Fast Dynamic Reranking in Large Graphs

Purnamrita Sarkar
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
psarkar@cs.cmu.edu

Andrew W. Moore
Google Inc.
Pittsburgh, PA 15213
awm@google.com

ABSTRACT

In this paper we consider the problem of re-ranking search results by incorporating user feedback. We present a graph theoretic measure for discriminating irrelevant results from relevant results using a few labeled examples provided by the user. The key intuition is that nodes relatively closer (in graph topology) to the relevant nodes than the irrelevant nodes are more likely to be relevant. We present a simple sampling algorithm to evaluate this measure at specific nodes of interest, and an efficient branch and bound algorithm to compute the top k nodes from the entire graph under this measure. On quantifiable prediction tasks the introduced measure outperforms other diffusion-based proximity measures which take only the positive relevance feedback into account. On the Entity-Relation graph built from the authors and papers of the entire DBLP citation corpus (1.4 million nodes and 2.2 million edges) our branch and bound algorithm takes about 1.5 seconds to retrieve the top 10 nodes w.r.t. this measure with 10 labeled nodes.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval

General Terms

Algorithms, Experimentation

Keywords

search, random walk, harmonic function, semi-supervised learning

1. INTRODUCTION

With the ever-increasing popularity of information networks such as the web, citation networks (e.g. DBLP), friendship graphs (e.g. Facebook) or movie recommendation networks (e.g. Netflix) graph-based search algorithms are gaining importance. Standard search algorithms rank the nodes in the graph based on their relevance to the given query. However because of the complex structure of these networks, the top ranked nodes are often not relevant to the user's query. One common example is where the query entered is ambiguous. For example, in a DBLP graph different authors with the same name are often treated as one entity, which completely confuses standard search algorithms. Assume that the user is allowed to enter positive and negative relevance feedback on the top k nodes. The question is how to quickly produce a reranked list of results by incorporating this feedback? Since these networks are very large, the algorithm has to be extremely fast and memory efficient.

One solution would be to compute a proximity measure from the relevant nodes. TrustRank ([8]) uses personalized pagerank [9] from the trusted nodes to discriminate between good and spammy nodes in the web. Recent work [10] on ranking refinement uses boosting to learn the new ranking function simultaneously from the base ranking function and the user feedback. The problem with the first approach is that it does not take the negative information into account, and query-time computation of personalized pagerank is still an active area of research. The second approach is quadratic in the number of entities to be ranked and is not appropriate for quick reranking.

Harmonic functions for graph-based semi-supervised learning were introduced in [19]. Given a few positive and negative labels the harmonic function value at any node is simply the probability of hitting a positive label before a negative label. Variations of the same measure have also been successfully used for web-spam detection [11], automated image colorization [13] and image segmentation with user feedback [7]. Similar to much spreading activation work, the main intuition is that nodes close in the graph topology will have similar labels.

The standard tool for computing harmonic functions either involves sparse solvers for graph Laplacians or iterative matrix operations. There are algorithms which solve undirected graph Laplacians in near-linear time [17]. However these results do not apply to directed graphs. The main reason these approaches are not applicable to dynamic reranking is that they are computed once for the entire graph with static positive and negative labels. We want fast local computation for labels generated by users in real time.

In this paper we propose a short term variation of harmonic functions. In Entity-Relation graphs, or social networks, we are interested in information flow in shorter range. Hence we compute the probability of hitting a positive node before a negative node in T steps, where T is set to be around 10. This is similar to [11], where the harmonic rank is computed with respect to a random walk with a restart probability. We present two different algorithms for computing the above function. One is a simple sampling algorithm, and the other is a dynamic neighborhood expansion technique which provably returns approximate top k nodes under this measure. Both of these are useful for two different and complementary scenarios. If one had a candidate set of nodes to be ranked based on the feedback, the sampling technique is appropriate to compute function values at the specific nodes. On the other hand, the second algorithm is more appropriate for computing top k nodes in this measure for a given set of labels.

We used quantifiable entity disambiguation tasks in the DBLP corpus to evaluate the proposed measure and its conditional version (probability of hitting a positive node before a negative node given the random walk hits some label in T steps). We compare our results with two standard diffusion based proximity measures from the positive nodes in the labeled set: a) expected time to hit a node from the positive labels in T steps, and b) personalized pagerank from the positive labels. Our results indicate that the measure which combines information from both the positive and negative feedback performs much better than proximity measures which use only positive relevance information. We also show timing results on state of the art paper-author-citation graph (not containing explicit word nodes) built from DBLP. The branch and bound algorithm takes about 1.5 seconds to compute top 10 nodes for 10 labels on average.

The paper is organized as follows: we briefly describe related work in section 2, and motivate the reranking problem in section 3. In section 4 we define the discriminative measures (conditional and unconditional probabilities), and illustrate their behavior on toy examples. Section 5 contains the proposed algorithms. In section 6 we present our experimental results.

2. PREVIOUS WORK

In this paper we present a short term variation of harmonic functions on graphs. We show the effectiveness of these functions for semi-supervised classification and also present efficient algorithms to compute them. We first briefly describe previous applications of harmonic functions for semi-supervised learning. Then we discuss semi-supervised learning algorithms for web-spam detection and conclude with algorithms devised to incorporate user feedback for improved ranking and information retrieval.

Harmonic functions on graphs have been successfully used for graph-based semi-supervised learning algorithms. A harmonic function value at a node in a graph is an average over the values of its neighbors. This property leads to label-smoothness over the graph topology. Previous applications include standard machine learning classification tasks (the newsgroup and digits datasets) [19], image coloring [13] and image segmentation [7].

In the first application, a sparse graph is built using feature similarity between documents or digits, and then a subset of labels are used to compute the harmonic function. The other two applications use a graph-representation of an image frame, where two neighboring pixels share a strong connection if they have similar color, intensity or texture. The colorization application involves adding color to a monochrome image or movie. An artist annotates the image with a few colored scribbles and the indicated color is propagated to produce a fully colored image. The segmentation example uses user-defined labels for different segments and quickly propagates the information to produce high-quality segmentation of the image. All of the above examples rely on the same intuition: neighboring nodes in a graph should have similar labels.

A closer look at the formulation of a harmonic function (equation 1) shows that it can be computed using linear solvers for graph Laplacians. There has been much work on near-linear-time solvers [17] for undirected graph Laplacians. Recently a linear-work parallel algorithm was proposed for solving planar Laplacians [12]. Unfortunately these do not even apply to our directed graph setting.

There have been a number of learning algorithms to aid spam detection in the web, which is an extremely important task. These algorithms can be roughly divided into two classes: content-based methods, and graph based approaches. The content-based approaches ([14]) focus on the content of the webpage, e.g. number of words in the page and page title, amount of anchor text, fraction of visible content etc. to separate a spam-page from a non-spam page.

Graph-based algorithms look at the link structure of the web to classify web-spam. We will briefly discuss two of these. TrustRank [8] uses a similar idea as personalized pagerank computation [9]. The main idea behind personalized pagerank is to compute a stationary distribution of the Markov chain defined on the graph which is biased towards a set of nodes. At each step the random walk can be reset to this starting distribution with a small `teleportation` probability. In pagerank [4] this distribution is uniform over all webpages. In content-based keyword search applications in databases [3] this distribution is over the documents containing the searched keywords. In TrustRank this distribution contains a selected set of reputable web-pages.

Harmonic ranking has been successfully used for spam detection in [11]. The authors build an anchor set of nodes, and compute a harmonic function with restart. For the good anchor nodes the authors use the harmonic ranking where as for the bad anchors they use a forward-propagation from the anchor set to identify other nodes with similar labels. Note that although the authors only used the harmonic rank with a homogeneous anchor set (all good or all bad nodes), when the anchor set consists of both good and bad anchor nodes, the harmonic rank is very similar to our formulation. Other approaches include algorithms which combine both of the above, i.e. classify the spam pages using content and graph-based features. The authors in [1] optimize an objective function which minimizes the error on the labeled examples with an additional constraint of regularizing the function over the graph Laplacian. The idea is to penalize unsmooth functions over the graph-topology.

All these algorithms are used to classify a *static* set of spammy vs. good webpages. However in order to rerank nodes where the user-feedback is changing quickly these will have to be recomputed over the entire graph.

Ranking refinement using boosting was used by [10], where the main idea is to boost a base ranking function using the set of labeled examples. Every instance (a movie or a document) is defined by a set of features, and a base ranking algorithm outputs a ranking function from these objects. There is also additional label information obtained from user feedback which can be encoded as ranking preference. The goal is to combine these two and learn a ranking function which is consistent with both the base ranker and the user labels. The authors show that user-feedback considerably improves the performance of the base-ranking algorithm. A straight-forward use of the RankBoost algorithm[6] on the weak learners' ranking will give rise to an $O(n^4)$ algorithm. However the authors reduce the complexity of the algorithm to $O(n^2)$, i.e. quadratic in the number of instances to rank. Note that this is still expensive when the candidate set of instances to rank is large or the labeled set is changing over time and hence is not appropriate for producing quick reranking of results.

3. MOTIVATION

The graph in figure 1 is an example Entity-Relation graph constructed from the papers of an author *awm* in DBLP. There are three types of nodes: words (leftmost), papers (middle), and authors (rightmost). Words are connected to a paper if they appear on the title of the paper. Papers are connected via citation and authors are connected to a paper if they wrote the paper. We only show words which appear at least twice in this figure. For this example we also stemmed the word nodes, however for the experiments on the DBLP corpus we do not do so.

Careful examination of this graph will show that author *awm* has written papers on detecting disease outbreaks, and other topics including bayesian network structure learning, and link mining. Papers 4-10 contain the words significant, disease, outbreak, food, safety, scan, statistic, monitoring etc. Papers $0, 1, 2, 3$, and 12 contain the words asteroid, tractable, link, completion, group, structure, learning. Both of these groups contain common words, e.g. algorithm, detection, bayesian, network, pattern, spatial.

Let us assume a user searches for "awm", "disease" and "bayesian". This will return papers on disease outbreaks, and also other irrelevant papers from the other group of papers containing the word "bayesian" by author *awm*. Lets assume that the user is shown four papers. The user is unsure about the relevance of first result, and hence chooses not to label it, marks the second two (about disease outbreaks) as relevant and the fourth (about bayesian network structure learning) as irrelevant. A node is ranked high if in a 10-step random walk starting from it the probability of hitting a relevant node before an irrelevant one is large. Table 1 contains the results before and after incorporating user feedback. Note that the papers on spatial scan or spatial cluster detection are related to disease outbreaks, although the titles do not contain either of the words. They also come up to the top of the list. Also after incorporating the user feedback the list in the bottom panel shows a clear division between the relevant and irrelevant documents.

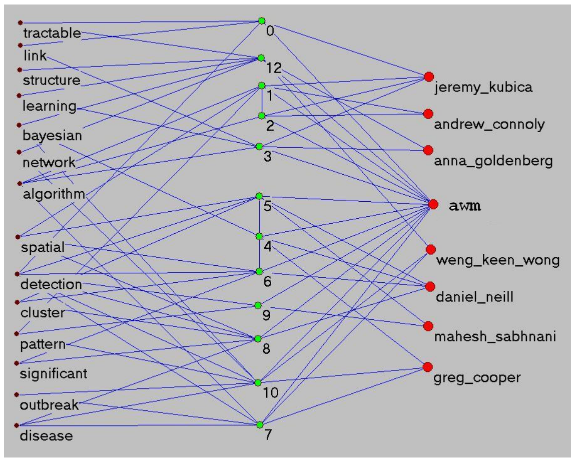


Figure 1: ER-graph for co-authors and papers of author *awm*

4. HARMONIC FUNCTIONS ON GRAPHS, AND T-STEP VARIATIONS

For the sake of clarity we will only describe a two class semi-supervised problem. However this can be easily generalized to a multiclass classification setting. Consider a graph with vertices V, among which U, and L are respectively the set of unlabeled and labeled nodes. A labeled node i has label $y_i \in \{0, 1\}$. Also from now on, we will interchangeably use 'positive' for label '1' and 'negative' for label '0'.

A harmonic function is defined as $f : V \rightarrow \Re$ which has fixed values at the given labeled points and is smooth over the graph topology. More specifically the harmonic property means that the function value at an unlabeled node is the average of function values at the neighbors. Let $w(i, j)$ be the weight on edge $< i, j >$, $d(i)$ be the weighted degree of node i, i.e. $d(i) = \sum_j w_{ij}$.

$$f(i) = \frac{\sum_j w_{ij} f(j)}{d(i)}, i \in U$$

For a directed graph w_{ij} might not equal w_{ji}, hence $d(i)$ will denote the weighted out-degree of node i. Let's assume that for the labeled nodes we assign $f(i) = 1$, if i has label 1, and 0 otherwise. This can also be expressed as $f = Pf$ where P is a row stochastic transition matrix of the graph. We now divide P into four blocks by grouping the labeled and unlabeled points together. f is grouped into f_u (unlabeled) and f_l (labeled nodes with fixed values). This gives:

$$f_u = (I - P_{uu})^{-1} P_{ul} f_l \tag{1}$$

The above function represents the probability of hitting a label '1' before a label '0'. We can easily generalize this to compute the probability of hitting '0' before '1', using a one vs. all encoding of labels. In order to compute the probability of hitting label $y \in \{0, 1\}$ before label $1 - y$ we set $f_l(i) = 1$, if node i has label y and 0 otherwise. By using a 2 column representation of f, where the y^{th} column $(f(:, y))$ encodes the label y nodes, we can compute $f_u(i, y)$ for $y \in \{0, 1\}$ simultaneously from equation (1). In other words $f(i, y)$ is the probability of hitting a node with label y node before a node with label $1 - y$.

In this paper we will use a T-step random walk. $f^T(i, 1)$ denotes the probability of hitting a label 1 node before a label 0 node in a T step random walk starting at node i. As T approaches infinity f^T approaches f. Note that in a strongly connected graph $f(i, 1) + f(i, 0) = 1$, since in infinite number of steps a random walk will hit some label. However in a T step random walk that is not the case.

Effectively f is the result of an infinite length markov chain, and is sensitive to long range paths. In this section we will briefly demonstrate how that could influence the classification obtained from f. We will use a simple way to classify a node: assign label 1 to it, if $f(i, 1) > f(i, 0)$, and similarly for f^T. Figure 2 has an undirected random graph of 100 nodes, where each node is assigned random x and y coordinates, and nodes close in the Euclidean space share a link with high probability. We used $T = 10$. The relatively larger nodes in the graph are labeled. We classify the rest of the nodes based on their f values (2A.) and their f^T values (2B.). The resulting classifications are color-coded (red squares:label 1) and (green circles: label 0).

In Figure 2A note that the nodes within the blue hand-marked area are labeled green by f, whereas they are marked

Table 1: Paper nodes from figure 1 ranked before and after incorporating user feedback (the papers annotated with "⇒" were labeled by an user. A "✓" implies that the paper is relevant to the query, whereas a "×" implies that it was irrelevant.)

Search results prior to user feedback	relevance
A Bayesian Scan Statistic	✓
⇒Bayesian Network Anomaly Pattern Detection for Disease Outbreaks.	✓
⇒Algorithm for Early Disease Outbreak Detection	✓
⇒Optimal Reinsertion: Bayesian Network Structure Learning.	×
Tractable Learning of Large Bayesian Network Structures	×
Detecting Significant Multidimensional Spatial Clusters.	✓
A Fast Multi-Resolution Method for Detection of Significant Spatial Disease Clusters.	✓
Detection of Emerging Spatial Cluster	✓
A Multiple Tree Algorithm for the Efficient Association of Asteroid Observations.	×
Variable KD-Tree Algorithms for Spatial Pattern Search and Discovery	×
Comparison of Statistical Machine Learning Algorithm on Link Completion Tasks	×
Monitoring Food Safety by Detecting Patterns in Consumer Complaints	✓
Tractable Algorithm for Group Detection and Link Mining in Large Datasets	×
Search results after incorporating user feedback	**relevance**
Bayesian Network Anomaly Pattern Detection for Disease Outbreaks.	✓
A Fast Multi-Resolution Method for Detection of Significant Spatial Disease Clusters.	✓
Algorithm for Early Disease Outbreak Detection	✓
A Bayesian Scan Statistic	✓
Detecting Significant Multidimensional Spatial Clusters.	✓
Detection of Emerging Spatial Cluster	✓
Monitoring Food Safety by Detecting Patterns in Consumer Complaints	✓
Tractable Algorithm for Group Detection and Link Mining in Large Datasets	×
Comparison of Statistical Machine Learning Algorithm on Link Completion Tasks	×
Variable KD-Tree Algorithms for Spatial Pattern Search and Discovery	×
A Multiple Tree Algorithm for the Efficient Association of Asteroid Observations.	×
Tractable Learning of Large Bayesian Network Structures	×
Optimal Reinsertion: Bayesian Network Structure Learning.	×

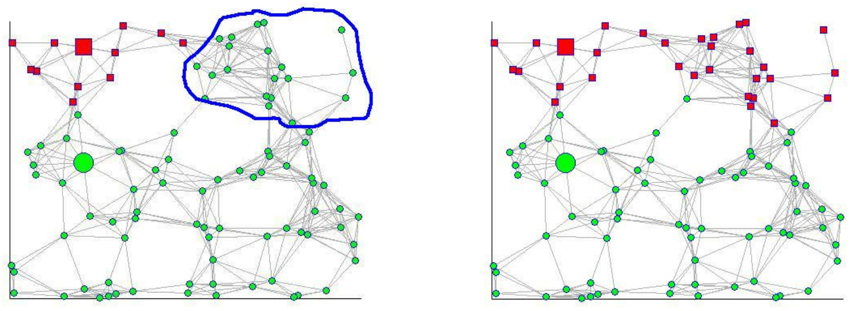

(A) (B)

Figure 2: A. Classification obtained from f, B. Classification obtained from f^T. A circle implies label '0', and a square label '1'. The two nodes with larger size are the two labeled nodes. The hand-marked area in (A) shows the difference in classification obtained from the infinite and short term random walks.

red by the short term random walk f^T. This shows the difference in behavior of the measures arising from long and short term random walks.

4.1 Unconditional vs. Conditional Measures

We have mentioned that $f^T(i,1)+f^T(i,0)$ can be strictly less than 1 for small T. This is because of the fact that the random walk might not hit any label in T steps. This observation leads to a new measure g^T.

$$g^T(i,1) = \frac{f^T(i,1)}{f^T(i,0)+f^T(i,1)} \quad (2)$$

g can also be viewed as the probability of hitting a label 1 node before a label 0 node in T steps, conditioned on the fact that the random walk hits some label in T steps. This measure clearly has more information than unconditioned f^T. However it can be misleading for the nodes where the probability of hitting any label in T steps, i.e. $f^T(i,0)+f^T(i,1)$ is very small. In order to alleviate this, we smooth the above with a small factor λ, which works like a prior belief on hitting a positive or negative label from i.

$$g^T_\lambda(i,1) = \frac{f^T(i,1)+\lambda}{f^T(i,0)+f^T(i,1)+2\lambda} \quad (3)$$

When $f^T(i,0)+f^T(i,1)=0$, this small change makes $g^T_\lambda = 0.5$. Note that this formulation puts a uniform prior over the two classes: a non-uniform prior is also possible.

In the rest of the section we will demonstrate how the unconditional, conditional and smoothed-conditional probabilities perform in terms of AUC scores in a toy example. We construct a toy example with 200 nodes, 2 clusters, 260 edges, out of which 30 are inter-cluster links. We will denote one cluster as the positive cluster and the other as the negative cluster. We made intra-cluster connections sparse, in order to emphasize the effect of T (not all nodes are reachable from others for small T). 20 randomly picked nodes are assigned labels. We vary the number of positive labels n_p from 1 to 19. For each number the AUC score is averaged over 10 random runs. In each random run n_p positive nodes are chosen from the positive cluster and $n_n = 20 - n_p$ negative nodes are chosen from negative cluster uniformly at random. For each random run the $f^T(i,1), g^T(i,1), g^T_\lambda(i,1)$ values of the unlabeled nodes i are computed and AUC scores for each function is recorded and averaged over the runs. The results are shown in figure 3. The upper panel of the figure has the results for $T = 5$, whereas the lower panel has results for $T = 10$. Here are the interesting observations:

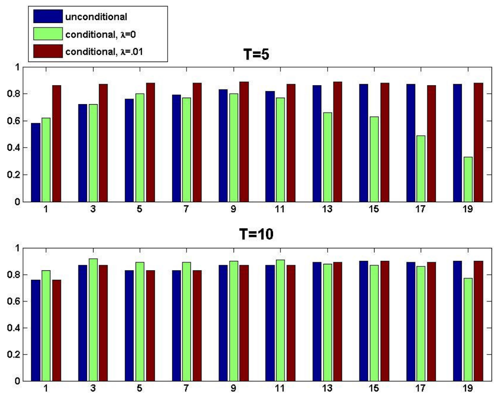


Figure 3: From left to right the bars show AUC scores for unconditional, conditional with $\lambda = 0$ and conditional with $\lambda = 0.01$, respectively. x-axis shows number of positive labels, and y-axis shows AUC scores.

1. The three functions perform comparably for $T = 10$.
2. When $T = 5$, for small n_p the unconditional probabilities (f^T) and the unsmoothed conditional probabilities (g^T) perform poorly. As n_p is increased both of these show an improved performance. However the performance of g^T becomes poor as n_p increases beyond 10.
3. The performance of g^T_λ is good (around and above 80%) for different values of T and different number of positive labels.

Let us justify the three observations. For $T = 5$ some nodes might not reach any labels, leading to the different behavior of the three different algorithms. Whereas for $T = 10$ all of them perform reasonably well. Since the number of labels is fixed at 20, small n_p values indicate large n_n (number of negatives). For small values of n_p, length 5 random walks from many nodes hit some negative node before hitting any positive node, and hence $f^T(i,1)$ values are zero. For all these nodes $g^T(i,1)$ is also zero, leading to poor AUC score. The parameter λ enables $g^T_\lambda(i,1)$ to successfully use the $f^T(i,0)$. As n_p increases and n_n decreases we observe a symmetric situation w.r.t the negative nodes. Even if ranking w.r.t. $f^T(i,1)$ gives good AUC scores (now that we have a lot of positive labels), the unsmoothed conditional function $g^T(i,1)$ evaluates to 1 for many nodes, since length 5 random walks from those now do not hit a negative node before a positive node. g^T_λ however uses best of both worlds.

5. ALGORITHMS

In this section we will present two different algorithms to compute the probability of hitting a positive node before a negative node within T steps. We will consider two scenarios which justify the use of both algorithms. 1) The user wants to rerank only the top 100 search results shown before based on his/her user feedback. 2) The user wants to rerank the entire graph based on the user feedback.

How do we justify these two needs? The first one is useful when we believe that the top 100 results from the search engine prior to any relevance feedback are most relevant to the query. Hence reranking will simply reorder these to show the most relevant results on top of the ranklist. Now consider the following case: lets say the user searches for the author Ting Liu in DBLP. There are two authors in DBLP with the same name, and they are treated as the same record. One of these authors is much more prolific compared to the other. The top 100 results will mostly contain results contextually similar to the prolific author. If however the user is interested in the less prolific author, then the better approach is to rank-order all nodes in the graph.

For the first scenario we present a simple sampling algorithm, whereas for the second we propose a branch and bound algorithm which finds the potential top-ranked nodes by provably pruning away the un-interesting nodes. For notational simplicity we will ignore the superscript T and subscript λ for now. Unless pointed out f represents f^T and g represents g^T_λ. Also we will use positive label for label 1 and negative label for label 0.

5.1 The Sampling Scheme

For every node i in the candidate set simulate M independent length T random walks. A walk stops if it hits any labeled node. Lets say M_p of these hit a positive node, and M_n of these hit a negative node. Hence the estimates of $f(i,1)$ and $f(i,0)$ are respectively given by $\hat{f}(i,1) = \frac{M_p}{M}$ and $\hat{f}(i,0) = \frac{M_n}{M}$. By a straightforward application of the Hoeffding Bound we can say that one needs $O(1/\epsilon^2 \log(1/\delta))$ random runs to achieve $|\hat{f}(i,1) - f(i,1)| \leq \epsilon$ with probability at least $1 - \delta$. Lets say we also want to obtain estimates $\hat{g}$ of the conditional probability g.

$$\hat{g}(i,1) = \frac{\hat{f}(i,1) + \lambda}{\hat{f}(i,1) + \hat{f}(i,0) + 2\lambda}$$

If $\hat{f}(i,y) \in [f(i,y) - \epsilon, f(i,y) + \epsilon]$, for $y \in \{0,1\}$ with high probability then a simple algebraic manipulation gives: $\hat{g}(i,y) \in [g(i,y) - \epsilon', g(i,y) + \epsilon']$, w.h.p. where $\epsilon' = \epsilon/(f(i,0) + f(i,1) + 2\lambda)$. In order to use this for getting an intuition about how big λ should be, we notice that λ needs to be at least $0.5 * (1 - f(i,0) - f(i,1)))$ in order to obtain $\epsilon' = \epsilon$. Note that $1 - f(i,0) - f(i,1)$ is simply the probability that a random walk from i will not hit any label in T steps.

5.2 The Branch and Bound Algorithm

Computing functions over a graph using neighborhood expansion has been used by [15] for computing nearest neighbors in hitting time to a node, or for efficient computation of personalized pagerank ([5]). We use a technique similar to the first approach to compute top k nodes in f values.

Here is the main intuition behind the algorithm. We maintain a neighborhood around the labeled points. At any time we compute a lower ($fl^T(i,1)$) and upper ($fu^T(i,1)$) bound of $f(i,1)$ for all nodes i in this neighborhood. We characterize $f(i,1)$ for all i outside the neighborhood by a single lower and upper bound. This global lower bound is zero. Let the global upper bound be x. As we expand the neighborhood, these bounds keep getting tighter, and x keeps decreasing. We stop expansion when x falls below a small constant α. We will first describe the lower and upper bounds and then present the algorithm and the neighborhood expansion scheme. We have:

$$f^T(i,1) = \begin{cases} 1 & \text{If } i = + \\ 0 & \text{If } i = - \\ 0 & \text{If } T = 0 \\ \sum_j P(i,j) f^{T-1}(j,1) & \text{Otherwise} \end{cases} \tag{4}$$

The fastest way of hitting a positive node before a negative node from outside the neighborhood is achieved by magically jumping to the boundary node which has the maximum probability to hit a positive node before a negative node. Also from equation (4) note that the only case where we need to use this is when any of the direct outgoing neighbors of i is outside the neighborhood.

Since we are computing the neighborhood for $f^T(i,y)$ let us denote the neighborhood as S_y, for $y \in \{0,1\}$. Let $nout(i)$ be the set of direct outgoing neighbors of node i. Let $nin(i)$ be the direct incoming neighbors of i. Let $\delta(S_y)$ be the boundary of the neighborhood S_y, i.e. it contains all nodes in S_y which has at least one incoming neighbor outside S_y. Hence the T-step upper bound is defined as

$$\begin{aligned} fu^T(i,y) = & \sum_{j \in nout(i) \cap S_y} P(i,j) fu^{T-1}(j,y) \\ & + (1 - \sum_{j \in nout(i) \cap S_y} P(i,j)) \max_{m \in \delta(S_y)} fu^{T-2}(m,y) \end{aligned}$$

For the lower bound, we just use the fact that the lowest probability to hit a positive node before a negative one from outside the neighborhood is simply zero. This gives rise to

$$fl^T(i,y) = \sum_{j \in nout(i) \cap S_y} P(i,j) fl^{T-1}(j,y)$$

The global upper bound $ub^T(S_y)$ is simply defined as the maximum probability to reach label y before any other label from a node on the boundary of the neighborhood.

$$ub^T(S_y) = \max_{m \in \delta(S_y)} fu^{T-1}(m,y)$$

Here is the pseudo-code of the algorithm

```
1. Initialize $S_y = \bigcup_{i \in L} nin(i)$.
2. $\forall i \in S_y$ compute $fl^T(i,y)$, $fu^T(i,y)$
3. Compute $ub^T(S_y)$.
4. Expand $S_y$. (Details are provided below)
5. Stop if $ub^T(S_y) \le \alpha$. Else reiterate from step 2.
```

By definition all nodes i outside S_y have $f^T(i,y) \le ub^T(S_y) \le \alpha$. Hence we have pruned away all nodes with very small probability of hitting a positive before a negative.

Now how do we expand a neighborhood in step 4? We use the heuristic of picking the k nodes on the boundary with k largest fu^{T-1} values, and augment the boundary by adding their incoming neighbors to S_y. The intuition is that since all bounds use the maximum probability of hitting label y before any other label in $T-1$ steps from a boundary node, augmenting the neighborhood around these points will probably tighten the bounds more.

The ranking step to obtain top k nodes using upper and lower bounds is simple: we return all nodes which have lower bound greater than the $k+1^{th}$ largest upper bound (when $k = 1$, kth largest is the largest probability). We denote this as fu_{k+1}. Since all nodes which have probability greater than α to hit a positive node before a negative node are guaranteed to be in the neighborhood expanded, we know that the true $(k+1)^{th}$ largest probability will be smaller than fu_{k+1}. Hence any node with lower bound greater than fu_{k+1} is guaranteed to be greater than the $k+1^{th}$ largest probability. We use a multiplicative slack, e.g. ($fu_{k+1}(1-\epsilon)$) in order to return the top k *approximately* large probability nodes. In our algorithm we initialize α with a large value and keep decreasing it until the bounds are tight enough to return k largest nodes. Note that one could rank the probabilities using the lower bounds, and return top k of those after expanding the neighborhood a fixed number of times. This will only mean a larger approximation slack.

In this section we presented a branch and bound algorithm to compute top k nodes in the unconditional probability to hit a positive label before a negative label. How do we compute the conditional probability?

Given the upper and lower bounds $\{fu(i,0), fl(i,0)\}$ and $\{fu(i,1), fl(i,1)\}$ the upper and lower bounds on the conditional probabilities can be given by:

$$\begin{aligned} gu(i,1) &= \frac{fu(i,1) + \lambda}{fu(i,1) + fl(i,0) + 2\lambda} \\ gl(i,1) &= \frac{fl(i,1) + \lambda}{fl(i,1) + fu(i,0) + 2\lambda} \end{aligned} \tag{5}$$

Currently we obtain bounds on $f(i,y)$, for $y \in \{0,1\}$, and use equation (5) for obtaining the conditional probabilities. Doing neighborhood expansion for the conditional probabilities is part of ongoing work.

5.3 Number of nodes with high f value

How many nodes in the neighborhood will have probability greater than α? Its hard to answer this question for a general directed graph. So we will present a result for an undirected graph. Let S be the set of nodes with $f \ge \alpha$. Also let L_p be the set of nodes with a positive label. We prove (appendix) the following for an undirected graph.

THEOREM 5.1. *In an undirected graph, the size of set $S = \{i | f^T(i) \ge \alpha\}$ is upper bounded as: $|S| \le \frac{\sum_{p \in L_p} d(p)}{min_{i \in S} d(i)} \frac{T}{\alpha}$, where L_p is the set of nodes with a positive label.*

This implication of the above theorem is interesting. The upper bound on S is proportional to the total degree of all the positive labels and inversely proportional to the quantity $min_{i \in S} d(i)$, which is the minimum degree of nodes in S. This means, if the positive nodes have a higher degree relative to the neighboring nodes then it is easy to reach a positive node. As we will point out in section 6.6 this brings us directly to an active learning question: how to select a good set of nodes to be labeled?

6. RESULTS

In this section we present results on the DBLP graph. The graph has a similar schema as in figure 1. The original DBLP graph has about $200,000$ words, $900,000$ papers and $500,000$ authors. We will make this graph publicly available. There are about 10 million edges. We will mention the total number of undirected edges. As we will point out later, for each edge we have links in both directions with different weights. We present our results on two graphs built from this corpus:

1. **Two-layered graph**: We exclude words from the graph representation. Most citation-database related work [3, 5] use this representation, where the words are used only in an inverted index pointing to the document originally containing those. This graph has around $1.4M$ nodes and $2.2M$ edges.
2. **Three-layered graph**:We include all the words which occur more than 20 times and less than $5,000$ times in our graph representation. There are $15,000$ such words. This has around $6M$ edges and 1.4 M nodes. Note that the extra number of edges from $15,000$ word nodes is about $3.7M$.

The links between the paper and author layer is undirected. For citation links within the paper layer, a link from a paper x to paper y citing x, is assigned one-tenth of the weight on the link from y to x. This is very similar to the weighing convention of [5]. In the three layer graph-representation, 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. The details can be found in [16]. We set $W = 1$; $A = 10$ and $P = 10$ so that the word layer to paper layer connection is almost directed. The leaf nodes are augmented with a self loop, with the same weight as its single edge.

6.1 Experimental Settings

In this paper we have proposed a scheme to rerank search results based on user feedback. However it is hard to do fair evaluation of user-feedback based ranking. Hence we created an automated way to mimic user-feedback. In DBLP two authors with the same name are often listed as the same record. We use entity disambiguation as a task to evaluate the predictive performance of a measure. Here are the steps:

1. Pick 4 authors having a common surname.
2. Merge these authors in the same node.
3. Rank other nodes using proximity measure $\mathfrak{h}$ from the merged node.
4. Label the top L papers on this list using ground truth.
5. The testset consists of all papers written by these authors modulo the ones already labeled.
6. Compute different measures (e.g. conditional probability to hit a positive label before a negative label, etc) for each testnode and then compute AUC score against the ground truth.

We pick about 20 common surnames. Note that for each surname there are 4 different authors and hence 4 different one-vs-all classification scenarios. However sometimes the top L papers might belong to the same author if he is relatively more prolific than the others. Removing all those cases we had about $30-40$ points in our testset, which gives about 2 classifications per surname. For each surname we compute the mean AUC score, and then report the mean over all the surnames. We also apriori fixed the testset so that we can compare different experiments over the same graph, and same proximity measure $\mathfrak{h}$. The different experiments were on varying values of the truncation parameter T and number of labels L. We picked truncated hitting time [15] as the proximity measure ($\mathfrak{h}$) from the merged node, since its easy to compute and also is very similar to Personalized Pagerank (PPV). The hitting time from a node x to node y is the expected time to hit y for the first time in a random walk starting from node x. For the truncated version only paths of length less than T are considered. We hand-tuned the smoothing parameter λ to be $1e-4$ for all our experiments. We also used $2,500$ samples for the sampling algorithm.

We will describe the results on the two and three-layered graphs. Note that although experiments with varying number of labels and varying values of T are conducted on the same testset across a graph, the testsets are slightly different for two different graphs. The purpose of showing results on both is to examine the general predictive behavior of the different measures on the two different graph topologies. In section 6.6 we show how the choice of the original proximity function $\mathfrak{h}$ affects the performance of the different measures. Also in this case the testsets are different, since the ranklists generated from different choices of $\mathfrak{h}$ are different.

We compare with two alternative measures of proximity from only the positive labels. Note that this is a similar idea to TrustRank. We used 10-truncated hitting time (section 6.3) and PPV (section 6.4) with teleportation value 0.1 for our experiments. For all experiments other than section 6.2 we used $T = 10$ for computing conditional or unconditional probabilities to hit a positive node before a negative.

6.2 Effect of T

In this section we describe the effect of parameter T on the AUC scores. The results are described in figure 4. Figure 4A. contains the results on the two-layered graph, whereas figure 4B. contains the results on three-layered graph which contains the words as well, i.e. information can pass through common words in the graph. We consider the values of T in $\{1, 3, 10, 30, 1000\}$. For $T = 1$ the information can only come from direct neighbors. Since both our testset and labeled node-set consists of papers there is a very slight chance of hitting a labeled node via citation. Still in both cases the AUC scores are close to random. Another interesting behavior is that $T = 3$ and $T = 10$ perform comparably well and much better than random in both graphs. However in the two-layered graph, with 10 labeled nodes varying T from 3 to 1000 does not give much boost in performance. However in the graph with the word layer, from $T = 10$ to $T = 30$ there is about a 10% increase in AUC score. This is expected since in a graph with a word layer it takes longer time to hit a small set of labeled nodes. Also, note that the AUC scores increase only slightly as T is increased to 1000.

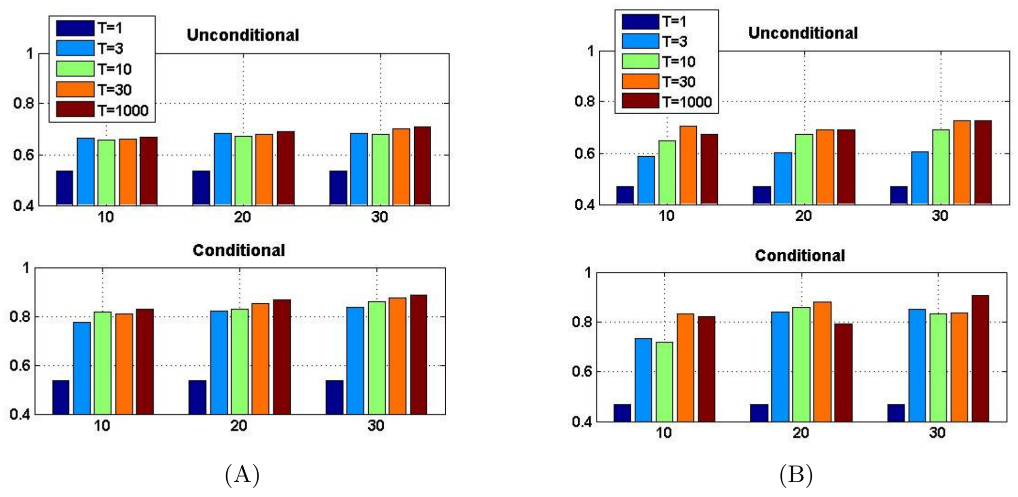


Figure 4: Experiments on A) The two-layered graph and B) The three-layered graph. The y axis has the AUC scores and the x axis has different number of labels. The different bars from left to right are for different values of T: 1,3,10,30 and 1000, respectively.

6.3 Comparison with different algorithms

In figure 5 we compare the unconditional and conditional probabilities with an alternative diffusion based proximity measure, namely the hitting time from the positive nodes. We use $T = 10$ for these experiments. From figure 4 it can be observed that other values of T have better AUC score than $T = 10$. However we fix $T = 10$ as a reasonable truncation parameter. Note that in both the two and three layered graphs the conditional measure outperforms the hitting time. However the unconditional probability performs comparably. This might be because the conditional measure successfully uses the information from both the positive and the negative labels. Also note that for the three-layered graph for 10 labels the difference between the conditional and the hitting time is the smallest. This is because of the fact that in presence of the word nodes it takes more time to hit a relatively smaller set of labeled nodes. For $T = 20$ and $T = 30$ the the conditional probability outperforms $T-$truncated hitting time by a large margin.

6.4 Comparison with PPV

So far we have compared our algorithms only against proximity measures like hitting time. In this section (figure 6) we will also present the performance of PPV with the start distribution uniform over the positive nodes. We only compare the conditional probability ($T = 10$) with PPV (restart probability 0.1). The results are similar to those of the last section. For the three-layered graph and 10 labels PPV and conditional probabilities perform comparatively well. However for all other cases PPV is outperformed by the former.

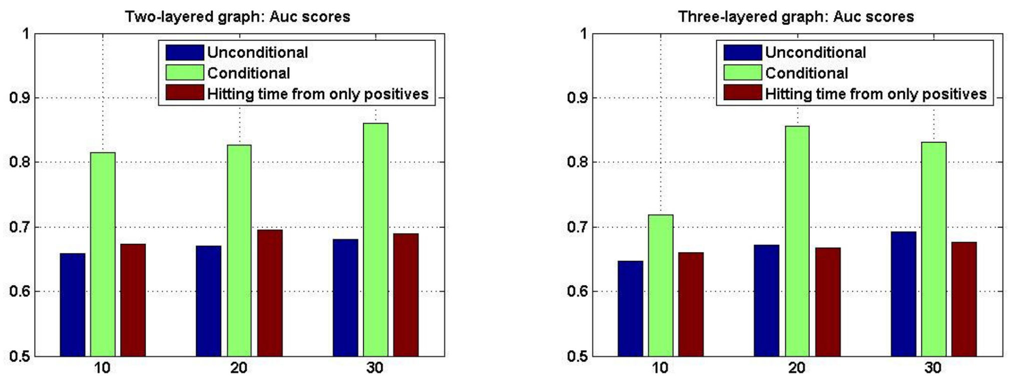
6.5 Sampling vs Branch and Bound

In all our experiments we have a small set of candidate nodes to be ranked, and hence we used sampling to compute the conditional and unconditional probabilities. In this section we demonstrate the scenario where we want to rank a larger set of nodes. In table 2 we present the average timing results for the branch and bound algorithm (BB) vs. the sampling algorithm (Samp) on around 15 disambiguation tasks with 10 labels each. The experimental setting is exactly the same as before. For each task we carry out sampling for $1,000$ candidate nodes. The timing provided for branch and bound is the average time in seconds to compute k nearest neighbors in $f(i, 0)$ and $f(i, 1)$. We also present the average number of neighbors (3-Hops) within 3 hops of the labeled nodes. This is to illustrate the growth properties of the two graphs. Note that the average time for branch and bound increases by a factor of 20 whereas that of sampling increases by a factor of 3. The number of edges in the three-layered graph is about 3 times as large as the two-layered one, whereas the number of nodes stay roughly the same. This implies that the average degree becomes about 3 times as large. Time for sampling is directly proportional to the average degree, which explains the increase in the time required for sampling.

Table 2: Timing results for Branch and Bound (BB) and Sampling (Samp) for 1000 candidate nodes. We also report the average number of nodes within 3 hops of the labeled nodes.

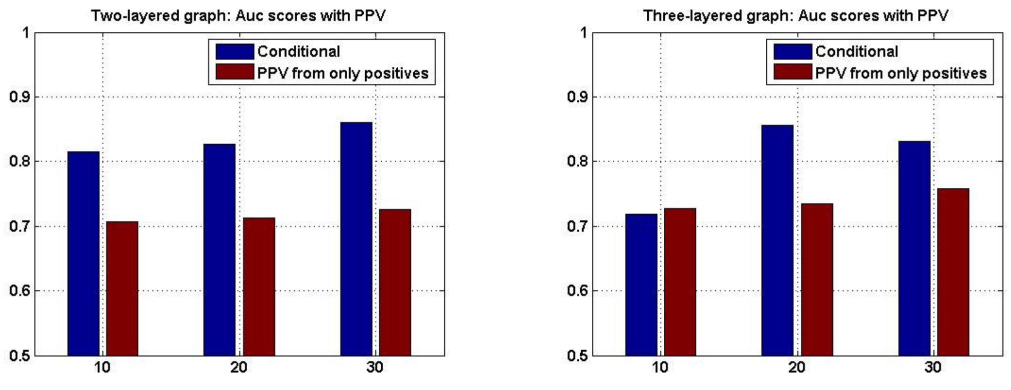
	Two-layered			Three-layered		
k	BB(s)	Samp(s)	3-Hops(#)	BB(s)	Samp(s)	3-Hops(#)
10	1.6	90	2,000	37	283	160,000
30	3			62		

The time increase for branch and bound can be explained by the numbers in the fourth and seventh columns (3-Hops(#)). The size of the neighborhood expanded depends on the growth rate of the graph. The total number of nodes within 3 hops of all the labeled nodes grow by a factor of around 80 from the two-layered to the three-layered representation. Although the neighborhood computed via branch and bound does not contain all nodes within 3 hops, it's size still grows faster in the three layered graph than in the two-layered one.



(A) (B)

Figure 5: Experiments on A) The two-layered graph and B) The three-layered graph. The y axis has the AUC scores and the x axis has different number of labels. The bars from left to right are for different measures, *unconditional*, *conditional*, and 10-*truncated hitting time from only positives.*, respectively.



(A) (B)

Figure 6: Experiments on A. The two-layered graph and B. The three-layered graph. The y axis has the AUC score and the x axis has different number of labels. From left to right the bars are for two measures, respectively *conditional*, and *Personalized Pagerank from the Positive Labels.*

6.6 Effect of the original ranking function $\mathfrak{h}$

In this section we show the effect of the original ranking function from the merged node. So far we have only considered the nodes ranked by the hitting time from the merged author node. In a web-search setting the user will be asked to label these nodes. Note that this is an active learning heuristic. In [18] the authors mention three desirable properties of an effective active learning strategy for relevance feedback in information retrieval. Let us denote the set of feedback nodes as F. F should be:

1. Relevant to the query.
2. Diverse, in order to avoid redundancy.
3. Selected from a high density region of the data, so that the query-feedback algorithm has more information.

By picking the labeled nodes from the top ranked documents we definitely fulfil the first criteria. In this section we will show that picking hitting time as the original retrieval algorithm automatically satisfies the third criterion.

In order to illustrate this we will pick truncated commute time as the ranking function. The commute time between two nodes x, y is defined as the expected time to hit node y from node x for the first time and come back. The truncated version is simply the sum of the truncated hitting time from x to y and y to x. We only present the results on the two-layered graph for simplicity. Also the comparison is not on the exact same testset, since the ranklist for the authors change from one measure to another. Note that the performance of the ranklist based on commute times for 10 labels is close to random. Recall the result in theorem 5.1. We showed that in an undirected graph, if the positive nodes have higher relative degree compared to the neighboring nodes, then the number of nodes which can reach a positive label before a negative label increases. This intuition also helps pick an appropriate active learning strategy for a directed graph. The hitting time from a merged node to high degree nodes are generally small, since a random walk is more probable to hit a high degree node. Hence the labeled set picked from the top-ranked nodes will also be easily reachable from other nodes. On the other hand

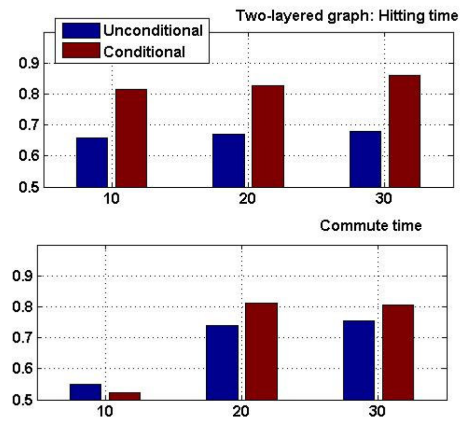


Figure 7: The top panel consists of the results when the original ranklist was based on Hitting time from the merged node. The bottom panel shows the results where the ranklist was created based on the Commute time from the merged node. The y axis has the AUC scores for *unconditional* and *conditional* probabilities (bars from left to right) and the x axis has different number of labels.

commute time between nodes x and y is small if degree of y is small, since that reduces the expected return time to x. As a result the top 10 labels in the commute-time based rank list can be harder to reach from other nodes. This is reflected in the results presented in figure 7. Note that for 20 or 30 labels this difference is not so noticeable as for 10 labels. This is because as we increase the labeled set size, probability of reaching some label increases.

7. CONCLUSION

In this paper we have presented an effective measure for re-ranking nodes in a graph based on user feedback. Using quantifiable tasks on the entire DBLP corpus we have shown that practical diffusion-based measures which use both positive and negative feedback outperform those which only use positive relevance feedback.

8. ACKNOWLEDGMENTS

The authors are grateful to Anupam Gupta, and Xiaojin Zhu for ideas and discussions. We thank Andy Schlaikjer for sharing the DBLP dataset. This research was partially supported by U.S. Dept. of Homeland Security (DNDO CFP06-TA03-LL15).

9. REFERENCES

[1] J. Abernethy, O. Chapelle, and C. Castillo. Web spam identification through content and hyperlinks. In *Proc. AIRWEB*, 2008.
[2] D. Aldous and J. A. Fill. *Reversible Markov Chains.* 2001.
[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004.
[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. WWW*, 1998.
[5] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, 2007.
[6] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 2003.
[7] L. Grady. Random walks for image segmentation. *PAMI*, 2006.
[8] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *Proc. VLDB*, 2004.
[9] G. Jeh and J. Widom. Scaling personalized web search. In *Stanford University Technical Report*, 2002.
[10] R. Jin, H. Valizadegan, and H. Li. Ranking refinement and its application to information retrieval. In *WWW*, 2008.
[11] A. Joshi, R. Kumar, B. Reed, and A. Tomkins. Anchor-based proximity measures. In *Proc. WWW*, 2007.
[12] I. Koutis and G. L. Miller. A linear work, $o(n^{1/6})$ time, parallel algorithm for solving planar laplacians. In *Proc. SODA*, 2007.
[13] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. *ACM Transactions on Graphics*, 2004.
[14] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *Proc. WWW*, 2006.
[15] P. Sarkar and A. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *Proc. UAI*, 2007.
[16] P. Sarkar, A. W. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *ICML*, 2008.
[17] D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the STOC'04*, 2004.
[18] Z. Xu, R. Akella, and Y. Zhang. Incorporating diversity and density in active learning for relevance feedback. In *ECIR*, 2007.
[19] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML, volume 20*, 2003.

APPENDIX

Proof of Theorem 5.1: Denote the probability of an event by $Pr\{.\}$, and the transition matrix by P. $d(i)$ denotes the weighted degree of a node in an undirected graph. Define $S = \{i|f^T(i) \geq \alpha\}$, and L_p as the set of positive labels. We have,

$$\begin{aligned}
f^T(i) &= Pr_i(\text{Hitting a '+' before a '}-\text{' in } T \text{ steps}) \\
&\leq Pr_i(\text{Hitting a '+' in } T \text{ steps}) \\
&= \sum_{t=1}^{T} Pr_i(\text{Hitting a '+' for the first time in exactly } t \text{ steps}) \\
&\leq \sum_{t=1}^{T}\sum_{p\in L_p} Pr_i(\text{Hitting } p \text{ in exactly } t \text{ steps}) \\
&\leq \sum_{t=1}^{T}\sum_{p\in L_p} P^t(i,p) = \sum_{t=1}^{T}\sum_{p\in L_p} \frac{d(p)}{d(i)} P^t(p,i) \qquad (6)
\end{aligned}$$

The final step of equation (6) is due to a straight-forward use of the reversibility of a random walk in an undirected graph [2]. Now we will upper bound the total probability of hitting a positive label before a negative label from the set of nodes S.

$$\begin{aligned}
\sum_{i\in S} f^T(i) &\leq \sum_{i\in S}\sum_{t=1}^{T}\sum_{p\in L_p} \frac{d(p)}{d(i)} P^t(p,i) \\
&\leq \frac{1}{min_{i\in S} d(i)} \sum_{t=1}^{T}\sum_{p\in L_p} d(p) \sum_{i\in S} P^t(p,i) \\
&\leq \frac{\sum_{p\in L_p} d(p)}{min_{i\in S} d(i)} T
\end{aligned}$$

Combining the definition of S and the above equation we get:

$$\alpha|S| \leq \frac{\sum_{p\in L_p} d(p)}{min_{i\in S} d(i)} T \rightarrow |S| \leq \frac{\sum_{p\in L_p} d(p)}{min_{i\in S} d(i)} \frac{T}{\alpha}$$