iPoG: Fast Interactive Proximity Querying on Graphs

Hanghang Tong
Carnegie Mellon University
Pittsburgh, PA, USA
htong@cs.cmu.edu

Huiming Qu
IBM T.J. Watson
Hawthorne, NY, USA
hqu@us.ibm.com

Hani Jamjoom
IBM T.J. Watson
Hawthorne, NY, USA
jamjoom@us.ibm.com

Christos Faloutsos
Carnegie Mellon University
Pittsburgh, PA, USA
christos@cs.cmu.edu

ABSTRACT

Given an author-conference graph, how do we answer proximity queries (e.g., *what are the most related conferences for John Smith?*); how can we tailor the search result if the user provides additional yes/no type of feedback (e.g., *what are the most related conferences for John Smith given that he does not like ICML?*)? Given the potential computational complexity, we mainly devote ourselves to addressing the computational issues in this paper by proposing an efficient solution (referred to as iPoG-B) for bipartite graphs. Our experimental results show that the proposed fast solution (iPoG-B) achieves significant speedup, while leading to the *same* ranking result.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – Data Mining

General Terms

Algorithm,experimentation

Keywords

proximity,scalability, graph mining

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 much previous work, most existing proximity measurements only consider the link structure of the underlying graph, ignoring any possible feedback 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.

Users' feedback exist in a wide range of scenarios, both implicitly or explicitly. For instance, in recommendation systems, feedback 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 feedback 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 [16], the authors proposed an effective solution for the above problem. The basic idea of their method is to use the feedback information to bias the graph structure, based on which the proximity is measured. It has a wide range of applicability as shown in [16]. The authors in [16] also proposed a fast solution for unipartite graphs and empirically demonstrated that it achieves a good balance between the on-line cost and off-line cost. However, there are several open questions that were not answered in [16]. For example, why should we expect the proposed technique in [16] to perform better than some simple heuristics (e.g., linear combination)? How can we develop fast solutions for more general graphs (such as bipartite graphs)? etc.

Given the potential computational complexity, we mainly devote ourselves to addressing the computational issues in this paper by proposing an efficient solution (referred to as iPoG-B) for bipartite graphs. Our experimental results show that the proposed fast solution (iPoG-B) achieves significant speedup, while leading to the *same* ranking result.

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

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 bold letters for matrices (e.g., $\mathbf{A}, \mathbf{B}$...), lower case bold letters for vectors (e.g. $\mathbf{a}$), and calligraphic fonts for sets (e.g. $\mathcal{I}$). We use the symbol "$\tilde{}$" to distinguish

Table 1: Symbols

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}, \ldots$	matrices (bold upper case)
$\mathbf{A}(i, j)$	element at the i^{th} row and j^{th} column of $\mathbf{A}$
$\mathbf{A}(i, :)$	i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	j^{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 settings with/without feedback. For example, $\mathbf{A}$ is the normalized adjacency matrix of the graph without feedback; and $\tilde{\mathbf{A}}$ is the normalized adjacency matrix of the refined graph by feedback.

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^{th} row and j^{th} column of the matrix $\mathbf{A}$, and $\mathbf{A}(:, j)$ is the j^{th} column of $\mathbf{A}$, etc.

We use a running example, depicted in figure1, 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 an edge represents some social contact between the two corresponding persons (e.g., a phone call). In traditional settings of proximity measurements, the goal is to quantify the closeness (i.e., relevance) between two nodes (the source and the target) based on the link structure of the underlying graph. In our settings, we assume the existence of like/dislike type of user feedback. In our running example, a user might not want to see (i.e., dislike) node 6 but favor (i.e., like) node 4.

Formally, we represent such feedback 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 to as positive nodes. The set $\mathcal{N}$ contains the node indices that users dislike (referred to as the 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 feedback information to measure the node proximity (e.g., the proximity from node 1 to node 3 in our running example).

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

PROBLEM 1. **(Proximity Search with Feedback)**

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

Find: *the proximity score* $\tilde{r}_{s,t}$ *from the source node* s *to the 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 feedback information does not need to exist simultaneously. For example, if we only have positive feedback, we can simply set the negative set to be empty (i.e., $\mathcal{N} = \phi$).

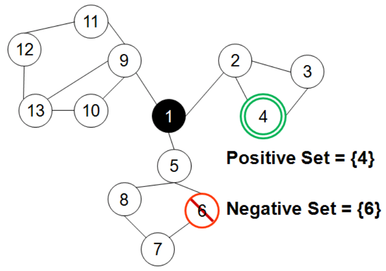

Figure 1: The running example. (node 1 is the source)

3. iPoG-B FOR BIPARTITE GRAPHS

3.1 Background #1: iPoG

In [16], the authors proposed an effective algorithm to incorporate users' feedback. The method is based on well-studied random walk with restart (RWR). Its basic idea is to leverage the feedback 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.

The algorithm (referred to as iPoG), which is the starting point of this paper, is summarized as Alg. 1.

Algorithm 1 iPoG

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

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

```
1:  initialize $\tilde{\mathbf{A}} = \mathbf{A}$
2:  if $n^+ > 0$ then
3:      $\tilde{\mathbf{A}}(:, s) = n^s/(n^s + n^+)\tilde{\mathbf{A}}(:, s)$
4:      for each positive node $x$ in $\mathcal{P}$ do
5:          $\tilde{\mathbf{A}}(x, s) = \tilde{\mathbf{A}}(x, s) + 1/(n^s + n^+)$.
6:      end for
7:  end if
8:  if $n^- > 0$ then
9:      for each negative node $y$ in $\mathcal{N}$ do
10:         decrease the weights of the out-links of node $y$ as well as
            that of $y$'s neighboring nodes.
11:     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 $\tilde{\mathbf{r}}_{s,t} = \tilde{\mathbf{r}}_s(t)$.
```

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

3.2 Background #2: BB_LIN for Bipartite Graphs without Feedback

Algorithm 2 BB_LIN (repeated from [25] for completeness)

```
Input: The adjacency matrix $\mathbf{B}$, and the query nodes $i$ and $j$.
Output: The ranking vector $\mathbf{r}_s$ for node $s$.
 1: Pre-Computate Stage(BB_Lin_Pre()):
 2: normalize for type 1 objects: $\mathbf{Br} = \mathbf{D}_1^{-1} \cdot \mathbf{B}$
 3: normalize for type 2 objects: $\mathbf{Bc} = \mathbf{D}_2^{-1} \cdot \mathbf{B}'$
 4: compute the core matrix: $\mathbf{\Lambda} = (\mathbf{I} - c^2\mathbf{Bc} \cdot \mathbf{Br})^{-1}$
 5: store the matrices: $\mathbf{Br}$, $\mathbf{Bc}$, and $\mathbf{\Lambda}$.
 6: Query Stage (BB_Lin_OQ()):
 7: let $\mathbf{e}_s = [\mathbf{y}_1; \mathbf{y}_2]$, where $\mathbf{y}_1$ and $\mathbf{y}_2$ are $n \times 1$ and $l \times 1$ vectors,
    respectively. If $s \leq n$, $\mathbf{y}_1(s) = 1$, $\mathbf{y}_1(i) = 0$ $(i \neq s)$ and
    $\mathbf{y}_2(i) = 0$, otherwise $\mathbf{y}_1(i) = 0$, $\mathbf{y}_2(s-n) = 1$, and $\mathbf{y}_2(i) =$
    $0$ $(i \neq s - n)$.
 8: compute $\mathbf{r}_1 = \mathbf{y}_1 + c^2 \cdot \mathbf{Br} \cdot \mathbf{\Lambda} \cdot \mathbf{Bc} \cdot \mathbf{y}_1 + c\mathbf{Br} \cdot \mathbf{\Lambda} \cdot \mathbf{y}_2$;
 9: compute $\mathbf{r}_2 = c \cdot \mathbf{\Lambda} \cdot \mathbf{Bc} \cdot \mathbf{y}_1 + \mathbf{\Lambda} \cdot \mathbf{y}_2$;
10: output $\mathbf{r}_s = (1-c)[\mathbf{r}_1; \mathbf{r}_2]$
```

Let $\mathbf{B}$ be an $n \times l$ adjacency matrix for the bipartite graph. One important observation from many real applications is that many real bipartite graphs are often highly skewed. For example, on the entire **NetFlix** data set, we have about $2.7M$ users, but only about $18K$ movies. In [14], the authors show that for such skewed, bipartite graphs, we only need to pre-compute and store a matrix inversion of size $l \times l$ to get all possible proximity scores. BB_LIN, which is the starting point for our fast algorithms, is summarized in Alg. 2. In Alg. 2, $\mathbf{D}_1 = \text{diag}(d_1(1), ...d_1(n))$, $d_1(i) = \sum_{j=1}^{l} \mathbf{B}(i,j)$, and $\mathbf{D}_2 = \text{diag}(d_2(1), ...d_2(l))$, $d_2(j) = \sum_{i=1}^{n} \mathbf{B}(i,j)$.

Based on Alg. 2, we only need to pre-compute and store a matrix inversion $\mathbf{\Lambda}$ of size $l \times l$. For skewed bipartite graphs ($l \ll n$), $\mathbf{\Lambda}$ is much cheaper to pre-compute and store. For example, on the entire **NetFlix** user-movie bipartite graph, which contains about $2.7M$ users, about $18K$ movies and more than $100M$ edges (see Section 6 for the detailed description of the data set), it takes 1.5 hours to pre-compute the $18K \times 18K$ matrix inversion $\mathbf{\Lambda}$. For the pre-compute stage, this is quite acceptable.

On the other hand, in the on-line query stage, we can get any proximity scores quickly. For example, to output a proximity score, we need at most two sparse matrix-vector multiplications. As an example, on the **NetFlix** data set, it takes less than 1 second to get one proximity score. Note that all possible proximity scores are determined by the matrix $\mathbf{\Lambda}$ (together with the normalized adjacency matrices $\mathbf{Br}$ and $\mathbf{Bc}$). We refer to the matrix $\mathbf{\Lambda}$ as the the *core matrix.*

3.3 iPoG-B for Bipartite Graphs with Feedback

As for the unipartite graphs, we cannot directly call BB_Lin_Pre() on the refined graph $\tilde{\mathbf{A}}$ in the case where user's feedback is obtained on-line. To deal with this issue, we propose iPoG-B, which is summarized in Alg. 3. In iPoG-B, it first calls BB_LIN_Pre() on the original adjacency matrix $\mathbf{B}$ (step 2). Then it calls BB_LIN_OQ() to determine the influence of the negative nodes (steps 4-14) and partial influence (i.e., scaling the s^{th} column of the adjacency matrix by a factor of $n^s/(n^s + n^+)$) of the positive nodes (steps 15-19), both of which are used to update the core matrix $\tilde{\mathbf{\Lambda}}$ (steps 20 - 28). This way, it avoids directly calling the function BB_LIN_Pre() on the refined graph $\tilde{\mathbf{A}}$, where it would need to do a multiplication between two big matrices and a matrix inversion of size $l \times l$, both of which are not efficient as on-line costs. Finally, it calls BB_LIN_OQ() twice (steps 29-31) and combines them as the final ranking result (step 32). Note that the second call on $\mathbf{e}_+$ (step 31) is used to compensate for the remaining influence of the positive nodes (i.e., adding new links from the source to the positive nodes).

Algorithm 3 iPoG-B

```
Input: The adjacency matrix $\mathbf{B}$, the source node $s$, the feedback
    information $\mathcal{P}$ and $\mathcal{N}$, the neighborhood size $k$, and the param-
    eter $c$.
Output: the ranking vector $\tilde{\mathbf{r}}_s$ for the source node $s$.
 1: Pre-Compute Stage
 2: call $[\mathbf{Br}, \mathbf{\Lambda}, \mathbf{Bc}] = $ BB_LIN_Pre$(\mathbf{B}, c)$
 3: On-Line Query (Feedback) Stage
 4: initialize $i_0 = 0$, $i_2 = 0$ and $\mathbf{\Theta}_1 = \Phi$, $\mathbf{\Theta}_2 = \Phi$;
 5: for each negative node $y$ in $\mathcal{N}$ do
 6:   call $\mathbf{r}_y = $ BB_Lin_OQ$(\mathbf{\Lambda}, \mathbf{Br}, \mathbf{Bc}, \mathbf{e}_y, c)$; let $\epsilon :=$ the $k^{\text{th}}$
      largest element in $\mathbf{r}_y$.
 7:   for each node $i$ s.t. $\mathbf{r}_{y,i} >= \epsilon$ do
 8:     if $i \leq n$ then
 9:       $i_1 + +$; set $\mathbf{\Theta}_1(i_1, 1) = i$; $\mathbf{\Theta}_1(i_1, 2) = 1 - \frac{\mathbf{r}_{y,i}}{\mathbf{r}_{y,y}}$;
10:     else
11:       $i_2 + +$; set $\mathbf{\Theta}_2(i_2, 1) = i - n$; $\mathbf{\Theta}_2(i_2, 2) = 1 - \frac{\mathbf{r}_{y,i}}{\mathbf{r}_{y,y}}$;
12:     end if
13:   end for
14: end for
15: if $i \leq n$ then
16:   $i_1 + +$; set $\mathbf{\Theta}_1(i_1, 1) = s$ and $\mathbf{\Theta}_1(i_1, 2) = n^s/(n^s + n^+)$
17: else
18:   $i_2 + +$; set $\mathbf{\Theta}_2(i_2, 1) = s$ and $\mathbf{\Theta}_2(i_2, 2) = n^s/(n^s + n^+)$
19: end if
20: set $\tilde{\mathbf{Br}} = \mathbf{Br}$ and $\tilde{\mathbf{Bc}} = \mathbf{Bc}$
21: set $\tilde{\mathbf{Br}}(:, \mathbf{\Theta}_2(:,1)) = \tilde{\mathbf{Br}}(:, \mathbf{\Theta}_2(:,1)) \cdot \text{diag}(\mathbf{\Theta}_2(:,2))$
22: set $\tilde{\mathbf{Bc}}(:, \mathbf{\Theta}_1(:,1)) = \tilde{\mathbf{Bc}}(:, \mathbf{\Theta}_1(:,1)) \cdot \text{diag}(\mathbf{\Theta}_1(:,2))$
23: set $\mathbf{Y}_1 = \tilde{\mathbf{Br}}(\mathbf{\Theta}_1(:,1), :)$
24: compute $\mathbf{X}_1 = -c^2 \cdot \mathbf{Bc}(:, \mathbf{\Theta}_1(:,1)) \cdot \text{diag}(1 - \mathbf{\Theta}_1(:,2))$
25: compute $\mathbf{X}_2 = -c^2 \cdot \mathbf{Bc} \cdot \mathbf{Br}(:, \mathbf{\Theta}_2(:,1)) \cdot \text{diag}(1 - \mathbf{\Theta}_2(:,2))$
26: set $\mathbf{Y}_2 = \mathbf{0}^{i_2 \times l}$; set $\mathbf{Y}_2(i, \mathbf{\Theta}_2(i,1) = 1(i = 1, ..., i_2)$
27: set $\mathbf{X} = [\mathbf{X}_1 \mathbf{X}_2]$, and $\mathbf{Y} = [\mathbf{Y}_1; \mathbf{Y}_2]$
28: compute $\mathbf{L} = (\mathbf{I} - \mathbf{X\Lambda Y})^{-1}$; update $\tilde{\mathbf{\Lambda}} = \mathbf{\Lambda} + \mathbf{\Lambda YLX\Lambda}$
29: set $\mathbf{e}_+ = \mathbf{0}^{n \times 1}$, $\mathbf{e}_+(\mathcal{P}) = 1/(n^s + n^+)$
30: call $\hat{\mathbf{r}}_s = $ BB_Lin_OQ$(\tilde{\mathbf{\Lambda}}, \tilde{\mathbf{Br}}, \tilde{\mathbf{Bc}}, \mathbf{e}_s, c)$
31: call $\mathbf{u} = $ BB_Lin_OQ$(\tilde{\mathbf{\Lambda}}, \tilde{\mathbf{Br}}, \tilde{\mathbf{Bc}}, \mathbf{e}_+, c)$
32: output $\tilde{\mathbf{r}}_s = \hat{\mathbf{r}}_s + c\hat{\mathbf{r}}_s(s)/(1 - c - c\mathbf{u}(s))\mathbf{u}$
```

4. EXPERIMENTAL EVALUATIONS

We use four data sets in our experiments, which are summarized in Table 2.

Figure 2 presents the results, where we compare iPoG-B with the iterative method In all the cases, iPoG-B is much faster than the iterative method. For example, iPoG-B is 90x faster (49 seconds vs. 4,423 seconds) on the **NetFlix** data set. Overall, the proposed iPoG-B achieves 4∼202x speedup over the iterative method. Note that there is *no quality loss* in iPoG-B.

5. RELATED WORK

One of the most popular proximity measurements is random walk with restart [8, 10, 14], which is the baseline of iPoG. Other representative proximity measurements include the sink-augmented delivered current [5], cycle free effective conductance [9], survivable

Table 2: Summary of data sets. 'B' for bipartite graphs, and 'U' for unipartite graphs.

Data Set	Number of Nodes	Number of Edges
AC	421,807	1,066,816
MovieLens	2,399	55,375
20NG	79,962	2,435,219
NetFlix	2,667,199	56,919,190

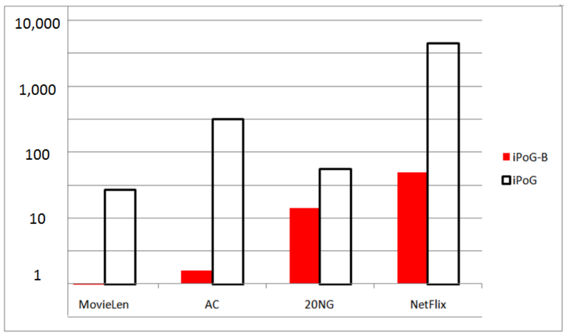

Figure 2: Comparison of speed. The proposed iPoG-B achieves 4∼202x speedup. See Section 6 for details.

network [7], and direction-aware proximity [13]. All these methods only consider the graph link structure and ignore the user feedback. Although we focus on random walk with restart in this paper, our approach (i.e., to use the feedback information to refine the graph structure) can be applied to other random walk-based measurements, such as [5, 13]. In terms of dealing with the feedback on ranking, our work is also related to [2], where the goal is to use partial order information to learn the weights of different types of edges. In terms of computation, the fast algorithms (NB_LIN and BB_LIN) for random walk with restart in [14] are the most related to the proposed fast solutions. Our fast solutions differ from that in [14] in the sense that the graph structure in our setting keeps getting changed by the feedback, whereas it is fixed in [14]. The core idea behind the proposed fast solutions is to leverage the smoothness between the graph structures with/without feedback. In [15], the authors have used the similar idea to track the proximity/centrality on a time-evolving skewed bipartite graph.

Graph proximity is an important building block in many graph mining settings. Representative work includes connection subgraphs [5, 9, 11], content-based image retrieval [8], cross-modal correlation discovery [10], the BANKS system [1], pattern matching [12], ObjectRank [3], RelationalRank [6] and recommendation system [4].

6. CONCLUSION

In this paper, we study how to incorporate user's feedback in proximity searching on large graphs. Our approach is based on a recent promising method [16]. Our main contribution is to propose an efficient solution for bipartite graphs (iPoG-B), which achieves up to *two orders of magnitude* speedup, while giving the same ranking results. A promising research direction is to parallelize the current method (e.g., using Hadoop).

7. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grants No. DBI-0640543 IIS-0705359 CNS-0721736 IIS0808661 iCAST and under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-CONF-404625), subcontracts B579447, B580840. This work is also partially supported by an IBM Faculty Award, a Yahoo Research Alliance Gift, a SPRINT gift, with additional funding from Intel, NTT and Hewlett-Packard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

8. REFERENCES

[1] 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.
[2] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. In *KDD*, pages 14–23, 2006.
[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.
[4] 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.
[5] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.
[6] F. Geerts, H. Mannila, and E. Terzi. Relational link-based ranking. In *VLDB*, pages 552–563, 2004.
[7] 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.
[8] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Multimedia*, pages 9–16, 2004.
[9] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, pages 245–255, 2006.
[10] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.
[11] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.
[12] 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.
[13] H. Tong, C. Faloutsos, and Y. Koren. Fast direction-aware proximity for graph mining. In *KDD*, pages 747–756, 2007.
[14] 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.
[15] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, pages 704–715, 2008.
[16] H. Tong, H. Qu, and H. Jamjoom. Measuring proximity on graphs with side information. In *ICDM*, pages 598–607, 2008.