2008 Eighth IEEE International Conference on Data Mining

# Measuring Proximity on Graphs with Side Information

Hanghang Tong
Carnegie Mellon University
htong@cs.cmu.edu

Huiming Qu
IBM T.J. Watson
hqu@us.ibm.com

Hani Jamjoom
IBM T.J. Watson
jamjoom@us.ibm.com

## Abstract

*This paper studies how to incorporate side information (such as users' feedback) in measuring node proximity on large graphs. Our method (ProSIN) is motivated by the well-studied random walk with restart (RWR). The basic idea behind ProSIN is to leverage side information to refine the graph structure so that the random walk is biased towards/away from some specific zones on the graph. Our case studies demonstrate that ProSIN is well-suited in a variety of applications, including neighborhood search, center-piece subgraphs, and image caption. Given the potential computational complexity of ProSIN, we also propose a fast algorithm (Fast-ProSIN) that exploits the smoothness of the graph structures with/without side information. Our experimental evaluation shows that Fast-ProSIN achieves significant speedups (up to 49x) over straightforward implementations.*

## 1 Introduction

Measuring proximity (i.e., relevance/closeness) between nodes on large graphs is a very important aspect in graph mining and has many real applications in ranking, anomaly nodes indentification, connection subgraphs, pattern matching, etc. Despite the successes of many previous work, most existing proximity measurements only consider the link structure of the underlying graph, ignoring any possible side information. For example, given an author-conference bipartite graph, existing proximity measurements may answer the question: *What are the most similar conferences to KDD?* However, for a particular user, s/he might have her/his own preferences: *I dislike ICML or I like SIGIR*. These preferences are typically localized to a particular search, and may not reflect a global sentiment by the user.

There are a wide range of scenarios where users' feedback, both implicit or explicit, can be naturally integrated as side information. For instance, in recommendation systems, side information could be users' ratings on items (e.g., *I like Kung-Fu Panda*). In Blog analysis, it could be opinions and sentiments. Additionally, for many real applications, users' preferences can be estimated from click-through data. That said, it is thus important to incorporate such side information in the proximity measurement so that search results are well-tailored to reflect a user's individual preferences. In the earlier example, the question will then become: *What are the most similar conferences to KDD, but dissimilar to ICML?*

In this paper, we address the above challenge by proposing a novel method, called ProSIN, that incorporates such like/dislike side information in measuring node proximity on large graphs. Our method is based on random walk with restart (RWR), where ProSIN uses the side information to refine the graph structure so that RWR is biased to avoid or to favor some specific zones on the graph according to the users' preferences. Additionally, ProSIN inherits existing capabilities from RWR, such as the ability to summarize the *multiple faceted* relationships, to be interpreted from the perspective of *steady-state probability*, etc. Therefore, we expect ProSIN to enrich a broad-range of applications by replacing their original proximity measurement implementation. We evaluate ProSIN in three case studies: neighborhood search, center-piece subgraph, and image caption. In all cases, we show that ProSIN naturally reflects the users' preference and/or improves the quality of the existing applications (e.g., boost the precision/recall of the image captions by more than 10%).

Because a straightforward implementation of ProSIN requires significant computation, we propose a fast algorithm (Fast-ProSIN) that computes the proposed proximity measurement, while radically reducing the computational overhead. Fast-ProSIN achieves the performance gains by exploiting the smoothness of the graph structures with/without side information. Our experimental results show that it achieves significant speedup (*up to 49x*) while maintaining high approximation accuracy (more than 93.0%).

The paper has three key contributions:

- A novel method (ProSIN) to incorporate side information (like/dislike) in measuring node proximity on large graphs, enriching a broad range of applications;
- A fast algorithm (Fast-ProSIN) to compute the proposed proximity measurement, achieving significant speedups (up to 49x);
- Extensive experimental evaluations on several real datasets.

1550-4786/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDM.2008.42

**Table 1. Symbols**

| Symbol | Definition and Description |
|---|---|
| $\mathbf{A}, \mathbf{B}, \ldots$ | matrices (bold upper case) |
| $\mathbf{A}(i,j)$ | element at the $i^{\text{th}}$ row and $j^{\text{th}}$ column of $\mathbf{A}$ |
| $\mathbf{A}(i,:)$ | $i^{\text{th}}$ row of matrix $\mathbf{A}$ |
| $\mathbf{A}(:,j)$ | $j^{\text{th}}$ column of matrix $\mathbf{A}$ |
| $\mathbf{a}, \mathbf{b}, \ldots$ | column vectors |
| $\mathcal{I}, \mathcal{J}, \ldots$ | sets (calligraphic) |
| $n$ | number of nodes in the graph |
| $n^i$ | number of out links of node $i$ |
| $c$ | $(1-c)$ is the restart probability |
| $r_{i,j}$ | proximity from node $i$ to node $j$ |
| $\mathbf{r}_i = [r_{i,j}]$ | ranking vector for node $i$ $(j = 1, ..., n)$ |
| $\mathcal{P}$ | positive set $\mathcal{P} = \{x_1, ..., x_{n^+}\}$ |
| $\mathcal{N}$ | negative set $\mathcal{N} = \{y_1, ..., y_{n^-}\}$ |
| $n^+$ | number of positive nodes $n^+ = \lvert\mathcal{P}\rvert$ |
| $n^-$ | number of negative nodes $n^- = \lvert\mathcal{N}\rvert$ |
| $\mathbf{e}_i$ | $n \times 1$ starting vector for node $i$, where $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0 (j \neq i)$ |

The rest of the paper is organized as follows. We introduce notations and formally define the problem in Section 2. We present the proposed proximity measurement in Section 3 and the fast algorithm in Section 4, respectively. We provide experimental evaluations in Section 5 and review the related work in Section 6. Finally, we conclude in Section 7.

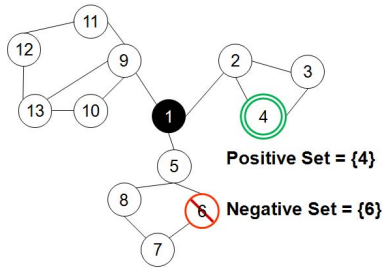
## 2 Problem Definitions

Table 1 lists the main symbols that we use throughout the paper. We represent a general graph by its adjacency matrix. Following the standard notation, we use capital letters for matrices (e.g. $\mathbf{A}$), lower case for vectors (e.g. $\mathbf{a}$), and calligraphic fonts for sets (e.g. $\mathcal{I}$). We use the symbol "˜" to distinguish the setting with/without side information. For example, $\mathbf{A}$ is the normalized adjacency matrix of the graph without side information; and $\tilde{\mathbf{A}}$ is the normalized adjacency matrix of the refined graph by side information.

We represent the elements in a matrix using a convention similar to Matlab, e.g., $\mathbf{A}(i,j)$ is the element at the $i^{\text{th}}$ row and $j^{\text{th}}$ column of the matrix $\mathbf{A}$, and $\mathbf{A}(:,j)$ is the $j^{\text{th}}$ column of $\mathbf{A}$, etc.

We use a running example, depicted in Fig.1(a), to describe the problem statement. There, each node represents a person (e.g., node 1 is 'John', node 2 is 'Smith', etc.) and the existence of edge represents some social contact between the two corresponding persons (e.g., phone call). In traditional settings of proximity measurement, the goal is to quantify the closeness (i.e., relevance) between two nodes (the source and target) based on the link structure of the underlying graph. In our settings, we assume the existence of side information, focusing primarily on like/dislike user feedback as side information. In our running example, a user might not want to see (i.e., dislike) node 6 but favors (i.e., like) node 4.

Formally, we represent such side information by two sets $\mathcal{P}$ and $\mathcal{N}$. The set $\mathcal{P}$ contains the node indices that users like (referred to as the positive set), where the corresponding nodes are referred as positive nodes. The set $\mathcal{N}$ contains the node indices that users dislike (referred as negative set), where the corresponding nodes are referred to as negative nodes. In our running example, both the positive set $\mathcal{P}$ and the negative set $\mathcal{N}$ contain one single element, respectively: $\mathcal{P} = \{4\}$ and $\mathcal{N} = \{6\}$. Our goal is to incorporate such side information to measure the node proximity (e.g., the proximity from node 1 to the node 3 in our running example).




(a) the graph (node 1 is the source.)

$$\begin{pmatrix}
0 & 0.33 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 \\
0.33 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.33 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0 & 0.5 & 0
\end{pmatrix}$$

(b) column normalized adjacency matrix $\mathbf{A}$

**Figure 1. The running example.**

With the above notations and assumptions in mind, our problem can be formally defined as follows:

**Problem 1 (Proximity with Side Information)**

**Given:** *a weighted direct graph* $\mathbf{A}$*, a source node* $s$ *and a target node* $t$*, and side information* $\mathcal{P}$ *and* $\mathcal{N}$*;*

**Find:** *the proximity score* $\tilde{r}_{s,t}$ *from source node* $s$ *to target node* $t$*.*

In problem 1, if the target node $t$ is absent, we measure the proximity score $\tilde{r}_{s,i} (i = 1, ..., n)$ from the source node $s$ to all the other nodes in the graph. If we stack all these scores into a column vector $\tilde{\mathbf{r}}_s = [\tilde{r}_{s,i}](i = 1, ..., n)$, it is equivalent to saying that we want to compute the ranking vector $\tilde{\mathbf{r}}_s$ for the source node $s$. In this paper, we assume that there is no overlap between the positive set and negative

set (i.e., $\mathcal{P} \cap \mathcal{N} = \phi$.[1]) Also, the positive and negative side information do not need to exist simultaneously. For example, if we only have positive side information, we can simply set the negative set to be empty (i.e., $\mathcal{N} = \phi$).

# 3 ProSIN

In this section, we introduce our proximity measurement with side information, ProSIN. We begin by reviewing random walk with restart (RWR), which is a good proximity measurement for the case where there is no side information. We, then, extend RWR to properly account for side information.

## 3.1 RWR: Proximity without Side Information

Random walk with restart (RWR) is considered one of the most successful methods for measuring proximity and is receiving increased interest in recent years—see Section 6 for a detailed review. For a given graph, RWR is defined as follows. Consider a random particle that starts from node $i$. The particle iteratively transits to its neighbors with probabilities proportional to the corresponding edge weights. At each step, the particle can returns to node $i$ with some restart probability $(1-c)$. The proximity score from node $i$ to node $j$ is defined as the steady-state probability $r_{i,j}$ that the particle will be on node $j$ [18]. Intuitively, $r_{i,j}$ is the fraction of time that the particle starting from node $i$ will spend on each node $j$ of the graph, after an infinite number of steps.

If we stack all the proximity scores $r_{i,j}$ into a column $\mathbf{r}_i$ (referred to as the ranking vector for the node $i$), the equation (1) gives the formal definition of RWR:

$$\mathbf{r}_i = c\mathbf{A}\mathbf{r}_i + (1-c)\mathbf{e}_i, \tag{1}$$

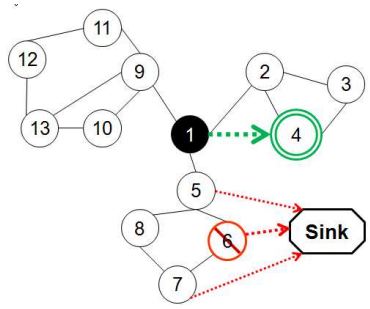
where $\mathbf{A}$ is the column normalized adjacency matrix for the graph and $\mathbf{e}_i$ is the starting vector for node $i$.

For our running example in Fig. 1(a), its normalized adjacency matrix $\mathbf{A}$ is shown in Fig. 1(b). If we ignore any side information, by setting the correct starting vector (e.g., $\mathbf{e}_1 = [1,0,0,0,0,0,0,0,0,0,0,0,0]'$ for node 1), we can solve the corresponding ranking vector using equation (1). Fig. 2(a) plots the ranking vector (sorted from highest to lowest) for node 1 of the running example. The scores are consistent with our intuition: nearby nodes (e.g., nodes 9, 2 and 5) receive higher proximity scores.

## 3.2 ProSIN: Proximity with Side Information

**Basic Ideas.** Our goal is to incorporate side information to measure the node proximity. Intuitively, for a given source node $s$, if positive nodes exist, the proximity score from the source node to such positive nodes as well as their neighboring nodes should increase, compared to the case where such side information is unavailable. In our running example, if we know node 4 belongs to the positive set $\mathcal{P}$, we expect that the proximity score from the source node 1 to node 4 to increase and so will the proximity scores from node 1 to node 4's neighboring nodes (e.g., node 2 and node 3). Analogously, if negative nodes exist, the proximity scores from the source node to such negative nodes as well as their neighboring nodes should decrease, compared to the case where such side information is unavailable. In our running example, if we know that node 6 belongs to the negative set $\mathcal{N}$, we expect the proximity score from node 1 to node 6 to decrease, and so will node 6's neighboring nodes (such as nodes 5 and 7).

The basic idea of ProSIN is then to use side information to refine the original graph structure so that the random particle (1) has higher chances of visiting the positive nodes as well as their neighboring nodes, and (2) has lower chances of visiting the negative nodes as well as their neighboring nodes.




(a) the updated graph

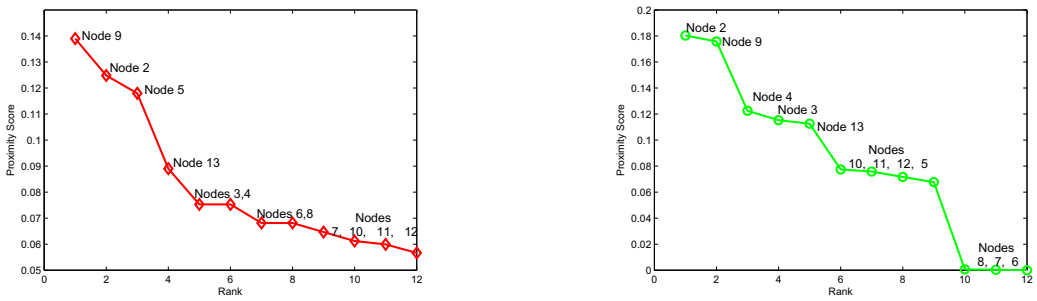
$$\begin{pmatrix}
0 & 0.33 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 \\
0.25 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.25 & 0.33 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.01 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.01 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0 & 0.5 & 0
\end{pmatrix}$$

(b) updated column normalized adjacency matrix

**Figure 3. Adjustment on the original graph in the running example in Fig. 1.**

**Dealing with Positive Nodes.** For each node $x$ in the positive set ($\mathcal{P}$), we create a direct link from the source node $s$ to node $x$. As in the running example, we add a direct link from the source node 1 to node 4 (See Fig. 3(a)). In this way, whenever the random particle visits (or restarts from) the source $s$, it has higher chances of visiting the nodes in the positive set. Note that we are also implicitly increasing the

[1] If this does not hold, we can remove the intersection from both positive set and negative set.




(a) without side information (b) with side information

**Figure 2. Ranking vector for node 1 in the running example in Fig. 1. (The proximity scores are normalized so that they sum up to 1.)**

chance that the random particle will visit the neighborhood of those positive nodes. The weight of each newly added link is set to $1/(n^s + n^+)$. For example, the newly added edge (1,4) for the running example will receive a weight of 0.25 (since $n^1 = 3$ and $n^+ = 1$).

**Dealing with Negative Nodes.** To deal with the negative nodes, we introduce a sink into the graph, which has no out links. For each node $y$ in the negative set ($\mathcal{N}$), we put a direct link from node $y$ to the sink. Thus, whenever the random particle visits this node, it can go to the sink and never comes back (since there is no out links from the sink). Therefore, this negative node $y$ is penalized and its corresponding proximity score will decrease. In order to penalize the neighborhood of node $y$, we also put a direct link from its neighboring nodes to the sink. In our running example, besides the link from node 6 (the negative node) to the sink, we placed a link from nodes 5 and 7 (the neighboring nodes of node 6) to the sink respectively (see Fig. 3(a)).

There are two remaining questions: (1) how to choose the neighborhood of a negative node $y$ and (2) how to determine the weights to the sink. Let the index of the sink node be $n + 1$, the procedure is summarized in Alg. 1. In Alg. 1, we use random walk with restart (on the original graph) to determine (1) the neighborhood of the negative node $y$ (steps 2-4), and (2) the weights of the newly added links to the sink (steps 5-6). Notice that we eventually (step 9) discard the last row/column (which corresponds to the sink node). We use it to simplify the description of the proposed method without affecting the ranking vector in accord to the property of a sink node.

**ProSIN Algorithm.** Based on the above preparations, the complete algorithm to measure proximity with side information (ProSIN) is given in Alg. 2. In Alg. 2, after initialization (step 1), we first use side information to refine the graph structure (steps 2-7 for positive nodes,[2] and steps 8-12 for negative nodes). Note that in step 10, we use the same $\mathbf{A}$ (i.e., the original graph) to add links for each negative node $y$. This is because we assume that all the negative

[2]Note that step 3 is to insure that the $s^{\text{th}}$ column of $\tilde{\mathbf{A}}$ sums up to 1.

**Algorithm 1** Add Links for One Negative Node

**Input:** The adjacency matrix $\mathbf{A}$, the negative node $y$, the neighborhood size $k$ and $c$.
**Output:** The updated adjacency matrix $\tilde{\mathbf{A}}$.

1: initialize $\tilde{\mathbf{A}} = \mathbf{A}$, $\tilde{\mathbf{A}}(n+1,:) = 0$, and $\tilde{\mathbf{A}}(:,n+1) = 0$.
2: get the ranking vector for the negative node $y$ by $\mathbf{r}_y = c\mathbf{A}\mathbf{r}_y + (1-c)\mathbf{e}_y$. Let $\epsilon := k^{\text{th}}$ largest element in $\mathbf{r}_y$.
3: **for** each node $i$ **do**
4: **if** $\mathbf{r}_{y,l} \geq \epsilon$ **then**
5: set $\tilde{\mathbf{A}}(n+1, l) = \mathbf{r}_{y,i}/\mathbf{r}_{y,y}$
6: set $\tilde{\mathbf{A}}(1:n, i) = (1 - \mathbf{r}_{y,i}/\mathbf{r}_{y,y})\tilde{\mathbf{A}}(1:n, i)$
7: **end if**
8: **end for**
9: output $\tilde{\mathbf{A}} = \tilde{\mathbf{A}}(1:n, 1:n)$.

nodes are obtained in a batch mode (i.e., there is no ordering among different negative nodes). Then, we perform a random walk with restart on the refined graph ($\tilde{\mathbf{A}}$) for the source node $s$ (step 13) and output the corresponding steady state probability as the proximity score (step 14). For example, Fig. 2(b) plots the ranking vector (sorted from highest to lowest) for node 1 of the running example with side inforamtion ($\mathcal{P} = \{4\}$, and $\mathcal{N} = \{6\}$). Compared to the case without side information (Fig. 2(a)), it can be seen that positive node (node 4) as well as its neighborhood (nodes 2 and 3) receives higher proximity scores; while the negative node (node 6) as well as its neighboring nodes (nodes 5 and 7) receives lowers scores.

## 4 Fast-ProSIN

In this section, we introduce our fast solution for ProSIN. We start by reviewing NB_LIN, which is a fast algorithm to compute random walk with restart (the proximity without side information) [25]. We then extend it to include side information.

### 4.1 Background: NB_LIN for RWR

According to the definition (equation (1)), we need to invert an $n \times n$ matrix. This operation is prohibitively slow for

**Algorithm 2** ProSIN

**Input:** The adjacency matrix $\mathbf{A}$, the source node $s$ and the target node $t$, the side information $\mathcal{P}$ and $\mathcal{N}$, the neighborhood size $k$, and the parameter c.

**Output:** the proximity score $\tilde{\mathbf{r}}_{s,t}$ from source $s$ to target $t$.

1: initialize $\tilde{\mathbf{A}} = \mathbf{A}$
2: **if** $n^+ > 0$ **then**
3: $\quad \tilde{\mathbf{A}}(:,s) = n^s/(n^s + n^+)\tilde{\mathbf{A}}(:,s)$
4: $\quad$ **for** each positive node $x$ in $\mathcal{P}$ **do**
5: $\quad\quad \tilde{\mathbf{A}}(x,s) = \tilde{\mathbf{A}}(x,s) + 1/(n^s + n^+)$.
6: $\quad$ **end for**
7: **end if**
8: **if** $n^- > 0$ **then**
9: $\quad$ **for** each negative node $y$ in $\mathcal{N}$ **do**
10: $\quad\quad$ update $\tilde{\mathbf{A}}$ by Alg. 1
11: $\quad$ **end for**
12: **end if**
13: solve the equation $\tilde{\mathbf{r}}_s = c\tilde{\mathbf{A}}\tilde{\mathbf{r}}_s + (1-c)\mathbf{e}_s$.
14: output $\tilde{\mathbf{r}}_{s,t} = \tilde{\mathbf{r}}_s(t)$.

large graphs. On the other hand, the iterative method (iterating equation (1) until convergence) might need many iterations, which is also not efficient. In [25], the authors solve this problem using a low-rank approximation, followed by a matrix inversion of size $l \times l$ (where $l$ is the rank of the low-rank approximation) to get all possible proximity scores. Their solution, called NB_LIN, is the starting point for our fast algorithm.

Alg. 3 summarizes NB_LIN, where it is divided into two stages: NB_LIN_Pre() and NB_LIN_OQ(). In NB_LIN_Pre() (steps 1-3), a low-rank approximation is performed for the normalized adjacency matrix $\mathbf{A}$ and a matrix inversion $\mathbf{\Lambda}$ is computed. Next, in NB_LIN_OQ() (steps 4-5), only a small number of matrix-vector multiplications are computed to output the ranking vector.

**Algorithm 3** NB_LIN

**Input:** The normalized adjacency matrix $\mathbf{A}$, the source node $s$ and $c$.

**Output:** The ranking vector for source node $\mathbf{r}_s$.

1: **Pre-Compute Stage** (NB_LIN_Pre())
2: do low-rank approximation for $\mathbf{A} = \mathbf{USV}$
3: pre-compute and store the matrix $\mathbf{\Lambda} = (\mathbf{S}^{-1} - c\mathbf{VU})^{-1}$
4: **On-Line Query Stage** (NB_LIN_OQ())
5: output $\mathbf{r}_s = (1-c)(\mathbf{e}_s + c\mathbf{U\Lambda Ve}_s)$

## 4.2 Fast-ProSIN

To incorporate side information, we need to solve random walk with restart in two places. First, we process the original graph $\mathbf{A}$ (step 10 in Alg. 4); and then we process the refined graph $\tilde{\mathbf{A}}$ to get the ranking vector for the source node $s$ (step 13 in Alg. 4). If we utilize NB_LIN in a straightforward way, we have to call it twice (for $\mathbf{A}$ and for $\tilde{\mathbf{A}}$, respectively). Unfortunately, this does not fit the expect usage model of side information, where it needs to reflect users' real-time interests. Imagine a user is querying an author-conference bipartite graph, and s/he wants to know *which conferences are most similar to KDD*. After the system gives the initial search results, s/he might further give her/his own preference (e.g., *dislike ICML*) and expect updated search results that matches her/his interests. This basically implies that calling NB_LIN_Pre() on the refined graph $\tilde{\mathbf{A}}$ is part of the on-line cost, which may pose a huge threat to the system's performance.

**Algorithm 4** Fast-ProSIN

**Input:** The adjacency matrix $\mathbf{A}$, the source node $s$, the side information $\mathcal{P}$ and $\mathcal{N}$, the neighborhood size $k$, and the parameter $c$.

**Output:** the ranking vector $\tilde{\mathbf{r}}_s$ for the source $s$.

1: **Pre-Compute Stage**
2: call $[\mathbf{U}, \mathbf{\Lambda}, \mathbf{V}] = \text{NB\_LIN\_Pre}(\mathbf{A}, c)$
3: **On-Line Query (Feedback) Stage**
4: initialize $i_0 = 1$ and $\mathbf{\Theta} = \mathbf{0}^{(kn^- + 1)\times 2}$
5: **for** each negative node $y$ in $\mathcal{N}$ **do**
6: $\quad$ call $\mathbf{r}_y = \text{NB\_LIN\_OQ}(c, \mathbf{U}, \mathbf{\Lambda}, \mathbf{V}, \mathbf{e}_y)$.
7: $\quad$ let $\epsilon := k^{\text{th}}$ largest element in $\mathbf{r}_y$.
8: $\quad$ **for** each node $i$ s.t. $\mathbf{r}_{y,i} >= \epsilon$ **do**
9: $\quad\quad$ set $\mathbf{\Theta}(i_0, 1) = i$ and $\mathbf{\Theta}(i_0, 2) = 1 - \mathbf{r}_{y,i}/\mathbf{r}_{y,y}$
10: $\quad\quad$ increase $i_0$ by 1
11: $\quad$ **end for**
12: **end for**
13: set $\mathbf{\Theta}(i_0, 1) = s$ and $\mathbf{\Theta}(i_0, 2) = n^s/(n^s + n^+)$
14: set $\tilde{\mathbf{U}} = \mathbf{U}$ and $\tilde{\mathbf{V}} = \mathbf{V}$
15: **for** $i = 1 : kn^- + 1$ **do**
16: $\quad$ set $\mathbf{X}(i,:) = \mathbf{U}(\mathbf{\Theta}(i,1),:)$
17: $\quad$ set $\mathbf{Y}(:,i) = \mathbf{V}(:,\mathbf{\Theta}(i,1))(\mathbf{\Theta}(i,2) - 1)$
18: $\quad$ set $\mathbf{V}(:,\mathbf{\Theta}(i,1)) = \mathbf{V}(:,\mathbf{\Theta}(i,1))\mathbf{\Theta}(i,2)$
19: **end for**
20: compute $\mathbf{L} = (\mathbf{I} - c\mathbf{X\Lambda Y})^{-1}$
21: update $\tilde{\mathbf{\Lambda}} = \mathbf{\Lambda} + c\mathbf{\Lambda YLX\Lambda}$
22: set $\mathbf{e}_+ = \mathbf{0}^{n\times 1}$, $\mathbf{e}_+(\mathcal{P}) = 1/(n^s + n^+)$
23: call $\hat{\mathbf{r}}_s = \text{NB\_LIN\_OQ}(c, \tilde{\mathbf{U}}, \tilde{\mathbf{\Lambda}}, \tilde{\mathbf{V}}, \mathbf{e}_s)$
24: call $\mathbf{u} = \text{NB\_LIN\_OQ}(c, \tilde{\mathbf{U}}, \tilde{\mathbf{\Lambda}}, \tilde{\mathbf{V}}, \mathbf{e}_+)$
25: output $\tilde{\mathbf{r}}_s = \hat{\mathbf{r}}_s + c\hat{\mathbf{r}}_s(s)/(1 - c - c\mathbf{u}(s))\mathbf{u}$

To deal with such challenge, we propose Fast-ProSIN, which is given in Alg. 4. Here, we assume that we want the whole ranking vector for a given source node $s$ since a single proximity score can be read out from such ranking vector. Also, we consider the most general case, where both positive nodes and negative nodes are present. In Fast-ProSIN, it first calls NB_LIN_Pre() on the original adjacency matrix $\mathbf{A}$ (step 2). Then it calls NB_LIN_OQ() to

determine the influence of the negative nodes (steps 5-12) and partial influence (i.e., scaling the $s^{\text{th}}$ column of the adjacency matrix by a factor of $n^s/(n^s + n^+)$) of positive nodes (step 13), both of which are used to update the low-rank approximation ($\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$) as well as matrix $\tilde{\mathbf{\Lambda}}$ (steps 14 - 21). This way, it avoids directly calling the function NB_LIN_Pre() on the refined graph $\tilde{\mathbf{A}}$, where it would need to do a low-rank approximation and a matrix inversion, both of which are not efficient as on-line costs. Finally, it calls NB_LIN_OQ() twice (steps 23-24) and combines them as the final ranking result (step 25). Note that the second call on $\mathbf{e}_+$ (step 24) is used to compensate for the remaining influence of the positive nodes (i.e., adding new links from the source to the positive nodes).

The correctness of Alg. 4 is guaranteed by theorem 1. By theorem 1, Fast-ProSIN will not introduce additional approximation errors beyond the first time it calls NB_LIN_Pre() on the original graph. Therefore, Fast-ProSIN is expected to obtain ranking results similar to calling NB_LIN_Pre() twice (one for $\mathbf{A}$ and the other for $\tilde{\mathbf{A}}$). On the other hand, Fast-ProSIN avoids the expensive steps (low-rank approximation on $\tilde{\mathbf{A}}$ and a matrix inversion of size $l \times l$) in calling NB_LIN_Pre(). This, as we will show, leads to significant on-line running cost savings.

**Theorem 1 Correctness of Fast-ProSIN.** *If $\mathbf{A} = \mathbf{USV}$ holds, then Alg. 4 gives the correct ranking vector for the source node $s$.*

**Proof:** let an $n \times n$ matrix $\hat{\mathbf{A}}$ s.t.,

$$\begin{aligned} \hat{\mathbf{A}}(:, \mathbf{\Theta}(j,1)) &= \mathbf{A}(:, \mathbf{\Theta}(j,1))\mathbf{\Theta}(j,1) \ (j = 1 : kn^- + 1) \\ \hat{\mathbf{A}}(:,i) &= \mathbf{A}(:,i) \quad \text{if} \quad i \notin \mathbf{\Theta}(:,1) \end{aligned} \tag{2}$$

First, we will show that $\hat{\mathbf{r}}_s$ in step 23 gives the correct ranking vector on the matrix $\hat{\mathbf{A}}$ if $\mathbf{A} = \mathbf{USV}$ holds.

By the construction of matrix $\hat{\mathbf{A}}$, we have

$$\begin{aligned} \hat{\mathbf{A}}(:, \mathbf{\Theta}(j,1)) &= \mathbf{USV}(:, \mathbf{\Theta}(j,1))\mathbf{\Theta}(j,1) \ (j = 1 : kn^- + 1) \\ \hat{\mathbf{A}}(:,i) &= \mathbf{USV}(:,i) \quad \text{if} \quad i \notin \mathbf{\Theta}(:,1) \end{aligned} \tag{3}$$

Thus, in the matrix form, we have $\hat{\mathbf{A}} = \tilde{\mathbf{U}}\mathbf{S}\tilde{\mathbf{V}}$, where the matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are as defined in steps 14-19 in Alg. 4.

Define the matrix $\hat{\mathbf{Q}} = (1-c)(\mathbf{I} - c\hat{\mathbf{A}})^{-1}$. By the property of NB_LIN algorithm [25], we have

$$\begin{aligned} \hat{\mathbf{Q}} &= (1-c)(\mathbf{I} - c\hat{\mathbf{A}})^{-1} \\ &= (1-c)(\mathbf{I} - c\tilde{\mathbf{U}}\mathbf{S}\tilde{\mathbf{V}})^{-1} \\ &= (1-c)(\mathbf{I} + c\tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{V}}) \end{aligned} \tag{4}$$

where $\hat{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - c\tilde{\mathbf{V}}\tilde{\mathbf{U}})^{-1}$.

Next, we will relate $\hat{\mathbf{\Lambda}}$ with the matrix $\tilde{\mathbf{\Lambda}}$ (step 21 of Alg. 4).

By the spectral representation, we have the following equation:

$$\begin{aligned} \mathbf{S}^{-1} - c\tilde{\mathbf{V}}\tilde{\mathbf{U}} &= \mathbf{S}^{-1} - c\sum_i \tilde{\mathbf{V}}(:,i)\tilde{\mathbf{U}}(i,:) \\ &= \mathbf{S}^{-1} - c(\sum_i \mathbf{V}(:,i)\mathbf{U}(i,:) + \delta) \end{aligned} \tag{5}$$

where $\delta$ satisfies

$$\begin{aligned} \delta &= \sum_{j=1}^{kn^-+1} \mathbf{V}(:, \mathbf{\Theta}(j,1))\mathbf{U}(\mathbf{\Theta}(j,1),:)(\mathbf{\Theta}(j,2) - 1) \\ &= \mathbf{YX} \end{aligned} \tag{6}$$

where the matrices $\mathbf{Y}$ and $\mathbf{X}$ are defined as steps 16-17 of Alg. 4.

Plugging equations (5) and (6) into the matrix $\hat{\mathbf{\Lambda}}$ and applying Sherman-Morrison Lemma [19], we have

$$\begin{aligned} \hat{\mathbf{\Lambda}} &= (\mathbf{S}^{-1} - c\tilde{\mathbf{V}}\tilde{\mathbf{U}})^{-1} \\ &= \mathbf{\Lambda} + c\mathbf{\Lambda}\mathbf{Y}\mathbf{L}\mathbf{X}\mathbf{\Lambda} \\ &= \tilde{\mathbf{\Lambda}} \end{aligned} \tag{7}$$

where the matrices $\tilde{\mathbf{\Lambda}}$ and $\mathbf{L}$ are defined as steps 20-21 of Alg. 4.

Plugging equation (7) into equation (4), we can verify the $\hat{\mathbf{r}}_s$ in step 23 satisfies:

$$\hat{\mathbf{r}}_s = \hat{\mathbf{Q}}(:,s) \tag{8}$$

Next, define the matrix $\tilde{\mathbf{Q}} = (1-c)(\mathbf{I} - c\tilde{\mathbf{A}})^{-1})$. We will try to relate $\tilde{\mathbf{Q}}$ with matrix $\hat{\mathbf{Q}}$.

By the construction of $\tilde{\mathbf{A}}$ and $\hat{\mathbf{A}}$, we have

$$\tilde{\mathbf{A}} = \hat{\mathbf{A}} + \mathbf{e}_+\mathbf{e}_s' \tag{9}$$

where vector $\mathbf{e}_+$ is defined as in step 22. In other words, there is only a rank-1 difference between $\tilde{\mathbf{A}}$ and $\hat{\mathbf{A}}$.

Now, applying Sherman-Morrison Lemma [19] to $\tilde{\mathbf{Q}}$, we have

$$\begin{aligned} \tilde{\mathbf{Q}} &= (1-c)(\mathbf{I} - c\tilde{\mathbf{A}})^{-1}) \\ &= (1-c)(\mathbf{I} - c\hat{\mathbf{A}} - c\mathbf{e}_+\mathbf{e}_s')^{-1}) \\ &= \hat{\mathbf{Q}} + b\hat{\mathbf{Q}}\mathbf{e}_+\mathbf{e}_s'\hat{\mathbf{Q}} \\ &= \hat{\mathbf{Q}} + b\mathbf{u}\hat{\mathbf{Q}}(s,:) \end{aligned} \tag{10}$$

where vector $\mathbf{u}$ is defined as in step 24 and the scale $b$ satisfies

$$\begin{aligned} b &= \frac{c}{1 - c - c\mathbf{e}_s'\hat{\mathbf{Q}}\mathbf{e}_+} \\ &= \frac{c}{1 - c - c\mathbf{e}_s'\mathbf{u}} \\ &= \frac{c}{1 - c - c\mathbf{u}(s)} \end{aligned} \tag{11}$$

**Table 2. Summary of data sets**

| dataset | number of nodes | number of edges |
|---|---|---|
| **AC** | 421,807 | 1,066,816 |
| **ML** | 4,563 | 20,469 |
| **CoMMG** | 54,200 | 354,186 |

Putting equations (7), (10) and (11) together, we have that the correct ranking vector for the source node $s$ on matrix $\tilde{\mathbf{A}}$ must satisfies:

$$\begin{aligned}\tilde{\mathbf{Q}}(:,s) &= \hat{\mathbf{Q}}(:,s) + b\mathbf{u}\hat{\mathbf{Q}}(s,s) \\ &= \hat{\mathbf{r}}_s + \frac{c\hat{\mathbf{r}}_s(s)}{1-c-c\mathbf{u}(s)}\mathbf{u} \\ &= \tilde{\mathbf{r}}_s \end{aligned} \quad (12)$$

where $\tilde{\mathbf{r}}_s$ is defined as in step 25, which completes the proof of theorem 1. □

## 5 Experimental Evaluations

In this section we present experimental results. All the experiments are designed to answer the following questions:

- *Effectiveness:* What data mining observations does the proposed ProSIN enable?
- *Efficiency:* How does the proposed Fast-ProSIN balance between speed and quality?

### 5.1 Experimental Setup

**Datasets.** We use three datasets in our experiments, which are summarized in Table 2.

The first dataset (**AC**) is from DBLP.[3] It is an author-conference bipartite graph, where each row corresponds to an author and each column corresponds to a conference. An edge weight is the number of papers that the corresponding author publishes in the corresponding conference. On the whole, there are 421,807 nodes (418,236 authors and 3,571 conferences) and 1,066,816 edges in the graph.

The second dataset (**ML**) uses author-paper information from two major machine learning conferences ('NIPS', and 'ICML') to construct a co-authorship graph, where each node represents an author and an edge weight is the number of co-authored papers between any two corresponding authors. On the whole, there are 4,563 nodes and 20,469 edges.

The third dataset (**CoMMG**) is used in [18], which contains around 7,000 captioned images, each with about 4 captioned terms. There are in total 160 terms for captioning. In our experiments, 1,740 images are set aside for testing. The graph matrix is constructed exactly as in [18], which contains 54,200 nodes and 354,186 edges.

[3] http://www.informatik.uni-trier.de/~ley/db/

**Parameter Settings.** There are two parameters in the proposed ProSIN: $c$ for random walk with restart, and $k$ for the neighborhood size of a given negative node. We set $c = 0.95$ (as suggested in [25]). To determine $k$, a parametric study has been performed[4] and ProSin shows little sensitivity to $k$ for a large range of settings (from $k = 2$ to $k = 10$). For the experimental results in this paper, $k$ is set to be 5.

**Machine Configurations.** For the computational cost, we report the wall-clock time. All the experiments ran on the same machine with four 3.0GHz Intel (R) Xeon (R) CPUs and 16GB memory, running Linux (2.6 kernel). For each experiment, we run it 10 times and report the average.

### 5.2 Effectiveness: Case Studies

In both the proposed ProSIN and the original random walk with restart, the proximity score is defined as the steady-state probability . Therefore, we expect it to enrich a broad range of applications by replacing the original random walk with restart with our ProSIN. In this subsection, we present three applications as case studies: neighborhood search, center-piece subgraphs, and image caption.

**Neighborhood Search.** By incorporating the users' feedback, we can allow interactive neighborhood search on the graph. Fig. 4 gives one such example, where we want to find the top 10 neighbors of 'KDD' conferences (i.e, the 10 most similar conferences as 'KDD') from the **AC** dataset. In Fig. 4(a), we plot the initial results when there is no side information (i.e, $\mathcal{P} = \phi$ and $\mathcal{N} = \phi$). Subjectively, the result makes sense, which reflects two major sub-communities in 'KDD': the AI/statistic community (e.g., 'ICML', 'NIPS', and 'IJCAI') and the databases community (e.g., 'SIGMOD', 'VLDB', 'ICDE' etc). Then, if the user gives negative feedback on 'ICML' (i.e, $\mathcal{P} = \phi$ and $\mathcal{N} = \{'\text{ICML}'\}$), all the AI/statistic related conferences ('NIPS' and 'IJCAI') disappear (See Fig. 4(b)). In Fig. 4(c), we present the updated result if the user further gives some positive feedback on 'SIGIR', which is one of the major conferences on information retrieval. Again, the result confirms the effectiveness of ProSIN: positive feedback on 'SIGIR' brings more information retrieval related conferences (e.g, 'TREC', 'CIKM', 'ECIR','CLEF', 'ACL', 'JCDL', etc).

**Center-Piece Subgraphs.** The concept of connection subgraphs, or center-piece subgraphs, was proposed in [7, 22]: Given $Q$ query nodes, it creates a subgraph $\mathcal{H}$ that shows the relationships between the query nodes. The resulting subgraph should contain the nodes that have strong connection to all or most of the query nodes. Moreover, since this subgraph $\mathcal{H}$ is used for visually demonstrating node relations, its visual complexity is capped by setting an upper limit, or a *budget* on its size. These so-called con-

[4] We skip the details of the parametric study for brevity.

| | | |
|---|---|---|
| 'ICDM'<br>'ICML'<br>'SDM'<br>'VLDB'<br>'ICDE'<br>'SIGMOD'<br>'NIPS'<br>'PKDD'<br>'IJCAI'<br>'PAKDD' | 'ICDM'<br>'SDM'<br>'PKDD'<br>'ICDE'<br>'VLDB'<br>'SIGMOD'<br>'PAKDD'<br>'CIKM'<br>'SIGIR'<br>'WWW' | 'SIGIR'<br>'TREC'<br>'CIKM'<br>'ECIR'<br>'CLEF'<br>'ICDM'<br>'JCDL'<br>'VLDB'<br>'ACL'<br>'ICDE' |
| (a) No feedback | (b) $\mathcal{N} = \{$'ICML'$\}$ | (c) $\mathcal{N} = \{$'ICML'$\}$<br>$\mathcal{P} = \{$'SIGIR'$\}$ |

**Figure 4. Interactive neighborhood search for 'KDD' conference.**

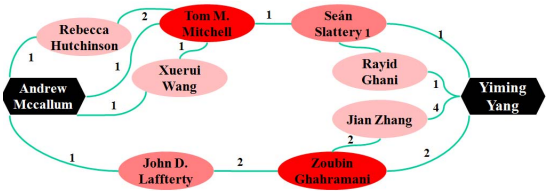
nection subgraphs (or center-piece subgraphs) were proved useful in various applications, but currently cannot handle users' interaction (i.e, feedback).

One of the building block in the original center-piece subgraphs [22] is to use RWR to measure the proximity from the query nodes to the remaining nodes on the graph. Therefore, by replacing the original random walk with restart by the proposed ProSIN, we can naturally deal with the users' interactions (for details of center-piece subgraph, please refer to [22]).

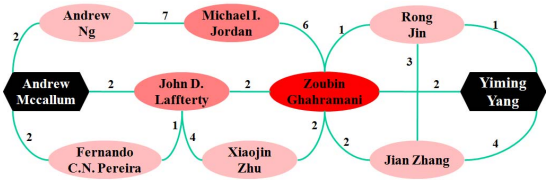
Fig. 5 plots an example to find the center-piece subgraphs between two researchers ('Andrew Mccallum' and 'Yiming Yang') from **ML** dataset. In Fig. 5(a), we plot the initial results when there is no side information (i.e, $\mathcal{P} = \phi$ and $\mathcal{N} = \phi$). It can be seen that there are two major connections between 'Andrew Mccallum' and 'Yiming Yang': one connection is on text mining/information retrieval (through 'Rebecca Hutchinson', 'Xuerui Wang', 'Tom M. Mitchell', 'Sean Slattery' and 'Rayid Ghani'), and the other connection in on AI/statistics (throught 'John D. Laffferty', 'Zoubin Ghahramani' and 'Jian Zhang'). Fig. 5(b) gives the updated result if the user gives negative feedback on 'Tom M. Mitchell'. It can be seen that the whole connection on text mining/information retrieval disappears, and more connection on AI/statistics (e.g. through 'Andrew Ng' and 'Michael I. Jordan') shows up.

**Image Caption.** Here, the goal is to assign some keywords for a given image as its text annotation. In [18], the authors proposed a graph based solution and showed its superiority over the traditional methods in feature space. The key idea of [18] is to construct an image-keyword-region graph and use RWR to measure the relevance between the test image and the known keywords. Similar to center-piece subgraphs, replacing RWR by ProSINcan easily incorporate side information (if available) in such process.

Fig. 6 presents the average precison/recall on **CoMMG** dataset. Here, the side-information is simulated as following: for each test image, 5 keywords that are most relevant to the test image based on the current proximity measurement are returned for users' yes/no (i.e., correct/wrong




(a) No feedback




(b) Negative feedback on 'Tom M. Mitchell'

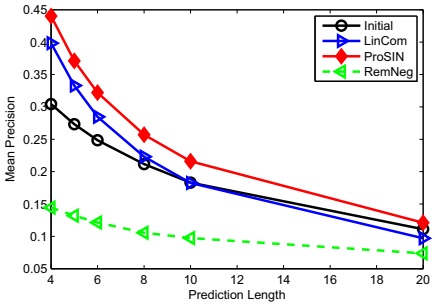
**Figure 5. Interactive center-piece subgraphs between 'Andrew Mccallum' and 'Yiming Yang'.**

caption) confirmation. Here, we also compare two simple strategies: (1) 'RemNeg', where the negative nodes are simply removed from the graph; and (2) 'LinCom' [13], where the proximity scores from positive/negative nodes are added/substracted from the score from the test image. From the figure, it can be seen that our ProSIN largely improves both precision/recall for image caption task by incorporating such side information. For example, it improves the precision by 13.59% (44.02% vs. 30.43%) and the recall by 17.39% (57.54% vs. 40.15%) when the prediction length is 4. It is interesting to notice that if we simply remove the negative nodes from the graph, it will actually hurts the performance ('RemNeg'). As for 'LinCom', it can be seen that (1) the improvement is limited compared with the proposed ProSIN for short prediction length; and (2) it might hurt the performance with the increase of the prediction length.

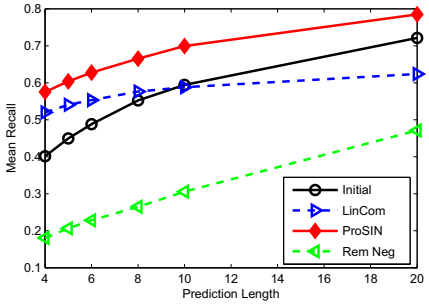
### 5.3 Efficiency

In this subsection, we study the quality/speed trade-off of the proposed Fast-ProSIN. We use the **CoMMG** dataset (since it is the only one with ground truth among the three datasets we used in this paper). Here, we fix the prediction length to be 4 (the results with other prediction length are similar and therefore skipped for brevity), and we compare the precision/recall between Fast-ProSIN and ProSIN where in ProSIN random walk with restart is performed by the iterative method.[5] Compared with ProSIN, there is one more parameter in Fast-ProSIN, the rank of the low-rank approximation for $\mathrm{NB\_LIN\_Pre}()$. We vary this parameter from 100 to 600 (denoted as Fast-ProSIN(100), Fast-

[5] An alternative choice for ProSIN is to run $\mathrm{NB\_LIN\_Pre}()$ on $\mathbf{A}$ and $\tilde{\mathbf{A}}$ respectively. However, we find it needs more wall-clock time but leads to lower quality compared with the iterative method. Therefore, we only compare the proposed Fast-ProSIN with that by iterative method.




(a) Mean precision




(b) Mean recall

**Figure 6. Incorporate side information for image caption.**

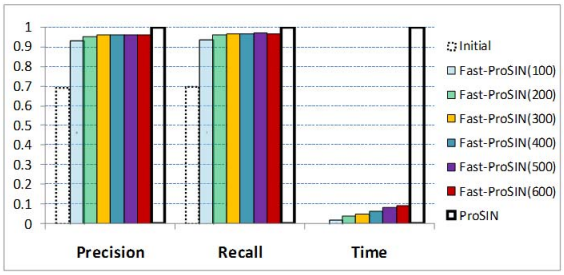



**Figure 7. Quality/speed trade-off of Fast-ProSIN.**

ProSIN(200), etc in Fig. 7). In order to put quality/speed in the same figure, we normalized (1) precision/recall by the largest value for ProSIN, and (2) time by the longest value for ProSIN.

From Fig. 7, it can be seen that the proposed Fast-ProSIN achieves significant speedup while maintaining high quality. For example, Fast-ProSIN(100) is 49x faster than ProSIN (the most right one) while it preserves 93.6% precision (41.2% vs. 44.0%) and 94.0% recall (54.1% vs. 57.5%); Fast-ProSIN(400) is 16x faster than ProSIN while preserving 96.1% precision (42.4% vs. 44.0%) and 96.7% recall (55.6% vs. 57.5%). Overall, Fast-ProSIN is 10∼49x faster than ProSIN, while preserving more than 93.0% quality (for both precision and recall). Note that in all cases, Fast-ProSIN significantly improves the precision/recall when compared with the initial case (the left-most dashed bar). As for the wall-clock time, ProSIN need 3.7 hours to annotate all the 1,740 images, while Fast-ProSIN(100) only needs 4.5 minutes.

## 6 Related Work

In this section, we review the related work, which can be categorized into two parts: node proximity and matrix low rank approximation.

**Node Proximity.** One of the most popular proximity measurements is random walk with restart [13, 18, 25], which is the baseline of ProSIN. Other representative proximity measurements include the sink-augmented delivered current [7], cycle free effective conductance [16], survivable network [10], and direction-aware proximity [24]. All these methods only consider the graph link structure and ignore the side information. Although we focus on random walk with restart in this paper, our approach (i.e., to use the side information to refine the graph structure) can be applied to other random walk-based measurements, such as [7, 24]. In term of dealing with the side information on ranking, our work is also related to [3], where the goal is to use partial order information to learn the weights of different types of edges. In term of computation, the fast algorithm (NB_LIN) for random walk with restart in [25] is most related to the proposed Fast-ProSIN. Our Fast-ProSIN differs from that in [25] in the sense that the graph structure in our setting keeps changing by the side information, whereas it is fixed in [25]. The core idea behind the proposed Fast-ProSIN is to leverage the smoothness between graph structure with/without side information. In [26], the authors has used the similar idea to track the proximity/centrality on a time-evolving skewed bipartite graph. Other remotely related work includes [11], where the goal is to propagate the trust/distrust to predict the trust between any two persons.

Graph proximity is an important building block in many graph mining settings. Representative work includes connection subgraphs [7, 16, 22], neighborhood search in bipartite graphs [20], content-based image retrieval [13], cross-modal correlation discovery [18], the BANKS system [2], link prediction [17], pattern matching [23], ObjectRank [4], RelationalRank [8] and recommendation system [5]. Note that for the ranking-related tasks (such as neighborhood search, image retrieval, etc.), we can also use a linear combination strategy suggested in [13], which itself includes personalized PageRank [12] as a special case when negative set is absent, to incorporate like/dislike type of side information. Our experimental evaluation on image caption task shows that although it is effective for small prediction lengths, its performance is not as good as the proposed ProSIN and sometimes it actually hurts the performance. What is more important, it is not clear how to use such strategy (linear combination) for more complicated applications (such as center-piece subgraphs, pattern match etc). This is exactly one major advantage of the proposed ProSIN: it can

be easily plugged into such applications by simply replacing the original proximity measurement by our ProSIN.

**Low Rank Approximation.** Low rank approximation [9, 6, 1] plays a very important role in graph mining. For example, the low rank approximation structure is often a good indicator to identify the community in the graph. A significant deviation from such structure often implies anomalies in the graph. For the proposed Fast-ProSIN, we need the low rank approximation in the pre-computational stage (in function NB_LIN_OQ()). The most popular choices include SVD/PCA [9, 15] and random projection [14]. More recent methods includes CUR [6] and its improved version CMD [21] to deal with the sparseness of many real graphs. Notice that our Fast-ProSIN is orthogonal to the specific method of low rank approximation.

## 7 Conclusion

In this paper, we study how to incorporate like/dislike type of side information in measuring node proximity on large graphs. Our main contributions are in two folds. First, we proposed a novel method (ProSIN) to incorporate side information in measuring node proximity on large graphs and showed its broad applicability through various case studies. Second, to enhance the efficiency of ProSIN, we also took advantage of the smoothness of the graph structures with/without side information and proposed a fast algorithm (Fast-ProSIN). We demonstrated that Fast-ProSIN achieves significant speedup (up to 49x) in our evaluation on real datasets. Overall, we expect the proposed algorithms to enrich a broad range of applications that receive online feedback/side information.

## References


[1] D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2), 2007.

[2] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and S. S. Parag. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, 2002.

[3] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. In *KDD*, pages 14–23, 2006.

[4] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[5] H. Cheng, P.-N. Tan, J. Sticklen, and W. F. Punch. Recommendation via query centered random walk on k-partite graph. In *ICDM*, pages 457–462, 2007.

[6] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal of Computing*, 2005.

[7] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.

[8] F. Geerts, H. Mannila, and E. Terzi. Relational link-based ranking. In *VLDB*, pages 552–563, 2004.

[9] G. H. Golub and C. F. Van-Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[10] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In *Handbooks in Operations Research and Management Science 7: Network Models*. North Holland, 1993.

[11] R. V. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *WWW*, pages 403–412, 2004.

[12] T. H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.

[13] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Multimedia*, pages 9–16, 2004.

[14] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.

[15] K. V. R. Kanth, D. Agrawal, and A. K. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD Conference*, pages 166–176, 1998.

[16] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, pages 245–255, 2006.

[17] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proc. CIKM*, 2003.

[18] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[19] W. Piegorsch and G. E. Casella. Inverting a sum of matrices. In *SIAM Review*, volume 32, pages 470–470, 1990.

[20] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.

[21] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *SDM*, 2007.

[22] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[23] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.

[24] H. Tong, C. Faloutsos, and Y. Koren. Fast direction-aware proximity for graph mining. In *KDD*, pages 747–756, 2007.

[25] H. Tong, C. Faloutsos, and J.-Y. Pan. Random walk with restart: Fast solutions and applications. *Knowledge and Information Systems: An International Journal (KAIS)*, 2008.

[26] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, pages 704–715, 2008.