Learning on Large Graphs: Tractable Models and Algorithms

Purnamrita Sarkar

Machine Learning Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The main focus of this proposal is on understanding and analyzing entity relationships in large social networks. The broad range of applications of graph based learning problems includes collaborative filtering in recommender networks, link prediction (e.g. predicting future links from the current snapshot of a graph) in social networks, fraud detection and personalized graph search techniques. In all these real world problems the main question is: given a node which other nodes are close to it?

We use short term random walks to develop fast algorithms to compute approximate nearest neighbors in large graphs. In previous work we have combined sampling with deterministic pruning to retrieve top k neighbors of a query in truncated hitting and commute time incrementally without caching information about all nodes in the graph. Our algorithms can answer nearest neighbor queries on graphs of size $600,000$ nodes in 3 seconds on average. We have shown that on several link prediction tasks on real world datasets these measures outperform other popular proximity measures (Jaccard, Adamic-Adar, Katz, PPV) in terms on predictive power.

In this thesis we propose to analyze the short-term behavior of random walks to improve upon our algorithm, investigate other graph-based proximity measures, and apply these algorithms on real-world tasks like semi-supervised learning, information retrieval, recommender networks, and a meta-learning approach for link prediction.

Contents

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 proximity measure to capture the graph structure such as having many common neighbors or having several short paths between two nodes. These graphs are huge and continuously evolving over time. Therefore the need for fast incremental algorithms for proximity search is clear. This is the main focus of this thesis.

There are mainly two approaches to compute proximity measures. Since most real-world graphs are large and sparse, it makes sense to assume that the entities belong to a low dimensional Euclidian space, where nearby entities are also close in the graph. Spectral clustering (Ng et al., 2001) learns these low dimensional embeddings directly from the graph Laplacian, whereas other approaches (Raftery et al., 2002) propose a generative probabilistic model of graphs. The 'latent' locations of the entities are learnt via optimizing the likelihood of the probabilistic model. In our previous work (Sarkar & Moore, 2005) we generalized (Raftery et al., 2002)'s static model which associates each entity with a point in p-dimensional Euclidean latent space to a dynamic model which captured friendship ties evolving over time. Observed links between entities are more likely if the entities are close in latent space. The model based approach allows one to visualize graphs over time, and learn point estimates of these latent coordinates in $O(n \log n)$ time.

The other widely popular approach is to compute proximity between nodes in a graph by using random walks on graphs: diffusion of information from one node to another. Most of random-walk based ranking algorithms use either the probability of reaching a destination node from the query node or the expected number of hops to reach the destination as a measure of proximity. *Probability of reaching a node* is the basis of measures like *pagerank* (Brin & Page, 1998) and *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). All these approaches focus on computing approximate PPV at query time (details in section 3), and quantify the performance in terms of the deviation of the approximation from the exact. However, its not clear if PPV itself has good predictive power.

The other related proximity measure is the *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 image segmentation problems (Saerens et al., 2004). However these algorithms usually require $O(n^3)$ computation. A number of approximation techniques have been proposed to circumvent this problem, however these techniques do not scale to graphs with more than few thousands of nodes and they are not applicable to directed graphs. Recently (Spielman & Srivastava, 2008) has come up with a very novel approximation algorithm for efficiently computing commute times in undirected graphs by random projections. However the algorithm relies on a rather complicated nearly-linear time solver (Spielman & Teng, 2004), which has not yet been realized in practice.

The hitting and commute times suffer from sensitivity to long paths (Liben-Nowell & Kleinberg, 2003) (resulting in non-local nature) and are prone to be small if one of the nodes is of large degree (Brand, 2005) (ineffective for personalization purposes). We have introduced a truncated variant of random walks in order to exploit local structure in graphs, and devised algorithms to compute all interesting pairs of near-neighbors under this truncated version of commute times without having to compute the entire commute times matrix. However this algorithm (GRANCH) required storing potential nearest neighbors of all nodes in the graph in order to answer nearest neighbor queries. We addressed this with an incremental algorithm where we combine sampling with deterministic pruning to retrieve top k neighbors of a query in truncated hitting and commute time incrementally without caching information about all nodes in the graph.

In this thesis we propose to explore random walk-based proximity measures, investigate theoretical properties of short-term random walks on graphs in order to improve our existing algorithms and gain insight for designing novel algorithms for fast nearest neighbor search in large graphs.

Here is the organization of the document in a nutshell: we provide the basics of random walks on graphs in section 2. Section 3 discusses the use of random walk based approaches in various applications areas, and previous work on improving computational complexity. In section 4 we define the truncated hitting and commute times and investigate their local properties (4.3) for directed and undirected graphs. Section 5 describes the intuition behind our branch and bound style algorithm to efficiently prune away nodes which are far away from the query node. We discuss the hybrid algorithm which combines random sampling and deterministic pruning in section 6. In section 7 we present the performance of our algorithm, along with other popular proximity measures (Katz, PPV, Jaccard, Adamic-adar etc.) on various link-prediction tasks on undirected co-authorship networks, and directed entity relation graphs from Citeseer. We also present the average wall-clock time per query on these tasks. We present our proposed work in section 8. Section 8.1 shows how to use the locality properties proven in 4.3 for improving the existing algorithm. 8.2 contains other graph-based proximity measures we want to investigate. In section 8.3 we propose a meta learning algorithm for link prediction. Finally we conclude with possible applications in section 8.4.

2 Random walks on graphs: Background

In this section we introduce some basic concepts of random walks on graphs. 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. D is an $n \times n$ diagonal matrix, where $D_{ii} = \sum_j W_{ij}$. We will denote D_{ii} by the degree $d(i)$ of node i from now on. The Laplacian L of G is defined as $D - W$. $P = p_{ij}, i, j \in V$ denotes the transition probability matrix of this markov chain, so that $p_{ij} = w_{ij}/D_{ii}$ if $(i, j) \in E$ and zero otherwise. In a random walk, if node v_0 is chosen from a distribution x_0, then the distributions $x_0, x_1, ..$ are in general different from one another. However if $x_t = x_{t+1}$, then we say that x_t is the stationary distribution for the graph.

If the world-wide-web is considered as a graph then the pagerank p is defined as the distribution that satisfies the following linear equation(Brin & Page, 1998): $\mathbf{p} = (1 - c)\mathbf{p}P + \frac{c}{n}\mathbf{1}$. c is the probability of restarting the random walk at a given step. This gives a democratic view of the respective importance of the webpages. However by using a non-uniform teleportation density we can create a personalized view of the relative importance of the nodes. Thus the personalization vector for node r will be given by $\mathbf{p_r} = (1 - c)\mathbf{p_r}P + c\mathbf{e_r}$, where e_r is the vector with the r^{th} entry set to 1. We now introduce two main proximity measures derived from random walks, namely the hitting and the commute time (Aldous & Fill, 2001).

Hitting time H The hitting time between nodes i and 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. $H = h_{ij}, i, j \in V$ is an $n \times n$ matrix. Recursively h_{ij} can be written as $h_{ij} = 1 + \sum_k p_{ik}h_{kj}$, if $i \neq j$ and zero otherwise. The hitting time can also be interpreted as weighted path-length from node i to node j, i.e. $h_{ij} = \sum_{path(i,j)} |path(i, j)|P(path(i, j))$. Hitting times are nonsymmetric, however they follow the triangle inequality (Lovasz,).

Commute time C Commute time between a pair of nodes is defined as $c_{ij} = h_{ij} + h_{ji}$. In undirected graphs c_{ij} can also be defined as

$$\begin{aligned} c_{ij} &= V(G)(l^+_{ii} + l^+_{jj} - 2l^+_{ij}) \\ &= V(G)(e_i - e_j)^T L^+(e_i - e_j) \end{aligned} \tag{1}$$

Since each row of the graph Laplacian sums to zero, it has an eigenvector of all ones, with eigen-value zero. It is common practice (Saerens et al., 2004) to shift the zero-eigenvalue by subtracting the all ones matrix, inverting the resulting matrix, and then adding the all ones matrix back again, i.e. $L^+ = (L - \frac{1}{n}\mathbf{1}\mathbf{1}^T)^{-1} + \frac{1}{n}\mathbf{1}\mathbf{1}^T$.

The pseudo-inverse of L, i.e. L^+, can be viewed as a kernel (Smola & Kondor, 2003) which maps each vertex of a graph to a Euclidian space $i \mapsto x_i$. Pairwise commute time can be expressed as the squared Euclidian distance in the transformed space (equation (1)), where $x_i = (L^+)^{\frac{1}{2}}e_i$. Let us now look at the entries of L^+ in terms of these position vectors. The ij-th entry $l^+_{ij} = x_i^T x_j$, denotes the dot-product between the position vectors of vertex i and j. The diagonal element l^+_{ii} denotes the squared length of the position vector of i. The cosine similarity (Brand, 2005) between two nodes is defined as $l^+_{ij}/\sqrt{l^+_{ii}l^+_{jj}}$. We discuss different ways of computing the above quantities in Section 3.

For undirected graphs there is a direct analogy of hitting and commute times with electrical networks (Doyle & Snell, 1984). Consider an undirected graph. Now think of each edge as a resistor with conductance W_{ij}. For undirected graphs the commute time can be shown(Chandra et al., 1989) to be equal to the effective resistance between two nodes up to a multiplicative constant. This also gives us some intuition about the effectiveness of commute times as proximity measures. First, if the effective resistance between two nodes are small, then its easier for information to diffuse from one node to the other. Second, since electrical properties of networks do not change much by adding or removing a few edges, hitting and commute times should be robust to small perturbation in datasets.

3 Related Work

The random walks approach has been highly successful in social network analysis (Katz, 1953) and computer vision (Gorelick et al., 2004; Qiu & Hancock, 2005), personalized graph search (Jeh & Widom, 2002a; Haveliwala, 2002; Fogaras et al., 2004), keyword search in database graphs (Balmin et al., 2004; Chakrabarti, 2007), detecting spam (Gyongyi et al., 2004; Hopcroft & Sheldon, 2007; Wu & Chellapilla, 2007). Here we briefly describe some of these applications.

3.1 Application in Clustering and Computer Vision

Saerens et al. (2004) exploit equation (1) to embed a graph and provide distances between any pair of nodes. Yen et al. (2005) replace traditional shortest-path distances between nodes in a graph by hitting and commute times and show that standard clustering algorithms (e.g. K-means) produce much better results when applied to these re-weighted graphs. These techniques exploit the fact that commute times are very robust to noise and provide a finer measure of cluster cohesion than simple use of edge weight.

Most clustering applications either require extensive computation to produce pairwise proximity measures, or employ heuristics for restricting the computation to subgraphs. Saerens et al. (2004) compute the trailing eigenvectors of L to reconstruct L^+. This still requires at least $O(n^2)$ computation and storage.

Gorelick et al. (2004) use the average hitting time of a random walk from an object boundary to characterize object shape from silhouettes. Grady and Schwartz (2006a) introduced a novel graph clustering algorithm which was shown to have an interpretation in terms of random walks. Hitting times from all nodes to a designated node were thresholded to produce partitions with various beneficial theoretical properties. Grady and Schwartz (2006b) used the above approach for automated image segmentation. Also Qiu et al have used commute times clustering for robust multibody motion tracking (Qiu & Hancock, 2006) and image segmentation (Qiu & Hancock, 2005).

3.2 Collaborative Filtering

Commute times and hitting times have been successfully used for collaborative filtering. In Brand (2005) the authors use different measures resulting from random walks to recommend products to users based on their purchase history . The authors give empirical evidence of the fact that hitting and commute times often are small if one of the nodes has a high degree.whereas the cosine similarity (defined in Section 2) does not have that problem since in some sense the effect of individual popularity is normalized out. Cubic computation is avoided by using sparse matrix manipulation techniques. In Brand (2005) the authors try to compute sub-matrices of H and C by iterative sparse matrix multiplications. Note that computing the i^{th} row of C, i.e. $C(i, *)$ or the cosine-similarities of all j with i requires $l^+_{jj}, \forall j$. The exact value of l^+_{jj} requires solving linear systems of equations for all other rows of L^+. The authors state that for large markov chains the square of the inverse stationary distribution of j is a close constant factor approximation to l^+_{jj}, which is fairly loose. In short, it is only tractable to compute these measures on graphs with a few thousand nodes for most purposes.

3.3 Link prediction

Given a snapshot of a social network, a very interesting question is: which new connections among entities are likely to occur in the future? Liben-Nowell and Kleinberg (2003) investigate different proximity measures between nodes in a graph for predicting future links in social networks. The authors showed that the hitting and commute times suffer from 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 so that small paths are given more weight. 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. However, computing this for all pairs of nodes needs at least $O(n^2)$ time and storage. Even if A is sparse the fast linear solvers will take at least $O(E)$ time to compute the Katz score of all nodes from a query node.

Tong et al. (2007) uses escape probability from node i to node j (probability to hit node j before coming back to node i) to compute *direction aware proximity* in graphs. A fast matrix solver is used to compute this measure between *one pair of nodes*, in $O(E)$ time. Multiple pairs of proximity require computation and storage of the inverse of the matrix $I - \gamma P$. Although ranking using escape probabilities was not computed, proximity measures from the source to all other nodes would be needed for ranking purposes, 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.

3.4 Personalized graph search

Consequent to the immense success of *pagerank* (Brin & Page, 1998) there has been a significant amount of work in the area of computing personalized pagerank. 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. The problem is that 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 (Jeh & Widom, 2002a; Haveliwala, 2002; Fogaras et al., 2004). Most of these algorithms compute partial personalized pagerank vectors offline and combine them at query time.

The *ObjectRank* algorithm in (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. For keyword search surfers start random walks from different objects containing that word. Any surfer either moves randomly to a neighboring node or jumps back to a node containing the keyword. The final ranking is done based on the resulting probability distribution on the objects in the database. In essence the personalized pagerank for each word is computed and stored offline, and at query time combined linearly to generate keyword-specific ranking. 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.

3.5 Detecting Fraud

As more and more graph-related problems are springing to life, detecting spammers or fraudsters in networks is becoming a major issue. The term web-spam refers to hyperlinked pages on the World Wide Web that are created with the intention of misleading search engines(Gyongyi et al., 2004). There have been many algorithms which show the effectiveness of random walk based approaches (Gyongyi et al., 2004; Hopcroft & Sheldon, 2007; Wu & Chellapilla, 2007) to combat spam. These problems easily fit into the semi-supervised (Zhu et al., 2003) learning framework, where given labeled and unlabeled data-points as nodes in a graph, the main goal is to exploit the graph structure to label the unlabeled examples using the few labeled ones.

3.6 Other graph-based proximity measures

Spectral clustering (Ng et al., 2001) is a body of algorithms that cluster datapoints x_i using eigenvectors of a matrix derived from the affinity matrix W constructed from the data. Often w_{ij} is obtained from the Euclidian distance between the datapoints, i.e. $w_{ij} = \exp(\|x_i - x_j\|^2/\sigma^2)$, where σ is a free parameter. Zhu et al. (2003) uses graph theoretic approach for semi-supervised learning. Given labeled and unlabeled data-points as nodes in a graph, the main goal is to exploit the graph structure to label the unlabeled examples using the few labeled ones.

3.7 Constant factor approximation of commute times using random projections

Recently (Spielman & Srivastava, 2008) has designed a fast algorithm for computing pairwise commute times within a constant factor approximation. The idea is to use the Johnson-Lindenstrauss lemma in conjunction with a nearly linear time solver (Spielman & Teng, 2004) to compute a $n \times \log n$ matrix of random projections in $\tilde{O}(m)$ time. At query time the commute time can be computed by computing the Euclidian distance between two points in $O(\log n)$ time. This approach has couple of problems,

1. Its only applicable to undirected graphs.

2. The projections have to be recomputed if the underlying graph is changing continuously. Even if the underlying graph is not undergoing continuous changes, its not clear how to use this efficiently for computing distance from a set of nodes. If we merge the nodes, then all the coordinates will have to be computed again.

4 Short Term Random Walks on Graphs

In this section we present a novel algorithm to compute approximate nearest neighbors in commute time for all nodes, without computing the entire matrix.

4.1 Truncated Hitting times

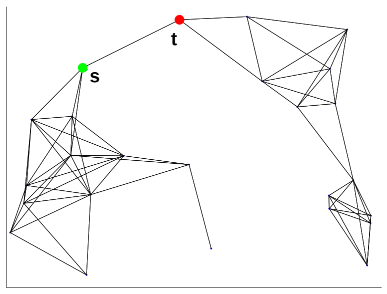


Figure 1: The non-local nature of hitting times

Consider the undirected graph such that two clusters C_s and C_t are connected via a single edge between s, t. Node s and t are direct neighbors, however the hitting time from s to t is very large, because the random walk spends a lot of time in parts of the graph which are far away from s. In fact for the above case where the edge (s,t) divides the graph in two clusters, C_s and C_t, such that s belongs to C_s and t to C_t, we can prove that $h(s,t) = 2 * E(C_s) + 1$, where $E(C_s)$ is the number of edges in cluster C_s. This implies that the rank of t in the list of nodes sorted according to the hitting time from s can be pushed down arbitrarily if we increase number of edges in C_s. This shows the non-local nature of hitting times.

Hitting and commute times decrease if there are many short paths between the nodes. On the other hand, as observed by Liben-Nowell and Kleinberg (2003), two major drawbacks are that they tend to be small whenever one of the nodes has a high degree. This renders them ineffective for personalization purposes. They are also sensitive to parts of the graph far away from the nodes, even when short paths between the nodes exist. This results in non-local nature.

In order to overcome these problems we define a *T-truncated hitting time*, where we only consider paths of length less than T. In Section 4.2 we show that h^T approaches the true hitting time as $T \to \infty$. We use h, h^t interchangeably to denote truncated hitting time.

The T truncated hitting time i.e. h^T_{ij} from i to j, measures the expected number of steps taken by a random walk starting at node i to hit j for the first time. It can also be defined recursively as

$$h^T_{ij} = 1 + \sum_k p_{ik} h^{T-1}_{kj} \tag{2}$$

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. Equation (2) can be easily computed by using Gauss Seidel iterations. Let the truncated hitting times matrix be H^T. Note that the hitting time from all nodes to a fixed destination, i.e. a column of H^T can be computed in $O(T\Delta n)$ time, where Δ is the average degree of a node. However if we need to compute the hitting time from one node to everyone else, i.e. a row of H^T, $H^T(i,*)$ then $\forall j$ we will compute the corresponding column in order to get one entry $H^T(i,j)$. Hence we shall end up computing the entire matrix. Instead of examining all pairs of nodes we develop a very efficient framework for doing a range-search to retrieve the k approximate nearest neighbors of *all* nodes in a graph.

4.2 Truncated and True Hitting Times

In this section we examine the relation between the true and truncated hitting times. Let $\mathcal{P}$ be the set of paths between i, j, such that j appears only once in the path. Let $\mathcal{P}_t$ be all such paths of length exactly t. So $\mathcal{P} = \bigcup_t \mathcal{P}_t$. We can then express the truncated hitting time as an expected path length:

$$h_{ij}^T = E(|\mathcal{P}|) = \sum_{t=1}^{T-1} t Pr\{i \xrightarrow{t} j\} + (1 - \sum_t Pr\{i \xrightarrow{t} j\})T$$

Where $Pr\{i \xrightarrow{t} j\}$ denotes the probability that i reaches j for the first time in exactly t steps.

Define $\tilde{P}$ as a matrix identical to P, except for the j^{th} row. $\forall m \;\; \tilde{P}(j,m) = 0$. Hence after the random walk hits j from i, there is a loss of probability mass from the system, which is why the sum is bounded. Thus $Pr\{i \xrightarrow{t} j\} = \tilde{P}_{i,j}^t$. Substituting in the above equation:

$$h_{ij}^T = \sum_{t=1}^{T-1} t\tilde{P}_{ij}^t + (1 - \sum_{t=1}^{T-1} \tilde{P}_{ij}^t)T$$

Note that as $T \to \infty$ the probability of i not reaching j goes to zero if the graph is connected. Hence the above gets reduced to the true hitting time.

The rate at which the T-truncated hitting time approaches the true hitting time depends on the mixing time of a random walk on the graph. The mixing time equals the number of steps before the random walk converges to the stationary distribution. It is the inverse of the mixing rate (Lovasz,) which measures how fast the distribution approaches the stationary distribution. This can be easily computed from the eigenvalues of the graph Laplacian. One way of setting the value of T dynamically will be by evaluating the mixing time. For a graph with a small mixing time, T should be small, since after a few steps the distribution becomes independent of the start node.

4.3 Locality Properties

Let us see how this property of the truncated hitting times affect the AP-set size. A very loose bound on the APset size can be obtained by taking all nodes within T hops of a node to compute the truncated hitting times exactly. This upper bound will give us $O(n^2)$ bound in most real world graphs due to the small world phenomenon(Milgram, 1967) in real graphs.

4.3.1 A bound on nodes within T' hitting time from i

Using this part we get a very simple bound for the APset size if it **only** consists of all pairs within $T-$truncated hitting time T'.

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

$$T' \leq h_{ij} \leq T(1 - P_{ij}^{<T}) \implies P_{ij}^{<T} \geq \frac{T - T'}{T} \tag{3}$$

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

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

However the left hand side is lower bounded by $|S_i^+| \frac{T-T'}{T}$ using (3). Which leads us to the upper bound

$$\boxed{|S_i^+| \leq \frac{T^2}{T - T'}} \tag{4}$$

.

4.3.2 A bound on nodes within T' hitting time to j

Define S_j^- as the neighborhood of j which consists of only the nodes within hitting time T' **to** j.

All total there are only $N\frac{T^2}{T-T'}$ pairs within T' hitting distance. Without any distributional assumption we can only claim that there can be at most $K\sqrt{n}$ columns with $\sqrt{n}$ elements in it. Note that other bounds are possible, but for $\sqrt{n}$ we can say that there are not *too many* columns with *too many* nonzero elements in them. For general graphs this is all we can claim so far. However if the graph is undirected, i.e. the markov chain is reversible then we can make stronger claims, as shown below.

Similarly as before we have

$$P_{ij}^{<T} \geq \frac{T-T'}{T} \tag{5}$$

We use the property of a reversible chain, since the underlying graph in undirected. Let π be the stationary distribution of the random walk. Then we have, (Aldous Fill chapter 3)

$$\pi(i)P^t(i,j) = \pi(j)P^t(j,i) \tag{6}$$

$$\begin{aligned}\sum_{i\in S_j^-} P_{ij}^{<T} = \sum_{i\in S_j^-}\sum_{t=1}^{T-1}\tilde{P}_{ij}^t = \sum_{t=1}^{T-1}\sum_{i\in S_j^-}\tilde{P}_{ij}^t \leq \sum_{t=1}^{T-1}\sum_{i\in S_j^-}P_{ij}^t \leq \sum_{t=1}^{T-1}\sum_{i\in S_j^-}P_{ji}^t\frac{\pi(j)}{\pi(i)} \\ \leq degree(j)\sum_{t=1}^{T-1}\sum_{i\in S_j^-}P_{ji}^t \leq T\times degree(j)\end{aligned}$$

We used the fact that $\pi(j)/\pi(j) = degree(j)/degree(i) \leq degree(j)$, since minimum degree is 1.

However the left hand side is lower bounded by $|S_j^-|\frac{T-T'}{T}$ using (5). Which leads us to the upper bound:

$$\boxed{|S_j^-| \leq degree(j)\frac{T^2}{T-T'}} \tag{7}$$

4.3.3 A bound on nodes within $2T'$ commute time to j

We already have a bound of $T^2/(T-T')$ on the number of nodes with hitting time smaller than T' **from** a node i, and a bound of $degree(i)T^2/(T-T')$ for undirected graphs. We want to bound the number of nodes with commute time $2T'$. Here we will give a bound for general graphs, and a stronger one for undirected graphs.

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

1. $S_1 = \{j|h_{ij}^T \leq T' \bigcap h_{ij}^T + h_{ji}^T < 2T'\}$
2. $S_2 = \{j|h_{ij}^T \geq T' \bigcap h_{ij}^T + h_{ji}^T < 2T'\}$

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

Lemma 4.1 *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 in any (directed or undirected) graph not more than $O(\sqrt{n})$ nodes will have more than $O(\sqrt{n})$ neighbors with truncated commute time smaller than $2T'$.*

Lemma 4.2 *For an undirected graph, the total number of neighbors within $2T'$ T- truncated commute time can be upper bounded by $(1+degree(i))\frac{T^2}{T-T'}$.*

$$\boxed{\{j|c_{ij}^T < 2T'\} \leq |S_i^+| + |S_i^-| \leq (1+degree(i))\frac{T^2}{T-T'}} \tag{8}$$

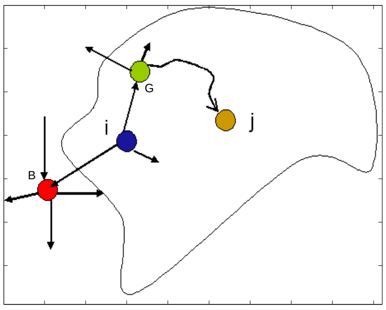


Figure 2: Neighborhood of node j. A directed graph is drawn for clarity.

5 GRANCH: Algorithm to compute all pairs of nearest neighbors

The main intuition is that we think about the graph in terms of n overlapping subgraphs. Each subgraph is a bounded neighborhood for one node (discussed in detail later in the section). Consider the hitting times arising from the random walk from any node in this neighborhood to the destination. We provide upper (*pessimistic*) and lower (*optimistic*) bounds of these hitting times, and show how to combine information from all these overlapping subgraphs to obtain k nearest neighbors for all nodes. We will use the terms "optimistic" and "lower", and "pessimistic" and "upper", interchangeably. In order to motivate the GRANCH algorithm we present the following claim before formally describing the algorithm.

Claim: Given k, ϵ, T and $T' \leq T$, GRANCH returns any k neighbors j of i $\forall i$, s.t. $c^T(i,j) \leq c^T(i, k_{i,T'})(1+\epsilon)$, where $k_{i,T'}$ is the true k^{th} nearest neighbor of i within T-truncated hitting time T'.

This makes sense since in all applications hitting and commute times are used for ranking entities, e.g. recommend the k best movies based on choices already made by an user; or, find the k most likely co-authors of an author in a co-authorship network.

Suppose we want to estimate h^T_{ij} for pairs of nodes i,j which have relatively low h^T_{ij} values, and suppose we want to avoid $O(n^2)$ time or space complexity. Therefore we cannot do anything that requires representing or iterating over all pairs of nodes. In fact, we cannot even afford to iterate over all pairs of nodes that are less than T hops apart, since in general the number of such pairs is $O(n^2)$. Suppose we restrict computation to iterate only over some subset of pairs of nodes, which we will call the Active-Pairs-set or the AP set. The interesting question is how good are the pessimistic and optimistic bounds we can get on h^T_{ij}, $\forall i,j$ using only $O(|AP|)$ work?

As mentioned before we will consider bounded neighborhoods of each node in the graph. For clarity, in case of a directed graph the neighborhood consists of all nodes with short paths *to* the destination node. Here is our algorithm in a nutshell: for each node, we will start with the direct neighbors in its neighborhood, and then compute the optimistic and pessimistic bounds on hitting times of the nodes within a neighborhood to that node. We will then expand the neighborhood and re-iterate. The bounds will be tighter and tighter as we keep expanding the neighborhoods.

Define the set AP as a set of pairs of nodes such that if i is in the neighborhood of j, then $(i,j) \in AP$. Note that $(i,j) \in AP$ does not imply that $(j,i) \in AP$. Each neighborhood of node i has a boundary, δ_i, such that a boundary node on the neighborhood of j has direct neighbors outside the neighborhood. Again in a directed graph a boundary node has outgoing edges to nodes outside the neighborhood.

Let's first assume that the neighborhood is given. It's clear from the expression for truncated hitting time that h_{ij} can be computed from the hitting time of i's neighbors to j. In Figure 2 the random walk from i can hit either of it's neighbors G or B. Since G is inside the neighborhood of j we already have an estimate of h_{Gj}. However B is outside the neighborhood of j, and hence we find the optimistic and pessimistic values of h_{Bj}, resulting into lower bounds (ho_{ij}) and upper bounds (hp_{ij}) of h_{ij}. The optimistic bound is obtained by allowing the random walk to jump to the boundary node with closest optimistic hitting time to j, and the pessimistic bound arises from the fact that the walk might never come back to the neighborhood after leaving it, i.e. takes T time.

Now we revisit the data-structures in terms of their notations. We denote the set of active pairs as AP. $AP(*,i)$ is the set of nodes k, s.t. $(k,i) \in AP$. Hence the neighborhood of i can also be denoted by $AP(*,i)$. Also $\delta(i)$ is the set of the boundary nodes of the neighborhood of i, i.e. $AP(*,i)$. Let $nbs(i)$ denote the set of direct neighbors of i. Also denote by $Gnbs(i,j)$ the set $AP(*,j) \cap nbs(i)$. The upper and lower bounds are:

$$hp_{ij}^{T} = 1 + \sum_{k \in Gnbs(i,j)} p_{ik} hp_{kj}^{T-1} + (1 - \sum_{k \in Gnbs(i,j)} p_{ik})(T-1) \tag{9}$$

$$ho_{ij}^{T} = 1 + \sum_{k \in Gnbs(i,j)} p_{ik} ho_{kj}^{T-1} + (1 - \sum_{k \in Gnbs(i,j)} p_{ik})(1 + \min_{p \in \delta(j)} ho_{pj}^{T-2}) \tag{10}$$

We can also use a tighter bound than the above by using one step lookahead from the nodes on the boundary, which is omitted for space purposes. The *ho* and *hp* values of all nodes in $AP(*,j)$ to j can be computed using the Gauss-Seidel technique, as in Algorithm 5. After obtaining the bounds on the hitting times

Algorithm 1: compute-H(dst,AP,G,T)

```
1: $ho, ho_{last}, hp, hp_{last} \leftarrow ones(1:N)$[1]
2: $minvals_{0:T} \leftarrow 0$; $minvals_1 = 1$
3: for $t = 2$ to $T$ do
4:   $min \leftarrow \infty$
5:   for $src \in AP(*,dst)$ do
6:     $s_1, s_2 \leftarrow 0$, $prob \leftarrow 0$
7:     for $i \in nbs(src)$ do
8:       if $i \in AP(*,dst)$ then
9:         $s_1 \leftarrow s_1 + p_{src,i} ho_{last}(i)$
10:        $s_2 \leftarrow s_2 + p_{src,i} hp_{last}(i)$
11:        $prob \leftarrow prob + p_{src,i}$
12:      end if
13:    end for
14:    $ho(src) \leftarrow 1 + s_1 + (1 - prob)(1 + minvals_{t-2})$
15:    $hp(src) \leftarrow 1 + s_2 + (1 - prob)(t-1)$
16:    if $ho(src) \le min$ & $src \in \delta(dst)$ then
17:      $min \leftarrow ho(src)$
18:    end if
19:  end for
20:  $minvals_t \leftarrow min$
21:  $ho_{last} \leftarrow ho$
22:  $hp_{last} \leftarrow hp$
23: end for
```

between all pairs in the AP-set, we can obtain bounds on hitting times as follows:

$$ho_{ij} = \begin{cases} ho_{ij} & \text{if } i \in AP(*,j) \\ 1 + \min_{p \in \delta(j)} ho_{pj}^{T-1} & \text{else} \end{cases} \tag{11}$$

$$hp_{ij} = \begin{cases} hp_{ij} & \text{if } i \in AP(*,j) \\ T & \text{else} \end{cases} \tag{12}$$

Note that we are explicitly giving the expressions for all pairs of nodes in the graph for clarity. However the bounds need only $O(|AP|)$ space. Using the above we also get optimistic and pessimistic bounds on commute times, namely *co* and *cp*.

$$co_{ij} = ho_{ij} + ho_{ji}; \quad cp_{ij} = hp_{ij} + hp_{ji} \tag{13}$$

[1]Note that $\forall i,j \;\; h^1(i,j) = 1$.

5.1 Expanding Neighborhood

Since time and space complexity of GRANCH is $O(|AP|)$, we have to be clever in having the right set of pairs in the APset, in order to avoid wasting computation on nodes that are very far apart. We present a *neighborhood expansion scheme* which will guarantee that we do not miss any potential nearest neighbors. We also note that adding nodes to the neighborhood of i can only tighten the bounds.

Theorem 5.1 *The optimistic bounds on H, i.e. ho values can only* ***increase****, and the pessimistic bounds, i.e. hp values can only* ***decrease****, as a result of adding nodes to the AP set.*

Proof: The hitting time for a node outside the AP-set must be at least the minimum hitting time to the boundary of the AP-set, plus the minimum hitting time from the boundary to the destination. For a destination node j, $ho^{T-1}_{m,j} \geq 1 + \min_{p \in \delta(j)} ho^{T-2}_{pj}$, $\forall m \notin AP(*, j)$. Hence ho^T_{ij} in (10) is indeed a valid lower bound on h^T_{ij}. Therefore it can never decrease.

The pessimistic bound in (9) is valid, since $h^t_{ij} \leq t \; \forall \; i, j$, by construction of h^t. Hence the pessimistic bounds can never increase.

Definition: Define the lower bound of the neighborhood of node i as

$$lb(i) = 1 + \min_{p \in \delta(i)} ho^{T-1}_{pi} \tag{14}$$

Lemma 5.2 *The lower bound on hitting time of nodes outside the AP-set of node i, i.e. $lb(i)$, can only* ***increase*** *as a result of adding nodes to the AP set.*

Proof: This is a direct application of Theorem 5.1.

We start with all direct neighbors of the destination node i. Note that all nodes outside the boundary will take at least $1 + \min_{p \in \delta(i)} ho^{T-1}_{pi}$ time to reach i. Let us denote this by $lb(i)$. Also remember that we are doing a range search of all neighbors within a range $T' < T$. We want to increase this lower bound, so that we can guarantee that we shall never leave out a potential nearest neighbor. The optimal way to increase the lower bound is to add in the neighbors of q, where $q = \arg\min_{p \in \delta(i)} ho^{T-1}_{pi}$.

Algorithm 2: Expand-AP(G, T, T')

```
1: Initialize[2]AP with all pairs $i, j \in E_G$
2: for $i = 1$ to $N$ do
3:    while $lb(i) \leq T'$ do
4:       compute-H($i$,$AP$,$G$,$T$)
5:       $lb(i) \leftarrow 1 + \min_{p \in \delta(i)} ho^{T-1}_{pi}$
6:       $q \leftarrow \arg\min_{p \in \delta(i)} ho^{T-1}_{pi}$
7:       $AP \leftarrow AP \cup \{(a, i) : a \in nbs(q)\}$
8:    end while
9: end for
```

Lemma 5.3 *If $i \notin AP(*, j)$ after running Algorithm 2, then $h_{ij} \geq T'$.*

After running algorithm 2, $ho_{i,j} \geq lb(j) \geq T'$. Also by construction we have $h_{i,j} \geq ho_{ij}$.

5.2 Approximate Nearest Neighbors

We shall first look at how to obtain the k ϵ-approximate nearest neighbors for each node in hitting time. Later we shall demonstrate how to extend that to approximate nearest neighbors in commute times.

Note that we only have upper and lower bounds on hitting times from a node to other nodes. The key idea is that any value x will be smaller than y if the upper bound of x is smaller that the lower bound of y.

[2]The AP-set can be initialized to all pairs within p hops too. We used $p = 2$ for some of our experiments.

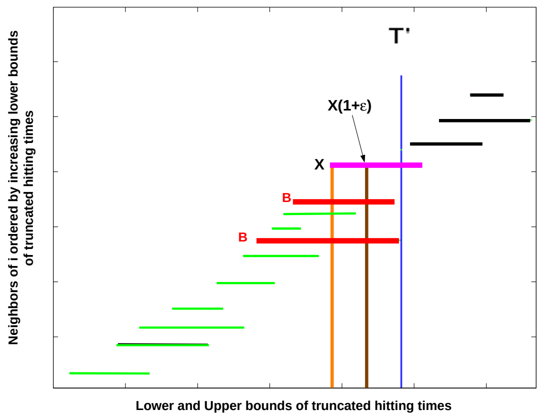


Figure 3: Upper and lower bounds of $h^T(i,*)$

5.2.1 Truncated hitting time

We illustrate with a toy example. Lets assume that we want to find the k nearest neighbors of node i within T'. In Figure 3 we plot the lower and upper bounds of $h_{i,j}$, in the order of increasing lower bounds. It will suffice to find all nodes j, s.t. hp_{ij} is smaller than the h value of the $(k+1)^{th}$ true nearest neighbor. Note that the $(k+1)^{th}$ nearest neighbor's hitting time is greater than the $(k+1)^{th}$ largest ho value. Hence the upper-bound becomes $X = (k+1)^{th}\_largest\_ho_{i,*}$. Now we also allow a small error margin of ϵ leading to a new upper bound $X(1+\epsilon)$. All entities with upper bound less than this are guaranteed to be ϵ-approximate.

We also want the neighbors to be within T' hitting time of the source i. Hence we find the largest $k' \leq k+1$, s.t. $ho_{i_{k'},i} \leq T'$ and $hp_{i_{k'},i} \geq T'$, where $i_{k'}$ is the node with the k'^{th} largest ho value from i. Now the upper bound X becomes $X = ho_{i,i_{k'}}$. We return all nodes x, s.t. $hp_{i,x} < X(1+\epsilon)$. Therefore in Figure 3 we have $k = 15$, $k' = 10$. This means that at this point the bounds are not tight enough to determine the exact ranks of the $k - k'$ neighbors. All the nodes with lower bounds beyond T' are guaranteed not to be within T'. The nodes labeled B are not guaranteed to be ϵ approximate and are not returned.

5.2.2 Truncated commute time

Now we look at the neighbors in commute times. The new distance for range-search becomes $2T'$. For any node i, we now look at all nodes in the set $AP(i,*) \cup AP(*,i)$. $AP(i,*)$ denotes the set of all nodes that have i in their neighborhoods. After running Algorithm 2, if there exists some node j, s.t. $j \notin AP(*,i)$ and $i \notin AP(*,j)$, then by Lemma 5.3, $ho(i,j), ho(j,i) \geq T'$ and hence $co(i,j) \geq 2T'$, so j cannot be within a commute distance of $2T'$ of i.

5.2.3 Ranking using upper and lower bounds

We compute the nearest-neighbors for node $i, i \in \{1:n\}$ using the APset resulting from Algorithm 2. Now we find the nearest neighbors of i in commute times using equation (13). Note that we only look at the nodes in set $S_i = AP(i,*) \cup AP(*,i)$. Sort all the nodes in increasing order of optimistic bounds i.e. co values. Let $\{i_1, i_2, ...\}$ denote the indexes of the sorted nodes. Let k' the largest integer less than k, s.t. $co(i, i_{k'}) \geq 2 * T'$. If such a k' exists define X as $2 * T'$. Otherwise clearly all k possible nearest neighbors might be within the range. Hence define X as $co(i, i_k)$. Return any node $j \in S_i$ s.t. $cp(i,j) \leq X(1+\epsilon)$. Also let B be the set of nodes b in S_i s.t. $co(i,b) \leq X$, but $cp(i,b) \geq X(1+\epsilon)$. Clearly these nodes cannot be guaranteed to be among the k nearest neighbors. Now we expand the AP-sets of $\{i\} \cup B$ to tighten the bounds more.

Because of the construction of the algorithm, the newly introduced nodes can never affect k', and the other nearest neighbors. By Lemma 5.3, we will never introduce a new possible nearest neighbor of a node $b \in \{i\} \cup B$, since all nodes outside S_b are guaranteed to have commute-distance worse than $2T'$. Also an expansion of the AP-set can only increase the lower bounds, and decrease the upper bounds on the estimates on hitting times, which will never decrease the number of neighbors returned so far. Hence we will only improve the performance of the algorithm by expanding our AP-set beyond algorithm 2.

GRANCH uses Algorithm 2 to build the AP-set and then computes the k ϵ-approximate nearest neighbors in truncated commute time.

6 Hybrid algorithm

In the last section we presented an algorithm which computes all pairs of approximate nearest neighbors in truncated commute times. In this section we present an algorithm to compute approximate nearest neighbors in commute times, without caching information about all nodes in the 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 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.

6.1 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 T-length random walks from i. Lets say out of these M runs m random walks hit j at $\{t_{k_1}, ...t_{k_m}\}$ time-steps. From these we can estimate

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 $1 - \delta$.

2. The number of samples M required in order to get the top k neighbors correct.

Theorem 6.1 *In order to obtain* $P(|h^T(i,j) - \hat{h}^T(i,j)| \geq \epsilon T) \leq \delta$ *number of samples* M *should be at least* $O(T^2/\epsilon^2 \times \log \frac{nT}{\delta})$.

It can be proved (in Appendix) that these estimates are also MLE's of the hitting times.

Lemma 6.2 $\forall j$ *the MLE of* h^T_{ij} *is* $\hat{h}^T_{ij}$.

Theorem 6.3 *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* $1/(2\alpha^2)\log(nk/\delta)$ *in order to have* $Pr(\exists j < k, r > k, \hat{h}^T(i, v_j) \geq \hat{h}^T(i, v_r)) \leq \delta$

Note that the above theorem says nothing about the order of the top k neighbors, only that all of them will be retrieved. We could change the statement slightly to obtain a sample complexity bound to guarantee the exact order with high probability. The only difference will be that the new α will be $\min_{j<k} h^T(i, v_{j+1}) - h^T(i, v_j)$. As expected, much fewer samples are required to get close approximation of correct ranking than for close approximation of actual values. The details are provided in the appendix.

6.2 Lower and Upper Bounds of Truncated Commute Times

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) from (10), (9). The hitting time from j to nodes within $NBS(j)$ can be estimated using the sampling scheme described in section 6.1. Combining the two leads to the following.

Theorem 6.4 *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 $1 - \delta$ if the number of samples for estimating $\hat{h}^T_{ij}$ exceeds the lower bound in theorem 6.1, where*

$$co^T_{ij} = \hat{h}^T_{ij} + ho^T_{ij} \tag{15}$$

$$cp^T_{ij} = \hat{h}^T_{ij} + hp^T_{ij} \tag{16}$$

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 14. Now note that for the outgoing lower bound we need the minimum of $h^T_{jk}, \forall k \notin NBS(j)$. Now from lemma 6.2 and the invariance property (Casella & Berger, 2001) of ML estimates we claim that

Lemma 6.5 *The MLE of $\min_{k \notin NBS(j)} h^T_{jk}$ equals $\min_{k \notin NBS(j)} \hat{h}^T_{jk}$.*

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

$$\textit{lb-commute}(j) = 1 + \min_{p \in \delta(j)} ho^{T-1}_{pj} + \min_{k \notin NBS(j)} \hat{h}^T_{jk} \tag{17}$$

6.3 Expanding neighborhood

Now the question is how to expand the neighborhood. We need to find a heuristic such that both the outgoing and incoming components of the lower bound increase quickly so that the threshold of $2T'$ is reached soon. Conceptually this is equivalent to evenly spreading out the neighborhood in commute times.

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 i.

As 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 i which stop at the boundary. These paths are augmented one step at a time. This enables us to look-ahead one step 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.

6.3.1 Ranking

The ranking scheme is similar to GRANCH and is rather intuitive. 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.

6.4 The algorithm at a glance

In this section we will 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 neighbor with high probability.*

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 such 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 bounds on commute times (16) with high probability. We also keep track of the lower bound $lb - commute$ of the commute time from any node outside the neighborhood to the query node, as in equation (17). At each step we expand the neighborhood using the heuristic in section 6.3. Similar to GRANCH we recompute the bounds again, and keep expanding until $lb - commute$ exceeds $2T'$.

This guarantees that all nodes outside the neighborhood have commute time larger than $2T'$. Note that earlier we proved that this lower bound can never decrease, since the bounds get tighter as the neighborhood is expanded. This is true for our hybrid algorithm as well (details excluded for lack of space).

Then we use the ranking as in section 6.3.1 to obtain k α-approximate nearest neighbors in commute time. We start with a small value of T' and increase it until all k neighbors can be returned. As in GRANCH its 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. We show that this quantity is not more than $O(degree(q))$ in section 4.3.

7 Empirical results

We will evaluate our algorithm on link prediction tasks on undirected co-authorship graphs, and directed Entity Relation (ER) datasets extracted from the Citeseer corpus.

7.1 Co-authorship networks

Here is a brief description of the experimental methodology. We remove a random subset of links from the original graph. Now we compute the $2T$-truncated-commute times on the training links. We only take nodes which has at least one link held out. For each such node we take the proximity measures from the algorithms of all nodes within 5 hops of the source in the original graph(not all other nodes) and count how many of the held-out neighbors appear in top 10 neighbors. We present the average score.

We present the performances of the following algorithms in table 1.

1. Sampled hitting times (H)
2. Hybrid commute times (C)
3. Number of hops (truncated at 5)(B1)
4. Number of common neighbors(B2)
5. Jaccard score(B3)
6. Adamic-Adar score(B4)
7. Katz score(B5)
8. PPV (B6)
9. Random (B7)

We present the average time in seconds per query in table 3. T in all tables translates to a $2T$-truncated commute time. It was shown in (Liben-Nowell & Kleinberg, 2003) that simple measures like number of

Table 1: AUC score on 30% links held out from graphs of size n, edges e , for sampled hitting times (H) and hybrid commute times (C), with $T \in \{10, 6\}$,**B1**: Number of hops between node-pairs within 5 hops, **B2**: Number of common neighbors, **B3**: Jaccard **B4**: Adamic-Adar, **B5**: Katz, **B6**:PPV and **B7**: Random.

n	e	H, C, T=10	H, C, T=6	B1	B2	B3	B4	B5	B6	B7
19,477	54,407	81,85	83,86	47	71	80	77	78	78	.07
52,033	152,518	76,80	80,82	41	66	76	73	72	74	.04
103,500	279,435	77,81	77,80	38.5	66.5	73.4	71	73	75	.02

common neighbors, Adamic-adar or Jaccard do surprisingly well on link prediction tasks. The Adamic adar score between two nodes i and j is defined as $\sum_{z \in nbs(i) \bigcap nbs(j)} 1/(log(|nbs(z)|)$. The Jaccard score is defined as $|nbs(i) \bigcap nbs(j)|/|nbs(i) \bigcup nbs(j)|$. Note that common neighbors, Adamic adar and Jaccard would not be applicable if we try to predict links between two layers of nodes in a bipartite graph. However these do perform quite well in our experiments, which shows that these co-authorship networks have a lot of transitive

Table 2: Time in seconds on link prediction tasks on graphs of size n, edges e , for sampled hitting times (H) and hybrid commute times (C), with $T \in \{10, 6\}$, Katz, and PPV.

n	e	H, C, T=10	H, C, T=6	Katz	PPV
19,477	54,407	.01,.05	.01,.03	.8	.5
52,033	152,518	.01,.11	.01,.05	2.6	1.7
103,500	279,435	.01,.19	.01,.09	6	3.7

links, i.e. a node is very likely to have a link with the neighbor of a neighbor. Katz and PPV, as shown in table 3 are the most expensive algorithms. In all the cases commute times perform the best, hitting times are the second best, and Jaccard and PPV are in the third place.

7.2 Entity Relation graphs

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.

7.2.1 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 (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 a weight of 10 times the weight of its single edge, so that the hitting time 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.

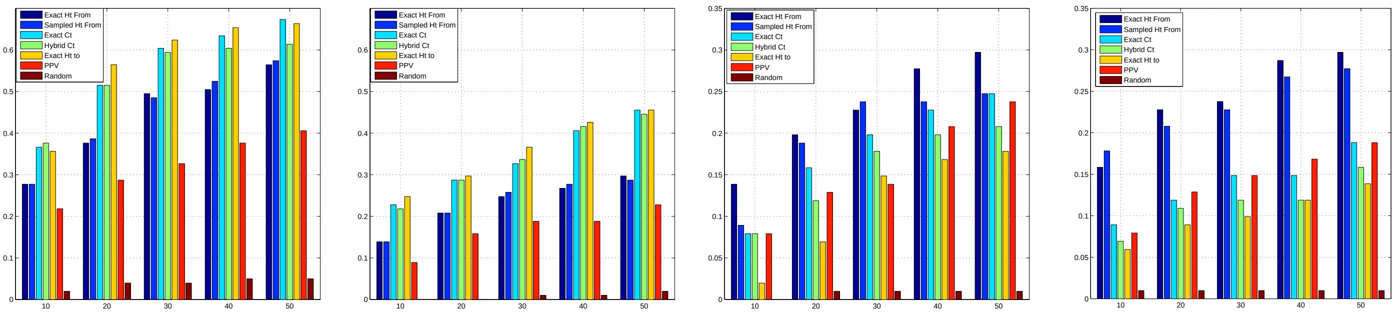
7.2.2 Preprocessing

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. Excluding these words *did not change* the predictive performance of any of the algorithms. We will make the exact dataset used available on the web.

7.2.3 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* 10 *results.*



(A) (B) (C) (D)

Figure 4: Author task for (A)Small dataset and (B)Large dataset. Word task for (C)Small dataset and (D)Large dataset

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 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 algorithms will slower than the sampling or the hybrid algorithm as in table 3.

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. 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.

Table 3: 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	1.53	.02	7.01	.26	5.21	35.4
		Word	2.83	.05	8.9	.63	5.5	62
628, 130	2, 865, 660	Author	6.1	.06	75.2	1.36	64.74	219
		Word	12.1	.14	85.9	3.2	68.4	316

Figure 7.2.3 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 3 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 (authors) 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 two values of k, 10 and 50.

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 sometimes the exact PPV. However for the word task the outcome of our experiments are **exactly 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 that the distance from the paper layer to the word layer. Also note that the prediction task for words is much harder than the prediction task for authors. Out of the 100 test nodes only around 40 paper-titles had the removed paper in the 50 nearest papers. On the other hand for the author prediction task the distance from the paper payer-layer to the authors and that from the author-layer to the paper are equally important. This is why the exact commute time and the exact hitting time to the author nodes do equally well and beat all the other competitors.

Note that the weakness of PPV, as revealed in our experiments is its predictive power, and not the *runtime*. We do not use any approximation algorithm for computing these vectors, since the goal was to compare the predictive power of PPV with our algorithms. We used $c = 0.1$ (c is the teleportation probability), so that the average path-length for PPV is around 10, since we use $T = 10$ for all our algorithms (this also gives better predictive performance than $c = .2$, which is the standard choice).

Word task: The sampling based hitting time algorithms beat everyone else by a large margin. PPV beats the hitting time to nodes for the word task. In fact for $k = 10$ the hitting time *to* a query node isn't much of an improvement over the random predictor. This again emphasizes our claim that the hitting time *to* the word layer does not provide a lot of information for the word task. In fact this also brings down the performance of the exact and hybrid commute times.

Author task: The hitting time *to the query* and the exact commute time *from a query* has the best performance. 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 smaller graph we use 1000 samples and $100,000$ samples for computing the exact hitting times from a query node. We use 1.5 times each for the larger graph. Note that in the above figures sometimes the exact hitting time does worse than the sampled hitting time. This might happen by chance, with a small probability.

8 Proposed Work

We propose to extend our previous work, investigate other proximity measures and also apply our algorithms to a variety of graph mining problems.

8.1 Scopes for extending and improving the current algorithm

8.1.1 Faster algorithm for undirected graphs

In section 4.3 we gave upper bounds for number of nodes within T'-truncated hitting and commute distances in graphs. However we have not yet used this to analyze the runtime of our algorithm. For an undirected graph, we want to ask the question: is there a lot of overlap between the incoming T'-neighborhood and the outgoing T'-neighborhood? In other words, does there exist a T'', such that $h^T(i,j) \leq T'$ will imply $h^T(j,i) \leq T''$. If this is true, then we can sample outgoing paths to guess the right neighborhood for the hitting time to a node.

8.1.2 Shape of T' neighborhood

In section 4.3 we have given upper bounds for number of nodes within T'-truncated hitting and commute distances in graphs. However we have not enumerated the shape of these neighborhoods. It would be interesting to explore how the shape of these neighborhoods vary from graph to graph. In other words we would ask following question

$$\boxed{\text{Given } T,\ T',\ \text{can we upper bound the number of hops between nodes } i \text{ and } j,\ \text{such that } h^T(i,j) \leq T'?}$$

This is interesting because if we can show that for some graphs this distance is small, then we would be able to argue that a naive search algorithm which looks at all nodes within some c hops of the query node, only examines $O(c^\rho)$ nodes, where ρ is the growth constant (Krauthgamer & Lee, 2003) of the graph. We have some very initial results in this direction. We intend to improve them in the future.

8.1.3 Filling in the gap between GRANCH and the incremental hybrid algorithm

In the previous sections we presented two algorithms for approximate nearest neighbor search in hitting and commute times, namely the GRANCH algorithm, and the Hybrid algorithm. The first one caches the potential nearest neighbors of all nodes in the graph in order to answer nearest neighbor queries, whereas the second can do the same *with high probability* incrementally. We have achieved two ends of the spectrum of exploiting cached information and incremental algorithms. The question is can we better the performance by using best of both worlds.

Lets review the locality results we obtained in section 4.3. For an undirected graph we have shown that the number of nodes within $2T'$ T-truncated commute distance from a node i is upper bounded by $(degree(i)+1)T^2/(T-T')$. Also lemma 4.1 shows that for a directed graph, if T' is bounded away from T by a constant (w.r.t n) then at most $O(\sqrt{n})$ nodes will have $O(\sqrt{n})$ neighbors within $2T'$ commute distance. This means that we can essentially detect the nodes which will have a largely populated column in the hitting time matrix. For undirected graphs these will be the high degree nodes, and for directed graphs we can detect them with high probability using sampling schemes. We want to explore this direction more in order to exploit cached information more in our incremental search.

8.2 Different random walk based proximity measures

8.2.1 Discounted Hitting Times

We have so far considered truncated hitting times, where only paths of length up to T are considered. A more natural way of formulating this is the γ-discounted hitting time, where at any time step the random walk dies with probability $1-\gamma$. Hence the maximum hitting time between any pair of nodes is $1/(1-\gamma)$. We can use our algorithms for computing approximate nearest neighbors in γ-discounted random walk. Given a neighborhood of node i $NBS(i)$, the lower and upper bounds on hitting times from any node j inside $NBS(i)$ to i, will be given by

$$\boxed{\begin{aligned} hp_{ij} &= 1+\gamma[\sum_{k\in Gnbs(i,j)} p_{ik}hp_{kj} + (1-\sum_{k\in Gnbs(i,j)} p_{ik})\frac{1}{1-\gamma}] \\ ho^T_{ij} &= 1+\gamma[\sum_{k\in Gnbs(i,j)} p_{ik}ho_{kj} + (1-\sum_{k\in Gnbs(i,j)} p_{ik})(1+\gamma \min_{p\in\delta(j)} ho_{pj}]) \end{aligned}}$$

We intend to also change the algorithm to compute approximate nearest neighbors in discounted hitting and commute times.

8.2.2 Other proximity measures

For undirected graphs I would also like to explore other proximity measures, such as spectral clustering based approaches, and un-truncated commute time and cosine-similarity measures (Brand, 2005). Recently there has been some work (Chung, 2005) which tries to generalize various properties of undirected graphs to directed graphs by introducing directed graph laplacians. I would like to explore this direction more to see if commute times in directed graphs can be computed using random projections (Spielman & Srivastava, 2008) as well.

8.3 A meta learning approach for link prediction

In our latest experiments we made an interesting, and intuitive observation: the performance of any link prediction algorithm is very task-specific. Consider, for example a ER-graph, that has a word layer, a paper layer and an author layer. Also consider the following tasks: given a paper we remove all the links between it and the words on its title, and then try to see if the paper gets a rank in top 10 neighbors in hitting/commute times in this modified graph. We denote this by the word task. The exact same task can be defined for the author layer and we will denote this by the author task. We observed that hitting times beat the competing algorithms for the word task whereas commute times perform the best for the author task.

We propose a link prediction algorithm such that the link predictor will be a function of several proximity measures, e.g. hitting time, commute time, shortest path, number of common neighbors etc. Since we have fast algorithms for computing these measures we should be able to do the above very fast. It would be interesting to see the weights on the different components. In fact we would be able to characterize a link prediction task by the different weights learnt for these different measures of similarity. In fact this can also be looked at as an efficient estimator of the truncation parameter T of our algorithm, since we can generate a different feature for different values of T, and the regressor will automatically pick the right T for the right graph. A graph with a small mixing time will need a small T, whereas one with a large mixing time might need a larger truncation parameter.

We would like to use this meta learner for various real-world data-sets, and different tasks for a given data-set.

8.4 Applications

Most real-world problems on graphs require ranking of nodes with respect to a given query node. For recommender networks the query is to find the top k products *close* to a particular customer. For keyword search in databases (Balmin et al., 2004) we are interested in finding the top k entities close to the keywords entered by the user. For spam detection the near neighbors of a spammer are very likely to be spammers as well. For personalized graph search (Fogaras et al., 2004; Jeh & Widom, 2002a) we are interested in top k neighbors of the set of websites particular to an user. The key property common to all the above applications is that two nodes are similar if they are connected by many short paths. Hence a metric like hitting/commute time is a perfect candidate for ranking nodes. The work in progress enables us to compute neighbors in truncated hitting/commute time without iterating over all nodes in the graph. Thus it would be possible to process queries on the fly *without caching any information* on very large graphs.

I have promising results on co-authorship-citation-keyword graphs (similar to (Balmin et al., 2004)) obtained from the Citeseer domain. In my thesis I also intend to explore the following application areas.

8.4.1 Semi-supervised Learning

Random walk based methods has been popular in the semi-supervised learning framework (Szummer & Jaakkola, 2001), (Zhu et al., 2003). The former uses the probability distribution in a t step random walk starting from the labeled nodes in the graph to obtain a distribution of labels over the nodes in the graph. The second uses harmonic functions, which are smooth over the graph, and are a priori fixed at the labeled nodes. The idea is that nodes connected to one another will not have different labels. For the first formulation the selection of t is crucial, whereas the second requires inverting an $U \times U$ matrix, where U is the number of unlabeled points. In both cases the assumption is that nodes with similar labels will be connected by many paths. We want to use our diffusion-based proximity measures in this semi-supervised setting, i.e. for detecting spam nodes in a web-graph where only a few nodes are labeled. The idea would be that nodes *close* to these spam nodes are also potentially spammers.

8.4.2 Recommender Networks:

Random walk based approaches has been successfully applied to recommender systems. Since these graphs (e.g. Netflix dataset) are huge, incremental search algorithms are very appropriate. It would be interesting to apply our techniques for prediction in these settings.

8.4.3 Information retrieval:

Recently there has been a lot of work in the area of information retrieval which uses random walks to capture similarity between entities in directed entity relation graphs, or semantic similarity in word co-occurrence graphs. For example, consider the problem of entity resolution, where the same author is mentioned using different names. The question is if we can say which all author-names belong to the same author. (Minkov et al., 2006) use random walks with restart for contextual search and name disambiguation in email-graphs. (Toutanova et al. 2004) focus on learning the edge weights in a graph, by using random walks. (Minkov & Cohen, 2007) has also proposed a model to learn the transition probabilities in ER-graphs for improving ranking. The idea is to iterate between a ranking phase using random walks with restart, and a gradient-descent update phase of the edge probabilities. (Collins-Thompson & Callan, 2005) use random walks to determine which words in a document corpus are good candidates for query expansion. Since most of these algorithms use short term random walks to rank nodes in an inner loop, we believe that there is a large scope for application of our fast incremental algorithms. We would like to explore this direction in the thesis.

9 Appendix

Proof of Theorem 6.1: We provide a bound on the number of samples M required in order to give an ϵ-correct answer with probability $1-\delta$. We shall denote the Maximum Likelihood estimate of a quantity x by $\hat{x}$ from now on. Lets denote by $X^r(i,j)$ the first arrival time at node j from node i on the r^{th} trial. Note that, $\widehat{\tilde{P}^t}(i,j)^3 = \frac{\sum_r I(X^r(i,j)==t)}{M}$. The truncated hitting time from i to node j is given by $h^T(i,j) = \sum_t t\tilde{P}^t_{ij} + (1-\sum_{t=1}^{T-1}\tilde{P}^t_{ij})T$. As the truncated hitting time is a linear combination of $\tilde{P}^t_{ij}$, the proof is based on decomposing the error between these probabilities. Then a simple hoeffding bound to obtain good estimates of these probabilities with high probabilities leads to the sample complexity bound.

If we allow an additive error of $\epsilon/2T$ on $\tilde{P}^t_{ij}$, i.e. for $t \in \{1,2,...,T-1\}$, we obtain an additive error of ϵT on the truncated hitting time estimates, as shown below. We have $|\widehat{\tilde{P}^t_{ij}} - \tilde{P}^t_{ij}| = |\Delta\tilde{P}^t_{ij}| \leq \frac{\epsilon}{2T}$.

$$\begin{aligned}&|\hat{h}^T(i,j) - h^T(i,j)| = \Delta h^T(i,j)\\ &\leq \sum_{t=1}^{T-1} t|\Delta\tilde{P}^t_{ij}| + \sum_{t=1}^{T-1}|\Delta\tilde{P}^t_{ij}|T \quad \leq \boxed{\epsilon T}\end{aligned} \tag{18}$$

Now we give a very simple Hoeffding-type union bound on the sample size. First note that $\widehat{\tilde{P}^t_{ij}}$ is an unbiased estimate of $\tilde{P}^t_{ij}$, since the later is basically a parameter for a multinomial distribution.

Now we want the probability of a bad estimate for any j and any $t \leq T$ to be low. We upper bound this error probability using union and Hoeffding bounds and set the upper bound to a small value δ.

$$\begin{aligned}&P(\exists j \in \{1,\dots,n\}, \exists t \ \in \{1,\dots,T-1\}|\widehat{\tilde{P}^t_{ij}} - \tilde{P}^t_{ij}| \geq \tfrac{\epsilon}{2T})\\ &\leq nTe^{-\frac{M\epsilon^2}{2T^2}} \leq \delta\end{aligned}$$

This gives the lower bound of $\frac{2T^2}{\epsilon^2}\log\frac{nT}{\delta}$ on the number of samples required to yield an ϵ-approximate answer with high $(1-\delta)$ probability.

Proof of Lemma 6.2: Sampling gives us estimates on $\tilde{P}^t(i,j)$. This is a parameter of a multinomial distribution. Also the estimate is a ratio of counts, which is a MLE of multinomial parameters. Since hitting time is a linear combination of these parameters, using the functional invariance property we can show that $\hat{h}^T(i,j)$ is MLE of $h^T(i,j)$.

Proof of Theorem 6.3: Consider a sampled path of length T starting from i. As before we denote by $X^r(i,u)$ the first arrival time at node u on the r^{th} trial. Define $X^r(i,u) = T$ if the path does not hit u on trial r. 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 $Z^r = X^r(i,u) - X^r(i,v)$. Note that $E(Z) = h^T(i,u) - h^T(i,v)$.

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 $P(\sum_{r=1}^M Z_r < 0)$ which using the Hoeffding bound is smaller that $e^{-2M(h^T(i,u)-h^T(i,v))^2}$.Let $v_1, v_2, \dots, v_k$ be the top k neighbors of i.

$$\begin{aligned}&Pr(\exists j \leq k, r > k, \hat{h}^T(i,v_j) \geq \hat{h}^T(i,v_r))\\ &\leq \sum_{j\leq k}\sum_{r>k} Pr(\hat{h}^T(i,v_j) \geq \hat{h}^T(i,v_r))\\ &\leq \sum_{j\leq k}\sum_{r>k} e^{-2M(h^T(i,v_r)-h^T(i,v_j))^2}\\ &\leq \mathrm{nk}e^{-2M(h^T(i,v_{k+1})-h^T(i,v_k))^2}\end{aligned}$$

Setting the above to δ gives us the desired lower bound of $1/(2\alpha^2)\log(nk/\delta)$ on M.

[3] P^t_{ij} equals the probability to transition from i to j in t steps, whereas $\tilde{P}^t_{ij}$ denotes the probability to hit j *for the first time* in t steps from i.

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.*

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Proc. WWW.*

Casella, G., & Berger, R. L. (2001). *Statistical inference.* Duxbury Press.

Chakrabarti, S. (2007). Dynamic personalized pagerank in entity-relation graphs. *WWW '07: Proceedings of the 16th international conference on World Wide Web* (pp. 571–580). New York, NY, USA: ACM Press.

Chandra, A. K., Raghavan, P., Ruzzo, W. L., & Smolensky, R. (1989). The electrical resistance of a graph captures its commute and cover times. *STOC'89* (pp. 574–586).

Chung, F. (2005). Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics.*

Collins-Thompson, K., & Callan, J. (2005). Query expansion using random walk models. *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management.*

Doyle, P. G., & Snell, J. L. (1984). *Random walks and electric networks.* The Mathematical Assoc. of America.

Fogaras, D., Rcz, B., Csalogny, K., & Sarls, T. (2004). Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. .

Gorelick, L., Galun, M., Sharon, E., Basri, R., & Brandt, A. (2004). Shape representation and classification using the poisson equation. *CVPR.*

Grady, L., & Schwartz, E. L. (2006a). Isoperimetric graph partitioning for data clustering and image segmentation. *IEEE PAMI.*

Grady, L., & Schwartz, E. L. (2006b). Isoperimetric partitioning: A new algorithm for graph partitioning. *SIAM Journal on Scientific Computing.*

Gyongyi, Z., Garcia-Molina, H., & Pedersen, J. (2004). Combating web spam with trustrank. *VLDB* (pp. 576–587).

Haveliwala, T. (2002). Topic-sensitive pagerank. *Proceedings of the Eleventh International World Wide Web Conference.*

Hopcroft, J., & Sheldon, D. (2007). *Manipulation-resistant reputations using hitting time* (Technical Report). Cornell University.

Jeh, G., & Widom, J. (2002a). Scaling personalized web search. .

Jeh, G., & Widom, J. (2002b). Simrank: A measure if structural-context similarity. *ACM SIGKDD, Proceeding of the Internation Conference on Knowledge Discovery and Data Mining.*

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

Krauthgamer, R., & Lee, J. R. (2003). The intrinsic dimensionality of graphs. *Proceedings of the STOC'03.*

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

Lovasz, L. Random walks on graphs: a survey.

Milgram, S. (1967). The small world problem. *Psychology Today.*

Minkov, E., & Cohen, W. W. (2007). Learning to rank typed graph walks: local and global approaches. *WebKDD/SNA-KDD '07: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis.*

Minkov, E., Cohen, W. W., & Ng, A. Y. (2006). Contextual search and name disambiguation in email using graphs. *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval.*

Ng, A., Jordan, M., & Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems.*

Qiu, H., & Hancock, E. R. (2005). Image segmentation using commute times. *Proc. BMVC.*

Qiu, H., & Hancock, E. R. (2006). Robust multi-body motion tracking using commute time clustering. *ECCV'06.*

Raftery, A. E., Handcock, M. S., & Hoff, P. D. (2002). Latent space approaches to social network analysis. *J. Amer. Stat. Assoc.*, *15*, 460.

Saerens, M., Fouss, F., Yen, L., & Dupont, P. (2004). The principal component analysis of a graph and its relationships to spectral clustering.

Sarkar, P., & Moore, A. (2005). Dynamic social network analysis using latent space models. *SIGKDD Explorations: Special Edition on Link Mining.*

Smola, A., & Kondor, R. (2003). Kernels and regularization on graphs. *Learning Theory and Kernel Machines.*

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

Spielman, D., & Teng, S. (2004). Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. *Proceedings of the STOC'04.*

Szummer, M., & Jaakkola, T. (2001). Partially labeled classification with markov random walks. *Advances in Neural Information Processing Systems.*

Tong, H., Koren, Y., & Faloutsos, C. (2007). Fast direction-aware proximity for graph mining. *ACM SIGKDD, Proceeding of the Internation Conference on Knowledge Discovery and Data Mining.*

Toutanova, K., Manning, C. D., & Ng, A. Y. (2004). Learning random walk models for inducing word dependency distributions. *ICML '04: Proceedings of the twenty-first international conference on Machine learning.*

Wu, B., & Chellapilla, K. (2007). Extracting link spam using biased random walks from spam seed sets. *WWW.*

Yen, L., Vanvyve, L., Wouters, D., Fouss, F., Verleysen, F., & Saerens, M. (2005). Clustering using a random-walk based distance measure. *ESANN 2005* (pp. 317–324).

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. *ICML, volume 20.*