Fast Incremental Proximity Search in Large Graphs

Purnamrita Sarkar PSARKAR@CS.CMU.EDU
Machine Learning Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213

Andrew W. Moore AWM@GOOGLE.COM
Amit Prakash AMITPRAKASH@GOOGLE.COM
Google Inc. Pittsburgh, PA 15213

Abstract

In this paper we investigate two aspects of ranking problems on large graphs. First, we augment the deterministic pruning algorithm in Sarkar and Moore (2007) with sampling techniques to compute approximately correct rankings with high probability under random walk based proximity measures *at query time*. Second, we prove some surprising locality properties of these proximity measures by examining the short term behavior of random walks. The proposed algorithm can answer queries *on the fly without caching any information about the entire graph*. We present empirical results on a $600,000$ node author-word-citation graph from the Citeseer domain on a single CPU machine where the average query processing time is around 4 seconds. We present quantifiable link prediction tasks. On most of them our techniques outperform Personalized Pagerank, a well-known diffusion based proximity measure.

1. Introduction

Link prediction in social networks, personalized graph search techniques, fraud detection and collaborative filtering in recommender networks are important practical problems that greatly rely on graph theoretic measures of similarity. Given a node in a graph we would like to ask which other nodes are most similar to this node. Ideally we would like this similarity measure to capture the graph structure such as having many common neighbors or having several short paths between two nodes. This kind of structural information can be easily quantified using random walks on graphs: diffusion of information from one node to another. Most random-walk based ranking algorithms can be categorized into two broad categories.

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

Probability of reaching a node: This is the basis of measures like *personalized page rank*. Personalized page-rank vectors (PPV) have been used for keyword search in databases (Balmin et al., 2004) and entity-relation graphs (Chakrabarti, 2007). These approaches focus on computing approximate PPV at query time (details in section 6), and quantify the performance in terms of the deviation of the approximation from the exact. However, it is not clear if PPV itself has good predictive power.

Expected number of hops to reach a node: This is also called the hitting time (Aldous & Fill, 2001). The symmetric version of this is the *commute time* between two nodes. These metrics have been shown to be empirically effective for ranking in recommender networks (Brand, 2005) and link prediction problems (Liben-Nowell & Kleinberg, 2003). These measures usually require $O(n^3)$ computation. Recently Spielman and Srivastava (2008) have come up with a novel approximation algorithm for efficiently computing commute times by random projections. However it is only applicable to undirected graphs.

Sarkar and Moore (2007) introduced the notion of truncated commute times and demonstrated that it had good predictive power for link prediction tasks. However their algorithm (GRANCH) required storing potential nearest neighbors of all nodes in the graph in order to answer nearest neighbor queries. The key contribution in this paper are: 1) we combine sampling with deterministic pruning to design an algorithm which retrieves top k neighbors of a query in truncated commute time incrementally without caching information about all nodes in the graph. 2) We investigate locality properties of truncated hitting and commute times. 3) We show that on several link prediction tasks these measures outperform PPV in terms of predictive power, while on others they do comparably. 4) Our algorithm can process queries at around 4 seconds on average on graphs of the order of $600,000$ nodes on a single CPU machine.

The rest of the paper is organized as follows: in section 2 we provide relevant background. In section 3 we introduce our hybrid algorithm, and provide sample complexity results for random sampling. The locality properties of truncated hitting and commute times are investigated in section 4. We present empirical results in section 5, and conclude with related work in section 6.

2. Background

A graph $G = (V, E)$ is defined as a set of vertices V edges E. The ij^{th} entry of the adjacency matrix W denotes the weight on edge (i, j), and is zero if the edge does not exist. $P = p_{ij}, i, j \in V$ denotes the transition probability matrix of this Markov chain, so that $p_{ij} = w_{ij} / \sum_j W_{ij}$ if $(i, j) \in E$ and zero otherwise.

Hitting time h_{ij}: The hitting time from node i to node j is defined as the expected number of steps in a random walk starting from i before node j is visited for the first time. Recursively h_{ij} can be written as $h_{ij} = 1 + \sum_k p_{ik} h_{kj}$, if $i \neq j$ and zero otherwise.

Commute time c_{ij}: Commute time between a pair of nodes is defined as $c_{ij} = h_{ij} + h_{ji}$.

2.1. Truncated Hitting Time

The hitting and commute times are sensitive to long range paths (Liben-Nowell & Kleinberg, 2003) which result in non-local nature. They are also prone to be small if one of the nodes is of large degree (Brand, 2005). This renders them ineffective for personalization purposes. In order to overcome these shortcomings, Sarkar and Moore (2007) define a *T-truncated hitting time*, where only paths of length less than T are considered. We shall use h, h^T interchangeably to denote truncated hitting time. h^T_{ij} can be defined recursively as

$$h^T_{ij} = 1 + \sum_k p_{ik} h^{T-1}_{kj} \tag{1}$$

where h^T is defined to be zero if $i = j$ or if $T = 0$. The above equation expresses h^T in a one step look-ahead fashion. The expected time to reach a destination within T timesteps is equivalent to one step plus the average over the hitting times of it's neighbors to the destination in $T - 1$ hops. If there is no path of length smaller than T from i to j, this automatically sets $h^T(i, j)$ to T.

2.2. GRANCH (Sarkar & Moore, 2007)

The truncated hitting time from all nodes to a destination node can be computed in $O(ET)$ time using dynamic programming. However in order to compute the hitting time from a query node to a destination, one has to compute the hitting time of all nodes to the destination, thus computing the entire matrix which takes $O(NET)$ time (N and E are the number of nodes and edges respectively).

In order to get around the above problem the graph is decomposed into N overlapping neighborhoods for each node. Each neighborhood is computed in a way to include potential nearest neighbors and prune away the rest. The authors provide bounds on the hitting time from all nodes within the neighborhood of i to i. The hitting time from any node outside the boundary to the destination is quantified by only two numbers: a lower and an upper bound. As the neighborhoods are expanded more the bounds become tighter. This way each column of the truncated hitting time (H^T) matrix is filled up **partially**. After iterating over all nodes it is possible to look at one row and obtain ranking from the bounds on hitting time from a node.

GRANCH computes all pairs of nearest neighbors by caching information for all nodes in the graph. This does not work when the graph is changing continuously. We introduce a hybrid algorithm which essentially combines the above branch and bound trick with sampling techniques to obtain nearest neighbors of a query node in commute time with high probability.

3. Hybrid Algorithm

We present an algorithm to compute approximate nearest neighbors in commute times, without iterating over the entire graph. We combine random sampling with the branch and bound pruning scheme mentioned before, in order to obtain upper and lower bounds on *commute times* from a node. This lets us compute the k nearest neighbors from a query node *on the fly*.

For any query node we compute hitting time from it using sampling. We maintain a bounded neighborhood for the query node at a given time-step. We compute estimated bounds on the commute time from the nodes within the neighborhood to the query. Commute time of nodes outside the neighborhood to the query are characterized by a single upper and lower bound. We expand this neighborhood until this lower bound exceeds $2T'$, which guarantees that with high probability we are excluding nodes which are more than $2T'$ commute distance away. These bounds are then used for ranking the nodes inside the neighborhood.

We will first describe a simple sampling scheme to obtain ϵ-approximate truncated hitting times **from** a query node with high probability.

3.1. Sampling Scheme

We propose a sampling scheme to estimate the truncated hitting time from a given query node i in a graph. We run M independent T-length random walks from i. Lets say out of these M runs m random walks hit j for the first time at $\{t_{k_1}, ...t_{k_m}\}$ time-steps. From these we can estimate the following

1. The probability of hitting any node j *for the first time* from the given source node within T steps can be estimated by $\frac{m}{M}$.
2. The first hitting time can be estimated by

$$\hat{h}^T(i,j) = \frac{\sum_r t_{k_r}}{M} + (1 - \frac{m}{M})T$$

We provide bounds (details in Appendix) similar to Fogaras et al. (2004)

1. The number of samples M required in order to give an ϵ- correct answer with probability $1 -$.
2. The number of samples M required in order to get the top k neighbors correct.

Theorem 3.1 *For a given node i, in order to obtain $P(\exists u \in \{1, \ldots, n\}, |\hat{h}^T(i,u) - h^T(i,u)| \geq \epsilon T) \leq$, number of samples M should be at least $\frac{1}{2\epsilon^2}\log(\frac{2n}{\delta})$.*

Theorem 3.2 *Let $v_j, j = 1 : k$ be the top k neighbors of i in exact T-truncated hitting time. Let $\alpha = h^T(i, v_{k+1}) - h^T(i, v_k)$. Then number of samples M should be at least $\frac{2T^2}{\alpha^2}\log(nk/\ )$ in order to have $Pr(\exists j \leq k, q > k, \hat{h}^T(i, v_j) > \hat{h}^T(i, v_q)) \leq$.*

The details are provided in the appendix. The above theorem says nothing about the order of the top k neighbors, only that if the gap between the hitting times from i to the k^{th} and $k+1^{th}$ nearest neighbors is large, then it is easy to retrieve the top k nearest neighbors. We could change the statement slightly to obtain a sample complexity bound to guarantee the exact order of the top k neighbors with high probability. The main difference will be that it will depend on $\min_{j \leq k} h^T(i, v_{j+1}) - h^T(i, v_j)$.

3.2. Lower and Upper Bounds on c^T_{ij}

Let us denote the neighborhood of node j by $NBS(j)$. The boundary of this is denoted by (j).In Eqn (1) $h^t(i,j)$ is computed using the hitting time from its direct neighbors to j, which are computed in the $t - 1^{th}$ iteration. Since only the hitting times of nodes within $NBS(j)$ are stored, a boundary node would not have access to the hitting time of at least one of its neighbors. Those values can be upper and lower bounded as follows. The fastest possible way to reach node j from any node outside $NBS(j)$ would be by jumping to the node on the boundary (j) which has the closest optimistic hitting time to j. This gives us a lower bound on the hitting time of all nodes outside $NBS(j)$ to j.

$$lb(j) = 1 + \min_{p \in \delta(j)} ho^{T-1}_{pj} \tag{2}$$

The pessimistic bound is T. Plugging in these bounds in equation (1) whenever the neighbors are outside the neighborhood of the destination gives the expressions for optimistic (ho^T_{ij}) and pessimistic (hp^T_{ij}) bounds on hitting times (details in Sarkar and Moore (2007)).

Now we have the expressions for the lower and upper bounds for the hitting times of the nodes in $NBS(j)$ to j (ho and hp values). The hitting time from j to nodes within $NBS(j)$ can be estimated using the sampling scheme described in section 3.1. Combining the two leads to the following.

Theorem 3.3 *The truncated commute time between nodes $i \in NBS(j)$ and j will be lower and upper bounded by co^T_{ij} and cp^T_{ij} with probability $1 -$ if the number of samples for estimating $\hat{h}^T_{ij}$ exceeds the lower bound in theorem 3.1, where*

$$co^T_{ij} = \hat{h}^T_{ji} + ho^T_{ij} - \epsilon T \tag{3}$$

$$cp^T_{ij} = \hat{h}^T_{ji} + hp^T_{ij} + \epsilon T \tag{4}$$

We would use $\widehat{co}_{ij} = \hat{h}^T_{ji} + ho^T_{ij}$ and similarly $\widehat{cp}_{ij}$ to denote estimates of these bounds. In order to prune away nodes which are not potential nearest neighbors we also need to obtain a lower bound on the commute time between j and any node outside $NBS(j)$. The incoming lower bound is given by equation 2. Now note that for the outgoing lower bound we need the minimum of $h^T_{jk}, \forall k \notin NBS(j)$.

Lemma 3.4 *The number of samples M should be at least $\frac{1}{2\epsilon^2}\log(\frac{2n}{\delta})$ in order to obtain $Pr(|\min_{k \notin NBS(j)} \hat{h}^T_{jk} - \min_{k \notin NBS(j)} h^T_{jk}| \geq \epsilon T) \leq 2$.*

Thus an estimate of the outgoing lower bound can be computed from the hitting times obtained from sampling. Combining the two we obtain an estimate on the lower bound on $2T$-truncated commute time $\widehat{lb\text{-}ct}(j)$ from j to any node outside $NBS(j)$.

$$\widehat{lb\text{-}ct}(j) = 1 + \min_{p \in \delta(j)} ho^{T-1}_{pj} + \min_{k \notin NBS(j)} \hat{h}^T_{jk} \tag{5}$$

For our implementation, we always used estimated, not the exact bounds. This introduces an additive error in our results (proof excluded for lack of space).

3.3. Expanding Neighborhood

Now we need to find a heuristic to expand the neighborhood such that both the outgoing and incoming components of the lower bound increase quickly so that the threshold of $2T'$ is reached soon.

For the incoming lower bound we just find the x closest nodes on the boundary which have small optimistic hitting time to the query. We add the neighbors of these nodes to the neighborhood of j. For the outgoing lower bound, we compute the nodes outside the boundary which a random walk is most probable to hit. We do this by maintaining a set of paths from j which stop at the boundary. These paths are augmented one step at a time. This enables one step lookahead in order to figure out which nodes outside the boundary are the most probable nodes to be hit. We add y of these nodes to the current boundary.

3.3.1. Ranking

The ranking scheme is similar to GRANCH and is rather intuitive. So far we only have lower and upper bounds for commute times from node j to the nodes in $NBS(j)$. Lets denote this set as S. The commute time from j to any node outside S is guaranteed to be bigger than $2T'$. The true k^{th} nearest neighbor will have commute time larger than the k^{th} smallest lower bound i.e. *co* value. Lets denote the k^{th} smallest *co* value by X. Now consider the nodes which have upper bounds (*cp* values) smaller than X. These are guaranteed to have commute time smaller than the true k^{th} nearest neighbor. Adding a multiplicative slack of α to X allows one to return the α-approximate k nearest neighbors which have commute time within $2T'$. Note that the fact that no node outside set S has hitting time smaller than $2T'$ is crucial for ranking, since that guarantees the *true* k^{th} *nearest neighbor* within $2T'$ commute distance to be within S. Since all our bounds are probabilistic, i.e. are true with high probability (because of the sampling), we return α-approximate k nearest neighbors with high probability. Also the use of estimated bounds ($\widehat{co}$,$\widehat{cp}$) will introduce an additive error of $2\epsilon T$ (ignoring a small factor of $\epsilon\alpha T$).

3.4. The Algorithm at a Glance

In this section we describe how to use the results in the last subsections to compute nearest neighbors in truncated commute time from a node. *Given T, α, k our goal is to return the top k α-approximate nearest neighbors (within $2\epsilon T$ additive error) w.h.p.*

First we compute the outgoing hitting times from a node using sampling. We initialize the neighborhood with the direct neighbors of the query node (We have set up our graph so that there are links in both directions of an edge, only the weights are different). At any stage of the algorithm we maintain a bounded neighborhood for the query node. For each node inside the neighborhood the hitting times *to* the query can be bounded using dynamic programming. Combining these with the sampled hitting times gives us the estimated $\widehat{co}$, and $\widehat{cp}$ values. We also keep track of the lower bound $\widehat{lb\text{-}ct}$ of the commute time from any node outside the neighborhood to the query node. At each step we expand the neighborhood using the heuristic in section 3.3. Similar to GRANCH we recompute the bounds again, and keep expanding until $\widehat{lb\text{-}ct}$ exceeds $2T'$. W.h.p this guarantees that all nodes outside the neighborhood have commute time larger than $2T' - \epsilon T$.

Then we use the ranking as in section 3.3.1 to obtain k α-approximate nearest neighbors (with an additive slack of $2\epsilon T$) in commute time. We start with a small value of T' and increase it until all k neighbors can be returned. As in Sarkar and Moore (2007) it is easy to observe that the lower bound can only increase, and hence at some point it will exceed $2T'$ and the algorithm will stop. The question is how many nodes can be within $2T'$ commute distance from the query node. In section 4 we prove that this quantity is not too large for most query nodes.

4. Locality Properties of h^T

In this section we analyze the locality properties of truncated hitting times. We show that *most* nodes in a graph will have only a *small* number of neighbors within $2T'$ T-truncated commute time. We would do this in three steps. First we show that number of nodes within hitting time T' **from** a node i is small. Then we would make a similar argument that the number of nodes within T'- hitting distance **to** i is also small. Finally we would make an argument about the neighbors of i in commute time.

Theorem 4.1 *For any graph G and constants T and T', the number of nodes within a truncated hitting distance of T' from any node is at most $T^2/(T - T')$.*

Let $P^{<T}_{ij}$ denote the probability of hitting node j starting at i within T steps and $\tilde{P}^t_{ij}$ the probability of hitting j in exactly t steps for the first time from i.

$$T' \geq h_{ij} \geq T(1 - P^{<T}_{ij}) \implies P^{<T}_{ij} \geq \frac{T - T'}{T} \qquad (6)$$

Define S_i as the neighborhood of i which consists of only the nodes within hitting time T' **from** i.

$$\sum_{j \in S_i} P^{<T}_{ij} = \sum_{j \in S_i} \sum_{t=1}^{T-1} \tilde{P}^t_{ij} \leq \sum_{t=1}^{T-1} \sum_{j \in S_i} P^t_{ij} \leq T - 1$$

However the left hand side is lower bounded by $|S_i| \frac{T-T'}{T}$ using (6). Which leads us to the upper bound

$|S_i| \leq \frac{T^2}{T-T'}$. So all total there are only $N\frac{T^2}{T-T'}$ pairs within T' hitting distance. If T and T' are constant w.r.t n, then using the above bound and counting arguments we can also show that there can be at most $O(\sqrt{n})$ nodes with more than $\sqrt{n}$ nodes within T' hitting time *to them.*

We have shown that not more than $O(\sqrt{n})$ nodes can have more than $O(\sqrt{n})$ nodes with hitting time smaller than T' **to** them. We already have a bound of $T^2/(T-T')$ on the number of nodes with hitting time smaller than T' **from** a node. We want to bound the number of nodes within commute time $2T'$.

In other words we have proven that $\{j|h^T_{ij} \leq T'\}$ and $\{j|h^T_{ji} \leq T'\}$ are small. Now we need to prove that $\{j|h^T_{ij} + h^T_{ji} \leq 2T'\}$ is also small. Note that the above set consists of

1. $S_1 = \{j|h^T_{ij} \leq T' \bigcap h^T_{ij} + h^T_{ji} \leq 2T'\}$
2. $S_2 = \{j|h^T_{ij} > T' \bigcap h^T_{ij} + h^T_{ji} \leq 2T'\}$

S_1 can have at most $T^2/(T-T')$ nodes. Now consider S_2. S_2 will have size smaller than $|\{j|h^T_{ji} \leq T'\}|$. Using our result from before we can say the following

Lemma 4.2 *Let T be constant w.r.t n. If T' is bounded away from T by a* ***constant*** *w.r.t n, i.e. $T^2/(T-T')$ is constant w.r.t n, then not more than $O(\sqrt{n})$ nodes will have more than $O(\sqrt{n})$ neighbors with truncated commute time smaller than $2T'$.*

The impact of lemma 4.2 is that in a sequence of $O(\sqrt{n})$ nearest neighbor queries (each selected at random), for each node, there would be at most $O(\sqrt{n})$ other nodes within $2T'$ commute distance on average.

5. Empirical Results

We have examined our algorithm on Entity Relation (ER) datasets extracted from the Citeseer corpus, as in Chakrabarti (2007). This is a graph of authors, papers and title-words extracted from Citeseer.

5.1. Dataset and Link Structure

The link structure is obvious:

1. Between a paper and a word appearing in its title.
2. From a paper to the paper it cites, and one with one-tenth the strength the other way.
3. Between a paper and each of its authors.

As observed by Chakrabarti (2007), the weights on these links are of crucial importance. Unlike some other approaches (Balmin et al., 2004; Chakrabarti, 2007) we also put links *from* the paper layer *to* the word layer. This allows flow of information from one paper to another via the common words they use. The links between an author and a paper are undirected. The links within the paper layer are directed. We use the convention in Chakrabarti (2007) to put a directed edge from the cited paper to the citing paper with one-tenth the strength.

For any paper we assign a total weight of W to the words in its title, a total weight of P to the papers it cites and A to the authors on it. We use an inverse frequency scheme for the paper-to-word link weight, i.e. the weight on link from paper p to word w is $W \times 1/f_w/(\sum_{p\to u} 1/f_u)$, where f_w is the number of times word w has appeared in the dataset. We set $W = 1, A = 10, P = 10$ so that the word layer to paper layer connection is almost directed. We add a self loop to the leaf nodes, with the same weight as its single edge, so that the hitting times from these leaf nodes are not very small.

We use two subgraphs of Citeseer. The small one has around $75,000$ nodes and $260,000$ edges: $16,445$ words, $28,719$ papers and $29,713$ authors. The big one has around $600,000$ nodes with 3 million edges : $81,664$ words, $372,495$ papers and $173,971$ authors.

5.2. Preprocessing

We remove the stopwords and all words which appear in more than 1000 papers from both the datasets. The number of such words was around 30 in the smaller dataset and 360 in the larger one. We will make the exact dataset used available on the web.

5.3. Experiments

The tasks we consider are as follows,

1. Paper prediction for words (Word task): We pick a paper X at random, remove the links between it and its title words. Given a query of exactly those words we rank the papers in the training graph. For different values of y the algorithm has a score of 1 if X appears in the closest y papers. *For any search engine, it is most desirable that the paper appears in the top k results, $k \leq 10$.*
2. Paper prediction for authors (Author task): Exactly the above, only the link between the paper and its authors are removed.

The hybrid algorithm is compared with: 1) Exact truncated hitting time from the query, 2) Sampled truncated hitting time from the query, 3) Exact truncated commute time from the query, 4) Exact truncated hitting time to the query, 5) Personalized Pagerank Vector and 6) Random predictor. Note that we can compute a high accuracy estimate of the *exact* hitting time *from* a query node by using a huge number of samples. We can also compute the exact hitting time *to* a node by using dynamic programming by iterating over all nodes. Both of these will be slower than the sampling or the hybrid algorithm as in Table 1.

Distance from a set of nodes Hitting and commute times are classic measures of proximity from a single node. We extend these definitions in a very simple fashion in order to find near-neighbors of a set of nodes. The necessity of this is clear, since a query often consists of more than one word. We define the hitting time from a set of nodes as an weighted average of the hitting times from the single nodes. For hitting time to a set, we can change the stopping condition of a random walk to "stop when it hits any of the nodes in the set". We achieve this via a simple scheme: for any query q, we *merge* the nodes in the query in a new mega node Q as follows. For any node $v \notin Q$ $P(Q, v) = \sum_{q \in Q} w(q)P(q, v)$, where $P(q, v)$ is the probability of transitioning to node v from node q in the original graph. $\sum_{q \in Q} w(q) = 1$. We use a uniform weighing function, i.e. $w(q) = 1/|Q|$. The hitting/commute time is computed on this modified graph from Q. These modifications to the graph are local and can be done at query time, and then we undo the changes for the next query.

Our **average query size** is the average number of words (authors) per paper for the word (author) task. These numbers are around 3 (2) for the big subgraph, and 5 (2) for the small subgraph.

Figure 1 has the performance of all the algorithms for the author task on the (A) smaller, (B) larger dataset and the word task on the (C) smaller and (D) larger dataset, and Table 1 has the average runtime. As mentioned before for any paper in the testset, we remove all the edges between it and the words (authors) and then use the different algorithms to rank all papers within 3 hops of the words (5 for authors, since authors have smaller degree and we want to have a large enough candidate set) in the new graph and the removed paper. For any algorithm the percentage of papers in the testset which get ranked within the top k neighbors of the query nodes is plotted on the y axis vs. k on the x axis. We plot the performances for six values of k: $1, 3, 5, 10, 20$ and 40.

The results are extremely interesting. Before going into much detail let us examine the performance of the exact algorithms. Note that for the author task the *exact hitting time* ***to a node*** and the *exact commute time from a node* consistently beats the *exact hitting time* ***from a node***, and PPV. However for the word task the outcome of our experiments are **the opposite**. This can be explained in terms of the inherent directionality of a task. The distance *from* the word-layer to the paper-layer gives more information than the distance from the paper layer to the word layer, whereas both directions are important for the author task, which is why commute times, and hitting time to a node outperform all other tasks.

We only compare the predictive power of PPV with our measures, not the *runtime*. Hence we use the exact version of it. We used $c = 0.1$, so that the average path-length for PPV is around 10, since we use $T = 10$ for all our algorithms (however, much longer paths can be used for the exact version of PPV). PPV and hitting time from a node essentially relies on the probability of reaching the destination from the source. Even though hitting time uses information only from a truncated path, in all of our experiments it performs better than PPV, save one, where it behaves comparably.

Word task: The sampling based hitting time beats PPV consistently by a small margin on the bigger dataset, whereas it performs comparably on the smaller one. Hitting times and PPV beat the hitting time to nodes for the word task. In fact for $k = 1, 3, 5$, for the smaller dataset the hitting time *to* a query node isn't much of an improvement over the random predictor (which is zero). This emphasizes our claim that the hitting time *to* the word layer does not provide significant information for the word task. As a result the performance of the exact and hybrid commute times deteriorates.
Author task: The hitting time *to the query* and the exact commute time *from a query* have the best performance by a large margin. The hybrid algorithm has almost similar performance. Hitting time from the query is beaten by these. PPV does worse than all the algorithms except of course the random predictor.

Number of samples: For the small graph, we use $100,000$ samples for computing the high accuracy approximation of hitting time from a node; 5000 samples for the word task and 1000 samples for the author task. We use 1.5 times each for the larger graph. We will like to point out that our derived sample complexity bounds are interesting for their asymptotic behavior. In practice we expect much fewer samples to achieve low probability of error. In Figure 1 sometimes the exact hitting (commute) times does worse than the sampled hitting time (hybrid algorithm). This might happen by chance, with a small probability.

6. Related Work

In this section we briefly examine algorithms which have been developed using random walks on graphs, and their applications. Brand (2005) uses different random walk based measures to compute the top k rec-

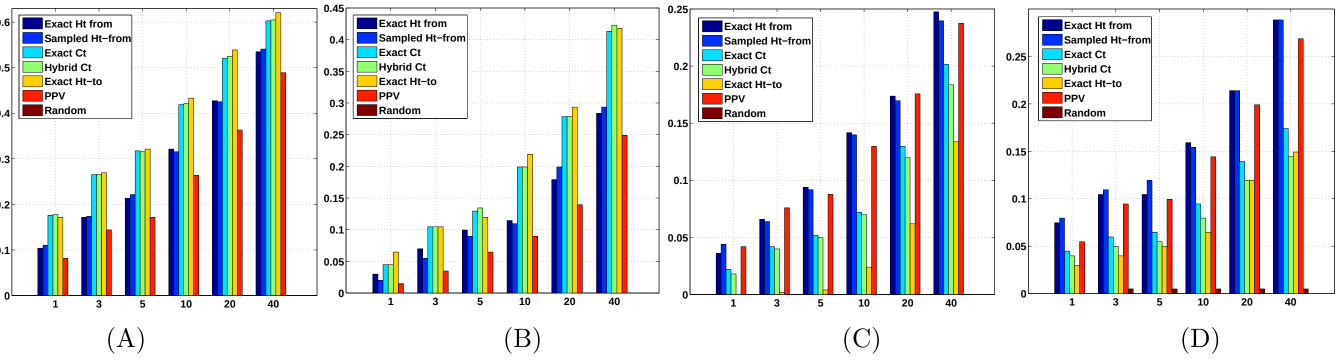

Figure 1. Author task for (A) Small and (B) Large datasets. Word task for (C) Small and (D) Large datasets. x-axis denotes the number of neighbors and y-axis denotes accuracy.

Table 1. Run-time in seconds for Exact hitting time from query, Sampled hitting time from query, Exact commute time, Hybrid commute time, Exact hitting time to query, PPV

# nodes	# edges	Task	Exact Ht-from	Sampled Ht-from	Exact Ct	Hybrid Ct	Exact Ht-to	PPV
74,877	259,320	Author	1.8	**.02**	9.2	**.28**	6.7	18
		Word	3.1	**0.3**	10.4	**1.2**	6.56	50
628, 130	2, 865, 660	Author	6.9	**.07**	79.07	**1.8**	67.2	337.5
		Word	12.3	**0.7**	88.0	**4.3**	70	486

ommendations for a particular customer in a customer-movie graph from the movielens dataset. The submatrices of the hitting and commute times matrices are computed by iterative sparse matrix multiplications (details in Sarkar and Moore (2007)). However it is only tractable to compute these measures on graphs with a few thousand nodes for most purposes.

Liben-Nowell and Kleinberg (2003) showed that the hitting and commute times perform poorly for link prediction tasks, because of their sensitivity to long paths. The most effective measure was shown to be the Katz measure (Katz, 1953) which directly sums over the collection of paths between two nodes with exponentially decaying weights. However, ranking under the Katz score would require solving for a row of the matrix $(I-\gamma A)^{-1}-I$, where I and A are the identity and adjacency matrices of the graph and γ is the decay factor. Even if A is sparse the fast linear solvers will take at least $O(E)$ time.

Tong et al. (2007) uses escape probability from node i to node j to compute *direction aware proximity* in graphs. A fast matrix solver is used to compute this between *one pair of nodes*, in $O(E)$ time. Multiple pairs of proximity require computation and storage of the inverse of the matrix $I-\gamma P$, which would be intractable for large graphs ($10K$ nodes). Jeh and Widom (2002b) use the notion of *expected f-hitting distance*, which is the hitting time (in a random walk with restart) between a set of nodes in a product graph with N^2 nodes. The quadratic time complexity is reduced by limiting the computation between source and destination pairs within distance of r.

The main idea of *personalized pagerank* is to bias the probability distribution towards a set of webpages particular to a certain user, resulting in a user-specific view of the web. It has been proven (Jeh & Widom, 2002a) that the PPV for a set of webpages can be computed by linearly combining those for each individual webpage. However it is hard to store all possible personalization vectors or compute the personalized pagerank vector at query time because of the sheer size of the internet graph. There have been many novel algorithms for efficiently computing PPV (Jeh & Widom, 2002a; Haveliwala, 2002; Fogaras et al., 2004). Most of these algorithms compute partial PPVs offline and combine them at query time.

The *ObjectRank* algorithm (Balmin et al., 2004) computes keyword-specific ranking in a publication database of authors and papers, where papers are connected via citations and co-authors. The personalized pagerank for each word is computed and stored offline, and at query time combined linearly. Chakrabarti

(2007) et al. show how to compute approximate personalized pagerank vectors using clever neighborhood expansion schemes which would drastically reduce the amount of offline storage and computation.

7. Conclusion

Many graph-based learning algorithms rely on computing proximity measures in graphs. These graphs can be very large and undergoing continuous change, hence fast *incremental* algorithms are needed. In this paper we have combined sampling techniques with branch and bound pruning to compute near neighbors of a query node with high probability. Our proximity measures have been empirically shown to often outperform a popular alternative, namely personalized pagerank on link-prediction tasks.

Acknowledgements

Thanks to Soumen Chakrabarti for sharing his code and data, Tamás Sarlós and Martin Zinkevich for helpful discussions and Geoffrey Gordon and Anupam Gupta for valuable suggestions.

Appendix

Proof of Theorem 3.1: We provide a bound on the number of samples M required in order to give an ϵ-correct answer with probability $1-\delta$. We denote the estimate of a random variable x by $\hat{x}$ from now on. Lets denote by $X^r(i,u)$ the first arrival time at node u from node i on the r^{th} trial. Define $X^r(i,u) = T$ if the path does not hit u on trial r. Note that $\hat{h}^T(i,u) = \sum_r X^r(i,u)/M$, and $E[\hat{h}^T(i,u)] = h^T(i,u)$. $\{X^r(i,u) \in [1,T], r = 1 : M\}$ are i.i.d. random variables. The Hoeffding bound gives

$$\begin{aligned}&P(|\hat{h}^T(i,u) - h^T(i,u)| \geq \epsilon T) \leq 2\exp(-\tfrac{2M(\epsilon T)^2}{T^2})\\ &= 2\exp(-2M\epsilon^2)\end{aligned}$$

Now we want the probability of a bad estimate for any u to be low. We upper bound this error probability using union and Hoeffding bounds and set the upper bound to be less than a small value δ. Hence we have $P(\exists u \in \{1,\ldots,n\}, |\hat{h}^T(i,u) - h^T(i,u)| \geq \epsilon T) \leq 2n\exp(-2M\epsilon^2) \leq \delta$. This gives the lower bound of $\frac{1}{2\epsilon^2}\log(\frac{2n}{\delta})$.

Proof of Theorem 3.2: Consider a sampled path of length T starting from i. We define $X^r(i,u)$ as before. For two arbitrary nodes u and v, WLOG let $h^T(i,u) > h^T(i,v)$. The idea is to define a random variable whose expected value will equal $h^T(i,u) - h^T(i,v)$. We define a random variable $Z^r = X^r(i,u) - X^r(i,v)$. $\{Z^r \in [-(T-1), T-1], r = 1 : M\}$ are i.i.d. random variables. Note that $E(Z^r) = h^T(i,u) - h^T(i,v)$.

The probability that the ranking of u and v will be exchanged in the estimated h^T values from M samples equals $P(\hat{h}^T(i,u) < \hat{h}^T(i,v))$. This probability equals $P(\sum_{r=1}^M Z_r/M < 0)$ which using the Hoeffding bound is smaller than $\exp(-2M(h^T(i,u) - h^T(i,v))^2/(2T)^2) = \exp(-M(h^T(i,u) - h^T(i,v))^2/2T^2)$. Let $v_1, v_2, \ldots, v_k$ be the top k neighbors of i in exact truncated hitting time.

$$\begin{aligned}&Pr(\exists j \leq k, q > k, \hat{h}^T(i,v_j) > \hat{h}^T(i,v_q))\\ &\leq \sum_{j\leq k}\sum_{q>k} Pr(\hat{h}^T(i,v_j) > \hat{h}^T(i,v_q))\\ &\leq \sum_{j\leq k}\sum_{q>k}\exp(-\tfrac{M(h^T(i,v_q)-h^T(i,v_j))^2}{2T^2})\\ &\leq nk\exp(-\tfrac{M(h^T(i,v_{k+1})-h^T(i,v_k))^2}{2T^2})\end{aligned}$$

Let $\alpha = h^T(i,v_{k+1}) - h^T(i,v_k)$. Setting the above probability to be less than δ gives us the desired lower bound of $\frac{2T^2}{\alpha^2}\log(nk/\delta)$ on M.

Proof of lemma 3.4: Let S be a set of nodes. Let $q = \arg\min_{k\in S} h(j,k)$. Let $m = \arg\min_{k\in S} \hat{h}(j,k)$. We know that $h(j,q) \leq h(j,m)$, since q is the true minimum, and $\hat{h}(j,m) \leq \hat{h}(j,q)$, since m is the node which has the minimum estimated h-value. Using the sample complexity bounds from theorem 3.1, we have $\hat{h}(j,m) \leq \hat{h}(j,q) \leq^{w.h.p} h(j,q) + \epsilon T$. For the other part of the inequality we have $\hat{h}(j,m) \geq^{w.h.p} h(j,m) - \epsilon T \geq h(j,q) - \epsilon T$. Using both sides we get $h(j,q) - \epsilon T \leq^{w.h.p} \hat{h}(j,m) \leq^{w.h.p} h(j,q) + \epsilon T$. Using S to be the set of nodes outside the neighborhood of j yields lemma 3.4.

References

Aldous, D., & Fill, J. A. (2001). *Reversible markov chains.*

Balmin, A., Hristidis, V., & Papakonstantinou, Y. (2004). ObjectRank: Authority-based keyword search in databases. *VLDB, 2004.*

Brand, M. (2005). A Random Walks Perspective on Maximizing Satisfaction and Profit. *SIAM '05.*

Chakrabarti, S. (2007). Dynamic personalized pagerank in entity-relation graphs. *WWW '07* (pp. 571–580). New York, NY, USA: ACM Press.

Fogaras, D., Rácz, B., Csalogány, K., & Sarlós, T. (2004). Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. .

Haveliwala, T. (2002). Topic-sensitive pagerank. *WWW.*

Jeh, G., & Widom, J. (2002a). Scaling personalized web search. *Stanford University Technical Report.*

Jeh, G., & Widom, J. (2002b). Simrank: A measure if structural-context similarity. *KDD.*

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika.*

Liben-Nowell, D., & Kleinberg, J. (2003). The link prediction problem for social networks. *CIKM '03.*

Sarkar, P., & Moore, A. (2007). A tractable approach to finding closest truncated-commute-time neighbors in large graphs. *Proc. UAI.*

Spielman, D., & Srivastava, N. (2008). Graph sparsification by effective resistances. *Proceedings of the STOC'08.*

Tong, H., Koren, Y., & Faloutsos, C. (2007). Fast direction-aware proximity for graph mining. *Proc. KDD.*