative error RE and the sampling cost in terms of the number of hops in the random walks needed to form the sample. The values of C were arbitrarily selected to clearly exhibit the tradeoff between accuracy and efficiency. Fig. 3 (left) demonstrates the effect of C in the sampling accuracy, where as C gets larger the relative error increases. Meanwhile, Fig. 3 (right) demonstrates the effect of C in the sampling cost, where as C gets smaller the number of hops in the 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 according to needs. A method to select a proper C is discussed next.

3.3.1 Selecting a Proper C

An adequate heuristic would be to set the parameter C to be equal to $1/N$, where N is the number of leaves in the transformed tree from which we want to sample. Recall that N represents the number of nodes in the original tree (before the transformation). This would assume that all leaves have the same probability to be selected, and as $1/N$ is expected to be much smaller than $1/maxDeg^d$, fewer random walks will be needed to generate a sample of desired size n. However, we cannot assume that the number of the nodes in the tree N is known a priori. Finding N would require to perform an exhaustive search on the graph, using depth-first-search (DFS) or breadth-first-search (BFS), but this method is impractical in our setting. The only feasible approach would be to try to estimate N. Estimating the size of a tree is a challenging problem that arises in many domains. It commonly appears as the problem of estimating the size of backtracking trees or branch-and-bound procedures for solving mixed integer

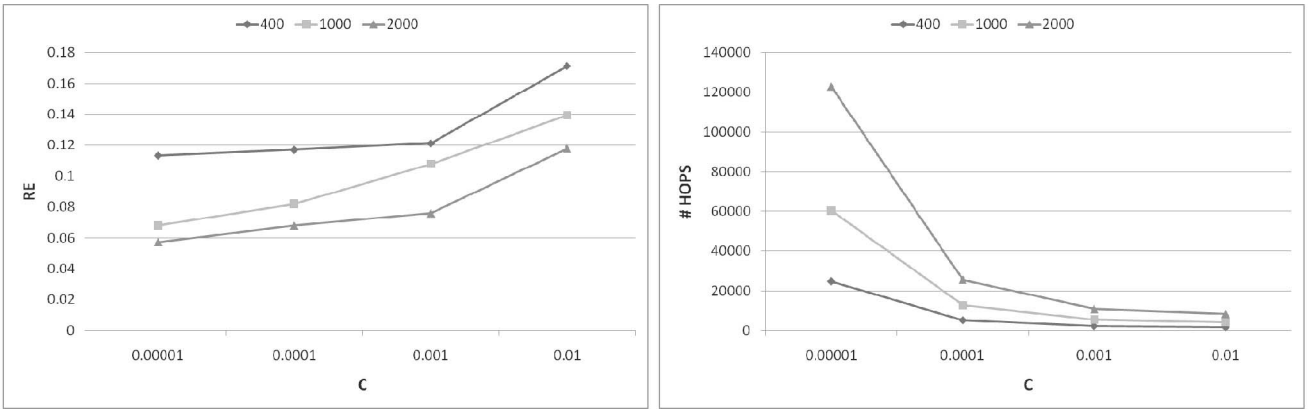

Fig. 3. Effect of C in sampling accuracy (left) and sampling cost (right).

programming problems. Knuth [3] was the first to discuss the problem and proposed a method based on random probing that would estimate the size of a search tree and eventually the running time of a backtracking program. In [4], Cornujols et al. predict the size of a branch-and-bound search tree by building a simple linear model of the branching rate using the maximum depth, the widest level and the first level at which the tree is no longer complete. Later, Kilby et al. [5] proposed two methods to solve the problem: one based on a weighted sample of the branches visited by chronological backtracking and another based on a recursive method that assumes that the unexplored part of the search tree will be similar to the part that has already been explored. Finally, statistical techniques have been applied to predict the size of search trees; in [6], Kautz et al. describe how Bayesian methods can be used to build models that predict the runtimes of constraint satisfaction algorithms; these predictions can then be used to derive good restart strategies. Unfortunately, these methods cannot be applied in our problem. More relevant to our setting are the methods described in [7], where the Katzir et al. try to estimate the size of an undirected graph or a social network. Their methods work by performing random node sampling and then counting the number of collisions (pair of identical nodes) or the number of nonunique elements (elements that have been sampled at least once before) in the sample. Despite their elegance, these methods cannot be applied in our case, as they assume that nodes are sampled from the graph's stationary probability, which we cannot assume to be known in our setting. The size of a tree can also be estimated using a method known as *mark and recapture*. This method is used in ecology to estimate the size of a natural population and involves marking a number of individuals in a population, returning them to that population, and subsequently recapturing some of them as a basis for estimating the size of the population at the time of marking and release. The main idea hinges on the so called "birthday paradox" effect. After sampling r nodes uniformly at random we expect to encounter $L \approx r^2/2N$ collisions (nodes already picked). Then, an estimate $\hat{N}$ of N can be computed by $\hat{N} = r^2/2L$. However, in order to use this method nodes need to be sampled uniformly. Going back to our problem, the following strategy can be followed to find a proper C.

- Set C to a very small value, that can guarantee uniform sampling of nodes, such as $C = 1/maxDeg^d$.
- Obtain r samples $\{x, \ldots, x_r\}$ using our sampling methodology, and count the number of collisions L. A collision is a pair of identical samples. Formally, let $P_{i,j}$ to be 1 if $x_i = x_j$ and 0 otherwise. Then, $L = \sum_{i<j} P_{i,j}$.
- Use the *mark and recapture* method to estimate the tree size N by $\hat{N} = r^2/2L$.
- Reset the C parameter to $C = 1/\hat{N}$ and use this value for feeding the sampling method in subsequent runs.

We investigate the accuracy of this method by running experiments on a synthetic network of 75k nodes and 450k edges, for variable network depth $d = \{4, 5\}$ and for variable sample size n (expressed as percent of nodes in the original network). We report on the normalized *mean absolute relative error*, i.e., $|N - \hat{N}|/N$, where N is the true size of the network and $\hat{N}$ is our estimate of it. Plots were produced by averaging over 10,000 independent experiments (we experimented with 100 random users and for each user we estimate the network size at depth d, 100 independent times). Fig. 4 demonstrates that as the sample size increases the error decreases, while the error converges faster in the case of the larger network. The advantage of this method is that taking only a small sample can guarantee an accurate estimate of N. For example, by using a sample of 10 percent of the network the estimation ensures a normalized mean absolute error of less than 20 percent. Note, that in setting C we only require a rough estimate of the network size, and as such this method is adequate. Throughout the experimental evaluation we set the C parameter to be equal to $1/N$ and do not further investigate the performance of the tree size estimator. Estimating the size of a tree or a network is an orthogonal problem; it is expected that other estimators might be advantageous to our methods with slight modifications, such as the ones presented in [7].

4 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

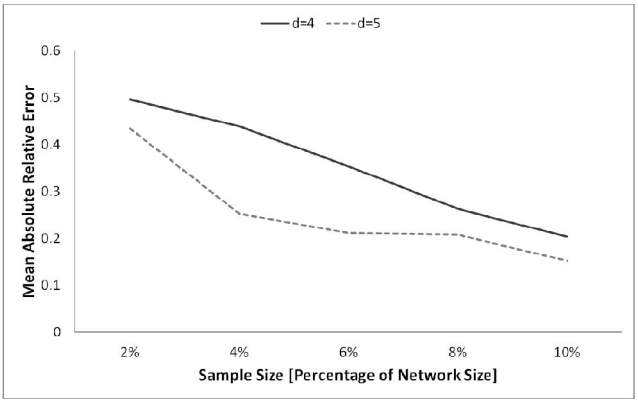

Fig. 4. Network size estimation error.

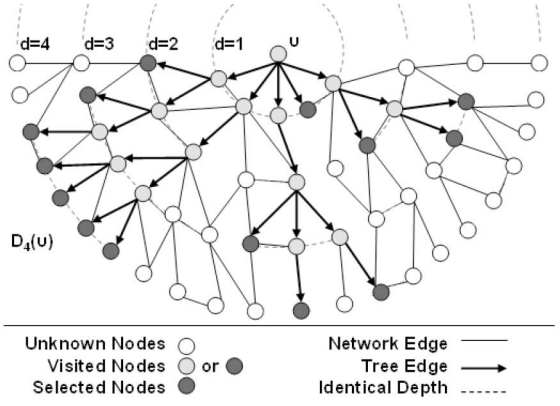

Fig. 5. Example random walks guided by $SampleDyn$.

two algorithms, $EvalSingle$ and $EvalBatch$, that utilize $SampleDyn$ in order to estimate counts for a set of items in a user's vicinity.

4.1 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_s (line 17). For example, Fig. 5 shows a network of depth $d = 4$ around user v along with a set of nodes that have been visited by previous walks (light and dark shadowed nodes) and nodes that have already been selected for the sample (dark shadowed nodes). Note that the random walks are forced to obey structural properties of a tree. The tree is defined by the union of the earlier successfull random walks (indicated by the directed edges). The structure of the tree remains valid throughout the current query evaluation (i.e., for as long as n sample nodes have been selected). Different tree structures are possible every time a query is evaluated at v depending on the sequence of the successful random walks and the underlying network structure of the user's vicinity at depth d.

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
```

4.2 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.

4.2.1 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 $EvalSingle$ 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.

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
```

4.2.2 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 [8]). The classical solution proposed in [8] 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 precompute 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 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
```

4.3 Cost Analysis

Our sampling algorithms provide an alternative to performing an exhaustive search or *crawling* on the network of a user using a depth-first-search or breadth-first-search. Both DFS and BFS assume that there is a designated initiator node from which the search starts and define a DFS or BFS tree. At the end, nodes at distance d from the initiator appear at level d of the tree. In this paragraph, we present we simple cost model that helps to analyze and compare the complexity of our basic sampling algorithms to that of crawling.

TABLE 2
Algorithm Complexity

Algorithm	Communication	Time
Distributed DFS	$O(E)$	$O(E)$
Distributed BFS	$O(E)$	$O(E)$
Random Walks	$O(n/p_s \cdot d)$	$O(n/p_s \cdot d)$

4.3.1 Cost Model

Let $D_d(v) = (N, E)$ be the neighborhood of a user v at depth d, where N is the set of nodes and E the set of links in the network. Nodes are autonomous in that they perform their computation and communicate with each other only by sending messages. Each node is unique and has local information, such as the identity of each of its neighbors. We assume that each node handles messages from and to neighbors and performs local computations in *zero time*, meaning that communication delays outweigh local computations on the nodes. The same assumption is made by Makki and Havas [10]. We evaluate the complexity of our algorithms using standard complexity measures. The *communication complexity* is the total number of messages sent, while the *time complexity* is given by the maximum time elapsed from the beginning to the termination of the algorithm. We assume that delivering a message over a link corresponds to traversing an edge to visit a node, a *hop*, and that delivering a message over a link (i.e., performing a hop) requires at most one unit of time. Therefore, for our algorithms, we use as a surrogate for both communication and time costs the number of hops performed during the execution of the algorithm.

Table 2 presents the communication and time complexities of the algorithms. Note that both DFS and BFS algorithms require at least one message to be sent on each edge, yielding a communication and time complexity of at least $O(E)$ [11], [12]. On the other hand, our sampling algorithm has communication and time complexity of $O(n/p_s \cdot d)$, where d is at most the length of each random walk (i.e., the number of hops per random walk) and n/p_s is the expected number of random walks required to collect a uniform sample of size n as shown in Lemma 4. Given that typically $n \ll N$ and assuming an appropriate value for parameter C as discussed before, our sampling algorithm is expected to outperform both DFS and BFS algorithms for both the communication and time complexities. We give examples of performance on typical networks in the experimental section.

5 Experimental Evaluation

Having presented our sampling methods and algorithms we now turn to evaluation. For the needs of our experiments we make a case of a social search application. 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. A few years ago, it would be difficult to assume that such a log exists due to privacy issues. However, recently, a number of Web2.0 services gather such kind of information. The most notable example might be Google's web history service.[1] But also, Bing's and Facebook's attempts to incorporate in search results a feature that shows you the opinion of your friends as it relates to that search, through the Facebook Instant Personalization feature.[2] Therefore, is reasonable to assume existence of such information. 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 rerank the results according to a reranking 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.

5.1 Description of Data Sets

For the needs of our experimental evaluation, we consider real and synthetic *network topologies* along with real and synthetic *user search history logs* (i.e., clickthrough data). To come up with social search logs suitable for our experiments we had to combine these sources. Below we provide details on data characteristics and the data collection process.

Network Topologies. In our experiments, we consider one *real* and two *synthetic* network topologies. The real network topology, *epinions-net*, is an instance of the Epinions'[3] real graph. The first synthetic topology, *uniform-net*, simulates a uniform random network topology, while the second synthetic topology, *prefatt-net*, simulates a preferential attachment network topology. When generating the synthetic networks, we set the parameters of the graph generators so that the formed networks have similar number of vertices and edges to the real network of Epinions (The JUNG[4] tool has been used to generate the networks.). 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 diverse assumptions of network connectivity and stress any interesting differentiation on performance due to network topology. Table 3 summarizes the characteristics of the three networks we consider.

1. Google Web History. http://www.google.com/psearch.
2. Facebook Instant Personalization. http://www.facebook.com/instantpersonalization.
3. Epinions.com is a general consumer review site.
4. JUNG. http://jung.sourceforge.net.

TABLE 3
Network Topologies

Name	Type	Model	Nodes	Edges
epinions-net	Real	Epinions	75888	450740
uniform-net	Synthetic	Uniform	75888	455292
prefatt-net	Synthetic	PrefAttach	75888	455292

TABLE 4
User Search History Logs

Name	Type	Users	Queries	Urls
real_log	Real	75888	4026350	2789866
synth_log	Synthetic	75888	1000000	3000000

User search history logs. We experiment with *real* and *synthetic* user search history logs. For the needs of our experiments we create *real_log* by randomly selecting 75,888 users from the AOL data set along with their search history logs (about 4M queries, 3M urls) [13]. 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. A summary of these logs is provided in Table 4.

Formation of final data sets. Thus far, we have come up with proper network topologies and user search history logs. In order to generate suitable final data sets for our experiments we randomly map each of the 75,888 AOL users in a search history log to the 75,888 nodes of a network topology. Note that since the focus of our work is on performance we do not require that "similar" users are placed in adjacent network nodes. Thus, we do not care for any semantics of the data destroyed, such as the fact that friends may have similar interests and so. The objective of our work is to efficiently collect information in social networks. In doing so, we consider all users in the network to be equal, independently on whether they lie a few or many hops away from the initiator node. In a real setting, one may make use of such semantics to design a better reranking algorithm, but this is orthogonal to the objective of this paper.

5.2 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 or 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

$$RE = \left| \frac{y - \hat{y}}{y} \right|.$$

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 *hops* in the random walks performed to obtain n samples from the network. Formally, the cost of the sampling is

$$Cost = \#HOPS.$$

5.3 Experimental Results

5.3.1 Sampling Accuracy

In Section 3, we have seen that performing random walks by selecting each outgoing edge with equal probability shall pick leaf nodes in a biased manner. This is because some leaves, e.g., leaves that are close to the root, are more likely to be destinations of random walks than other leaves. In our first set of experiments, we explore the effect of this bias in the sampling accuracy and compare the performance of the aforementioned naive sampling method, say *Naive*, to our sampling method, *EvalSingle*. To determine the accuracy performance of the sampling methods, we design experiments that aim to estimate the count of a predetermined url in the network of a user. More specifically, we first determine a target url by submitting a query q to a search engine, such as Google, and obtaining the *top* url. Then, we apply *EvalSingle* or *Naive* to quickly compute an estimate of its count in $D_d(v)$. The process is repeated many times for different queries (typically 100 random queries) and users (typically 100 random users) and at each iteration the relative error of the *estimated url count* to the *exact url count* is computed. We report the average value over all instances. Note that the *exact url count* can be easily obtained by exhaustively crawling the neighborhood of a user, using a breadth-first-search or depth-first-search algorithm, and aggregating the occurences of the targeted url. We experiment for variable network topology and sample size n. Fig. 6 presents the results for network depths $d = 4$ and $d = 5$, respectively. Let us first focus on the behavior of *EvalSingle* alone and then turn to the comparison of the sampling methods. In all topologies *EvalSingle* is able to estimate the targeted values with a very small relative error (always smaller than 15 percent). Furthermore, for a fixed network depth d the sampling accuracy of *EvalSingle* increases with the sample size n (i.e., the average relative error decreases for larger sample sizes) and 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 fixed sample size S the sampling accuracy in the *uniform-net* is better than the one in the *prefatt-net*. This can be explained by the fact that the preferential model has a systematically shorter average path length than the random graph model [14]. 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. Now, going back to the comparison of the two sampling methods we see in Fig. 6 that in all topologies and for all network depths and sample sizes, *EvalSingle* considerably outperforms *Naive*. This suggests huge savings in accuracy. The reason for which *Naive* performs so poor is that in the computations it utilizes more and more of the same sampled users, turning the estimator to be biased toward these sampled users. Remind that sampling is performed with replacement, thus, the same node can be selected in more than one random walks. On the other hand, *EvalSingle* succeeds in selecting samples in a near-uniform way, avoiding to a large extent selection of previously selected nodes. For example, if a sample set of size n is requested, then the number of distinct samples in the sample set formed by *Naive* should typically be much lower than the number of distinct samples in the sample set formed by *EvalSingle*. This effect is demonstrated in Fig. 7, where the number of distinct samples in the sample sets formed by *Naive* and *EvalSingle* is showed, for $d = 4$ and $d = 5$, respectively. Clearly, *EvalSingle* succeeds on considerably alleviating the bias of *Naive* by selecting more distinct samples into the sample sets. This is true for all network topologies tested and for variable sample sizes.

5.3.2 Sampling Cost

EvalSingle performs considerably better in terms of accuracy than a naive sampling method, but as many of the performed random walks end up rejecting a selected leaf node, it can be expensive. In this experiment, we evaluate the cost of our sampling method against the naive sampling method and against the cost of crawling the entire neighborhood of a user. As discussed in Section 3, we use as a surrogate for cost the number of *hops* that the algorithm needs to perform before it ends. Crawling of the entire neighborhood of a user can be performed with either of the depth-first-search or breadth-first-search algorithms. As DFS and BFS have the same performance—linear to the size of the graph—we report only the cost for DFS, which is equal to the number of edges in the spanning tree that has the user as root and extends to a limited depth d. We experiment for variable network topology, network depth d and sample size n. Results are shown in Fig. 8, where a

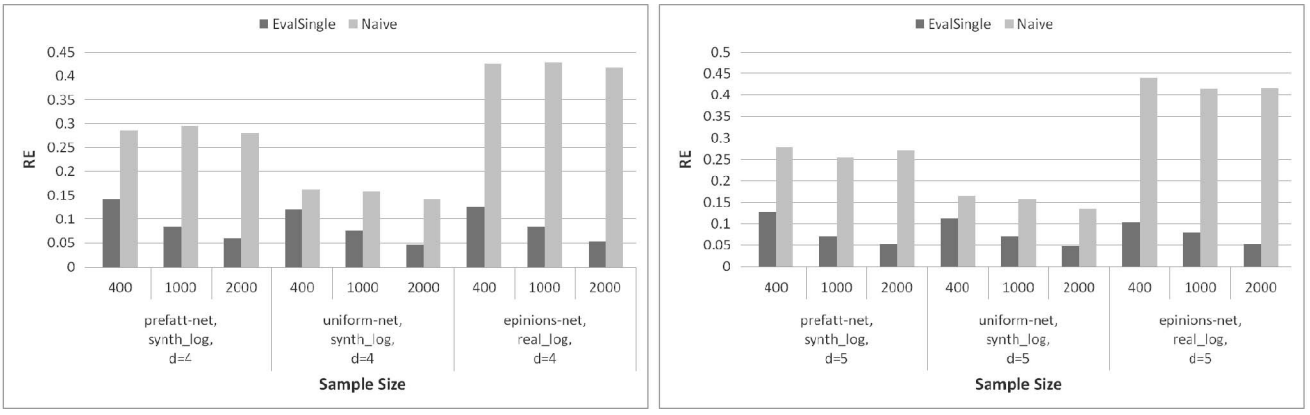

Fig. 6. Sampling Accuracy: EvalSingle versus Naive, RE.

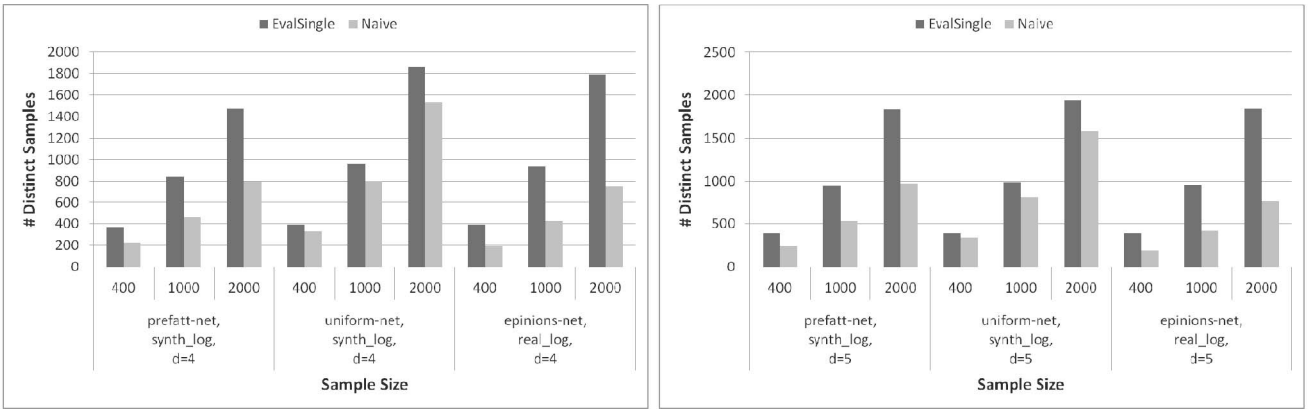

Fig. 7. Sampling Accuracy: EvalSingle versus Naive, # Distinct Samples.

balance of our sampling method between the naive sampling and the exhaustive crawling methods is illustrated. In all topologies and for all network depths and sample sizes, the performance of the naive sampling is better. This is expected, as the naive sampling sacrifices accuracy for efficiency. On the other hand, our sampling method retains high accuracy while at the same time performs considerably better than the exhaustive method in all settings and, as such, suggests huge savings in efficiency. The savings of the sampling method are greater for larger network sizes, where the number of nodes is also larger; savings are greater, as well, when smaller sample sizes are needed, since fewer random walks are required to form the sample. This is true for all network topologies. For the rest of the experimental evaluation we set the parameter $d = 4$. This is a reasonable choice for our research. 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 networks we examine, thus, there is no need to consider $d = 5$ or larger. Between $d = 3$ and $d = 4$ we choose $d = 4$ that will increase the network population and therefore make the approximation problem harder.

5.3.3 Batch Sampling Effect

In Section 4, we discussed how sampling can be used to estimate the popularity (ordering) of a set of items X in the network of a user and introduced the idea of using the same sample for estimating the count for each item (instead of using a fresh sample for each item). This method is much faster as it needs to compute only one sample and computes estimates for all urls at once (batch processing). 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 for both the naive sampling and our sampling method. In particular, we compare the accuracy of *NaiveSingle* to *NaiveBatch* and the accuracy of *EvalSingle* to *EvalBatch*. These algorithms represent the single and the batch way of estimating the ordering of a set of items, using the naive sampling and our sampling method, respectively. Accuracy results reported are average relative errors over a number of runs for random users and queries. At each run a set of top-k urls is determined, where $k = 10$, by submitting a query q to Google and retrieving the top-k results. Then, *EvalSingle*, *EvalBatch*, *NaiveSingle*, and *NaiveBatch* compute the estimates of the url counts. The estimates are then compared to the exact counts to find the relative error of each method. Fig. 9 presents the results of the experiment for the naive sampling and our sampling method, respectively. For each method, we experiment for a synthetic network (*prefatt-net* with *synth_log*) and a real network (*prefatt-net* with *synth_log*) for network depth $d = 4$ and variable sample size n. Results for both synthetic and real network topologies show that *NaiveBatch* has similar behavior to *NaiveSingle* and *EvalBatch* has similar behavior to *EvalSingle*, which indicates that using the same sample for evaluating url counts has a low effect in the sampling accuracy. This small effect is more obvious in the case of the

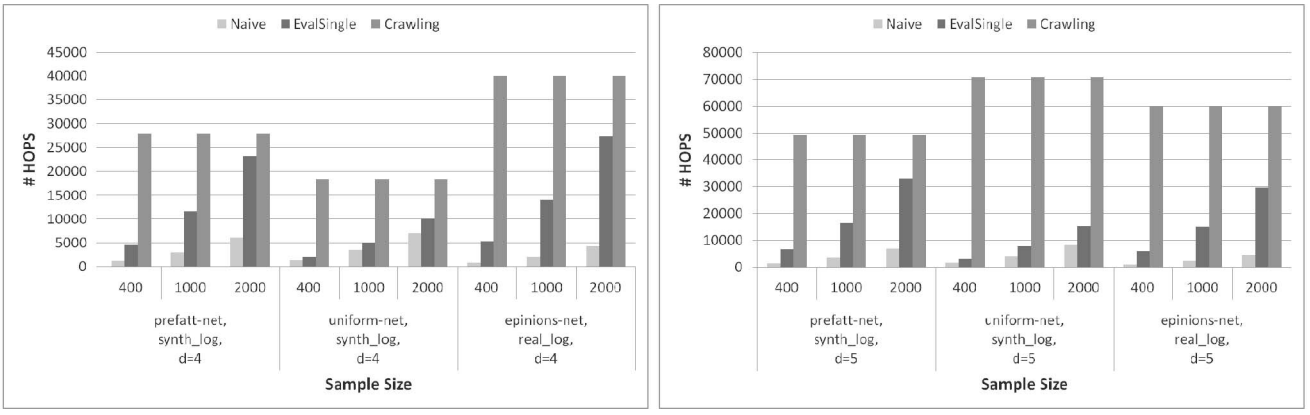

Fig. 8. Sampling cost: Naive versus EvalSingle versus Crawling.

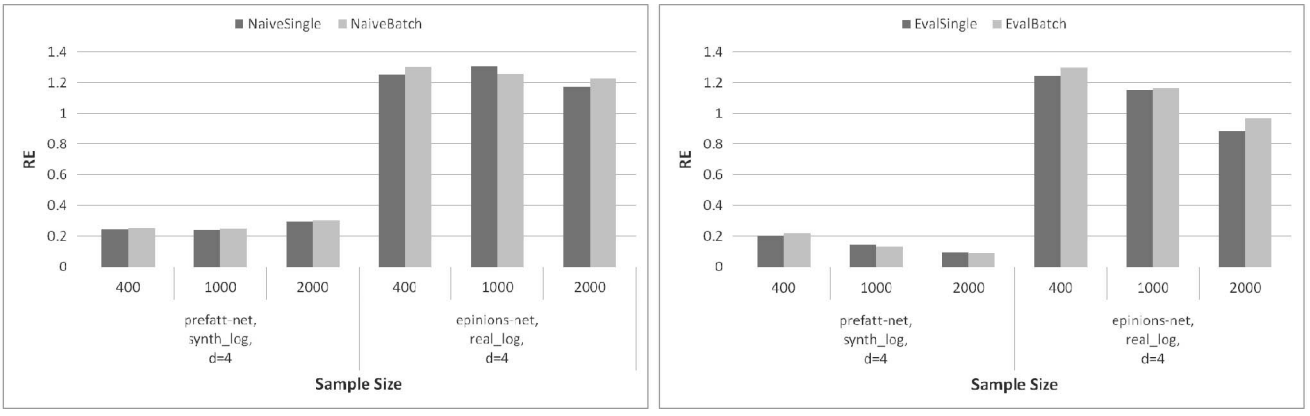

Fig. 9. Batch sampling effect: naive sampling (left), our sampling (right).

real network topology (*epinions-net* with *real_log*). This can be explained by the fact that the real data might exhibit some sort of correlation in the urls (while, in the synthetic topologies, we tried to avoid any data correlation by assigning queries and urls to users randomly). As aforementioned, the sampling accuracy results in the real data are not directly comparable to the results in the synthetic data. This is due to the fact that the url counts in the real data are much smaller and leverage the sampling performance. As a result, even if trends are similar to the ones observed in the synthetic data, larger absolute errors are demonstrated. Note, as well, that in the case of synthetic networks we only present the results for *prefatt-net* and omit the details for *uniform-net*. This is because similar trends were demonstrated over the two synthetic network topologies. 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. From now on, whenever similar trends are demonstrated between the two synthetic network topologies, we only present the results for *prefatt-net* due to space constraints. 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 batch sampling for estimating the counts of many urls at once, which is more efficient. Fig. 9 illustrates as well how *EvalBatch* behaves better than *NaiveBatch*, where as the number of samples increases, *EvalBatch* becomes more accurate; this is no true for *NaiveBatch*. We further investigate the accuracy of *EvalBatch* and *NaiveBatch* in estimating the ordering of items in the next paragraph.

5.3.4 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 the batch 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. For a given user v and network depth d we first compute the ordering r_{q_v} of these urls in the $D_d(v)$ of v according to their exact counts. Then, we apply any of the batch sampling methods (*NaiveBatch* or *EvalBatch*) to estimate the url counts of each $url_q \in r_q$ and come up with an approximate ordering r'_{q_v}. The two lists, r_{q_v} and r'_{q_v}, 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. Fig. 10 illustrates the accuracy performance of *NaiveBatch* and *EvalBatch* in estimating the correct order of the top-k urls, where $k = 10$. For each method we report the Normalized Spearman's Footrule Distance between the exact and the approximate ordering. We make cases of a synthetic network (*prefatt-net* with *synth_log*) and a real network (*epinions* with *real_log*) for $d = 4$ and variable sample size n. In all cases and in both networks, *EvalBatch* ourperfoms

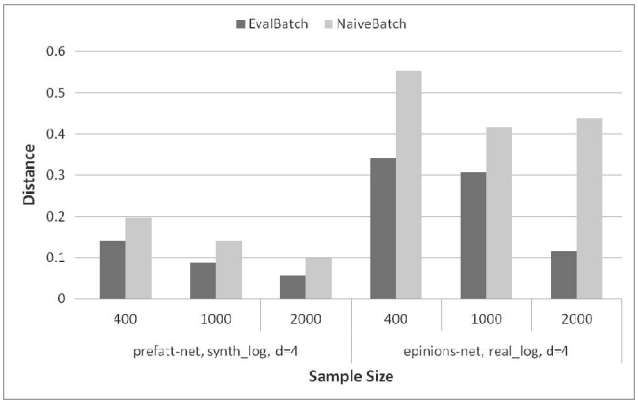

Fig. 10. Ordering accuracy: distance between lists.

NaiveBatch the distance is smaller, signifying a more accurate sampling method. Moreover, as the sample size increases *EvalBatch* the distance decreases, for both network types, which means that our sampling method becomes increasingly more accurate. *NaiveBatch* tends to behave better for larger sample sizes, as well, but this is an artifact of the number of unique users that is able to sample during the sampling process, as such its behavior is not always expected. We further illustrate the accuracy performance of the two methods in correctly estimating the top-k items in the lists. We report on the *Precision at k* (P@K) of the two methods in the case of a synthetic and a real network (See Fig. 11) for $d = 4$ and variable sample size n. In all cases and for both networks, *EvalBatch* ourperfoms *NaiveBatch* as it is able to retain higher precision, which increases as well with the size of the sample. Note, again, that in the case of the real network topology (*epinions-net* with *real_log*) the ordering accuracy results are not directly comparable to the results in the synthetic network. 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 Related Work

Our work is related to work on *sampling large graphs via random walks*. 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 [15], [16], [17], [18]) or the problem of sampling a search engine's index [19], [20]. 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), this probability distribution is *stationary* (i.e., 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 stationary distribution can be made uniform using techniques such as the Metropolis Hastings algorithm (see [21]), or using rejection sampling (where, after reaching a final node, the node is included in the sample with probability inversely proportional to its degree). This process can be repeated to obtain random samples of a desired size. Similar approaches have been employed in [22] where Hastings describes sampling-based methods to efficiently collect information from users in a social graph and in [23] where sampling techniques are used to collect unbiased samples of Facebook. Likewise, in [24] Katzir et al. design algorithms for estimating the number of users in large social networks via biased sampling, and in [25] sampling methods are proposed to approximate community structures in a social network. 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 (i.e., a few links away from v).

Our work is also related to work on *personalized and social search*. The premise of personalized search is that by tailoring search to the individual improved result accuracy may be brought off. A vast amount of literature on search personalization reveals significant improvement over traditional web search. In [26], the CubeSVD approach was developed to improve Web search by taking into account clickthrough data of the type "user, query, url." Further studies showed that taking into account such data and building statistical models for user behavior can significantly improve the result ranking quality [27], [28]. Other approaches exist, as well, that utilize some notion of relevance feedback to rerank web search results [29], [30]. Social search informed by online social networks has actually gained attention as an approach toward personalized search. In a sense, utilizing information from one's social environment to improve on user satisfaction is a form of "extended" personalization, with the extent being defined as a function of the neighborhood of an individual in the network. Many ideas have been suggested to realize online social search; from search engines that utilize humans to filter the search results [31], to systems that utilize real-time temporal correlations of user web history logs [32], [33], to tag-based social search systems [34]. Analyses suggest that integration of social search models improves the overall search experience. Our research is complementary as we aim to offer performance improvements, via sampling, to the process of collecting information from user logs by exploring the graph structure offered by a social network.

7 Conclusions

Our research suggests methods for quickly collecting information from the neighborhood of a user in a dynamic social network when knowledge of its structure is limited or not available. Our methods resort to efficient approximation algorithms based on sampling. By sampling we avoid visiting all nodes in the vicinity of a user and thus attain improved performance. The utility of our approach was demonstrated by running experiments on real and synthetic data sets. Further, we showed that our algorithms are able to efficiently estimate the ordering of a list of items that lie on nodes in a user's network providing support to ranking algorithms and strategies. Despite its competence,

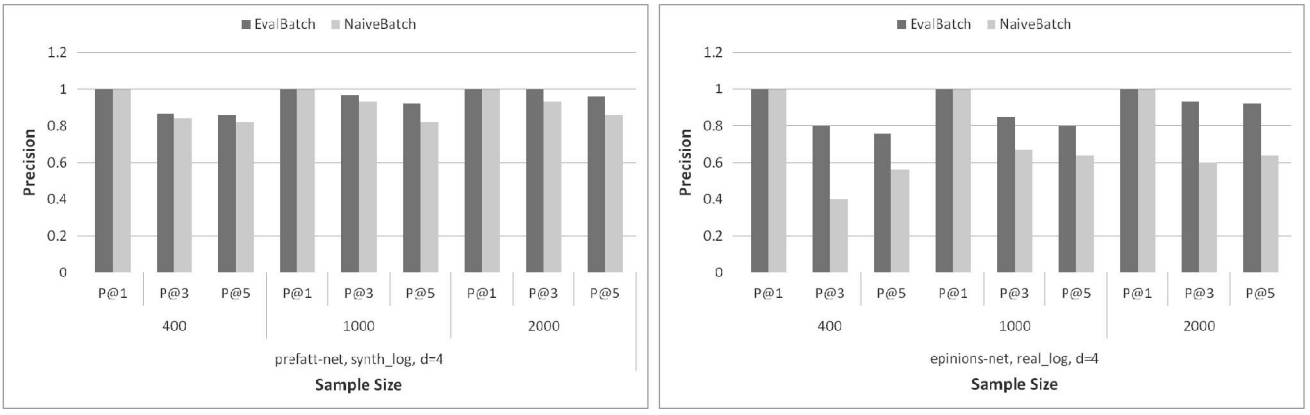

Fig. 11. Ordering accuracy: precision at K for synthetic (left) and real (right) network.

our work inherits limitations of the sampling method itself and is expected to be inefficient for quantities with very low selectivity. A similar problem arises in approximately answering aggregation queries using sampling. Solutions there rely on weighted sampling based on workload information [35]. However, in our context where data stored at each node are rapidly changing this method is not directly applicable. Our algorithms assume that information for each user in a network, such as web history logs, is available. Access to personal information infringes on user privacy and, as such, privacy concerns could serve as a major stumbling block toward acceptance of our algorithms. Systems that utilize our algorithms should adhere to the *social translucence* approach to designing social systems that entail a balance of visibility, awareness of others, and accountability [36].

REFERENCES

[1] A. Mislove, K.P. Gummadi, and P. Druschel, "Exploiting Social Networks for Internet Search," *Proc. Fifth Workshop Hot Topics in Networks (HotNets)*, 2006.

[2] W.G. Cochran, *Sampling Techniques*, third ed. John Wiley, 1977.

[3] D.E. Knuth, "Estimating the Efficiency of Backtrack Programs," *Math. of Computation*, vol. 29, no. 129, pp. 121-136, 1975.

[4] G. Cornujols, M. Karamanov, and Y. Li, "Early Estimates of the Size of Branch-and-Bound Trees," *INFORMS J. Computing*, vol. 18, pp. 86-96, 2006.

[5] P. Kilby, J. Slaney, S. Thiébaux, and T. Walsh, "Estimating Search Tree Size," *Proc. Nat'l Conf. Artificial Intelligence (AAAI)*, 2006.

[6] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman, "Dynamic Restart Policies," *Proc. 18th Nat'l Conf. Artificial Intelligence (AAAI)*, 2002.

[7] L. Katzir, E. Liberty, and O. Somekh, "Estimating Sizes of Social Networks via Biased Sampling," *Proc. 20th Int'l Conf. World Wide Web (WWW)*, 2011.

[8] R.G. Miller, *Simultaneous Statistical Inference*. Springer Verlag, 1981.

[9] M.N. Garofalakis and P.B. Gibbons, "Approximate Query Processing: Taming the Terabytes," *Proc. 27th Int'l Conf. Very Large Data Bases*, Nov. 2001.

[10] S.A.M. Makki and G. Havas, "Distributed Algorithms for Depth-First Search," *Information Processing Letters*, vol. 60, no. 1, pp. 7-12, 1996.

[11] T.-Y. Cheung, "Graph Traversal Techniques and the Maximum Flow Problem in Distributed Computation," *IEEE Trans. Software Eng.*, vol. SE-9, no. 4, pp. 504-512, July 1983.

[12] B. Awerbuch and R.G. Gallager, "A New Distributed Algorithm to Find Breadth First Search Trees," *IEEE Trans. Information Theory*, vol. 33, no. 3, pp. 315-322, May 1987.

[13] C.T.G. Pass and A. Chowdhury, "A Picture of Search," *Proc. First Int'l Conf. Scalable Information Systems (InfoScale)*, 2006.

[14] R. Albert and I. Barabasi, "Statistical Mechanics of Complex Networks," *Modern Physics Rev.*, vol. 74, p. 47, 2002.

[15] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks: Algorithms and Evaluation," *Performance Evaluation*, vol. 63, no. 3, pp. 241-263, 2006.

[16] M. Ajtai, J. Komlos, and E. Szemeredi, "Deterministic Simulation in Logspace," *Proc. 19th Ann. ACM Symp. Theory of Computing (STOC)*, 1987.

[17] R. Impagliazzo and D. Zuckerman, "How to Recycle Random Bits," *Proc. 30th Ann. Symp. Foundations of Computer Science (FOCS)*, 1989.

[18] D. Gillman, "A Chernoff Bound for Random Walks on Expander Graphs," *SIAM J. Computing*, vol. 27, no. 4, pp. 1203-1220, 1998.

[19] Z. Bar-Yossef and M. Gurevich, "Random Sampling from a Search Engine's Index," *Proc. 15th Int'l Conf. World Wide Web (WWW)*, 2006.

[20] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz, "Approximating Aggregate Queries About Web Pages via Random Walks," *Proc. 26th Int'l Conf. Very Large Data Bases (VLDB)*, 2000.

[21] W. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika*, vol. 57, no. 1, pp. 97-109, 1970.

[22] G. Das, N. Koudas, M. Papagelis, and S. Puttaswamy, "Efficient Sampling of Information in Social Networks," *Proc. ACM Workshop Search in Social Media (SSM)*, 2008.

[23] M. Gjoka, M. Kurant, C.T. Butts, and A. Markopoulou, "Walking in Facebook: A Case Study of Unbiased Sampling of Osns," *Proc. INFOCOM*, 2010.

[24] L. Katzir, E. Liberty, and O. Somekh, "Estimating Sizes of Social Networks via Biased Sampling," *Proc. 20th Int'l Conf. World Wide Web (WWW)*, 2011.

[25] A.S. Maiya and T.Y. Berger-Wolf, "Sampling Community Structure," *Proc. 19th Int'l Conf. World Wide Web (WWW)*, 2010.

[26] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen, "Cubesvd: A Novel Approach to Personalized Web Search," *Proc. 14th Int'l Conf. World Wide Web (WWW)*, 2005.

[27] J. Teevan, S.T. Dumais, and E. Horvitz, "Personalizing Search via Automated Analysis of Interests and Activities," *Proc. 28th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, 2005.

[28] E. Agichtein, E. Brill, and S. Dumais, "Improving Web Search Ranking by Incorporating User Behavior Information," *Proc. 29th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, 2006.

[29] Z. Dou, R. Song, and J.-R. Wen, "A Large-Scale Evaluation and Analysis of Personalized Search Strategies," *Proc. 16th Int'l Conf. World Wide Web (WWW)*, 2007.

[30] Q. Wang and H. Jin, "Exploring Online Social Activities for Adaptive Search Personalization," *Proc. 19th ACM Int'l Conf. Information and Knowledge Management (CIKM)*, 2010.

[31] D. Horowitz and S.D. Kamvar, "The Anatomy of a Large-Scale Social Search Engine," *Proc. 19th Int'l Conf. World Wide Web (WWW)*, 2010.

[32] A. Papagelis, M. Papagelis, and C.D. Zaroliagis, "Iclone: Towards Online Social Navigation," *Proc. ACM 19th Conf. Hypertext and Hypermedia (HT)*, 2008.

[33] A. Papagelis, M. Papagelis, and C. Zaroliagis, "Enabling Social Navigation on the Web," *Proc. IEEE/WIC/ACM Int'l Conf. Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2008.
[34] R. Wetzker, C. Zimmermann, C. Bauckhage, and S. Albayrak, "I Tag, You Tag: Translating Tags for Advanced User Models," *Proc. ACM Third Int'l Conf. Web Search and Data Mining (WSDM)*, 2010.
[35] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V.R. Narasayya, "Overcoming Limitations of Sampling for Aggregation Queries," *Proc. 17th Int'l Conf. Data Eng. (ICDE)*, 2001.
[36] T. Erickson and W.A. Kellogg, "Social Translucence: An Approach to Designing Systems that Support Social Processes," *ACM Trans. Computer-Human Interaction*, vol. 7, no. 1, pp. 59-83, 2000.

Manos Papagelis received the BSc and MSc degrees from the University of Crete, in Greece. During the graduate studies, he was also affiliated with the Institute of Computer Science, FORTH, in Greece, working as a research fellow. Currently, he is working toward the PhD degree at the University of Toronto. His research interests include social media, data mining and knowledge discovery, databases, search, recommendation algorithms and trust, and computational social science.

Gautam Das received the PhD degree in computer science from the University of Wisconsin, Madison, in 1990. He is a professor in the Computer Science and Engineering Department, University of Texas, Arlington. Before joining UT-Arlington, he held positions at Microsoft Research, Compaq Corporation, and the University of Memphis. His research interests include data mining and knowledge discovery, databases, algorithms, and computational geometry. His research has been supported by the US National Science Foundation (NSF), US Office of Naval Research (ONR), and industries including Microsoft, Nokia, and Cadence. He is a member of the IEEE.

Nick Koudas received the bachelor's degree from the University of Patras, Greece, the MSc degree from the University of Maryland at College Park, and the PhD degree from the University of Toronto. He is a faculty member in the Computer Science Department at the University of Toronto. He holds more than 20 patents, and has published more than 100 research publications in the areas of database systems, text analytics, and information mining. He serves on the editorial review boards of four scientific journals related to data management systems and data analysis. Before joining U of T, he was a member of AT&T's research laboratories' technical staff. He is a member of the IEEE.