EFFICIENT APPROXIMATION OF SOCIAL RELATEDNESS OVER LARGE SOCIAL NETWORKS AND APPLICATION TO QUERY ENABLED RECOMMENDER SYSTEMS

by

Pooya Esfandiar

B.Sc. in Computer Engineering, Sharif University of Technology, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

August 2010

Abstract

Social relatedness measures such as the Katz score and the commute time between pairs of nodes have been subject of significant research effort motivated by social network problems including link prediction, anomalous link detection, and collaborative filtering.

In this thesis, we are interested in computing: (1) the score for a given pair of nodes, and (2) the top-k nodes with the highest scores from a specific source node. Unlike most traditional approaches, ours scale to large networks with hundreds of thousands of nodes.

We introduce an efficient iterative algorithm which calculates upper and lower bounds for the pairwise measures. For the top-k problem, we propose an algorithm that only has access to a small subset of nodes. Our approaches rely on techniques developed in numerical linear algebra and personalized PageRank computing. Using three real-world networks, we examine scalability and accuracy of our algorithms as in a short time as milliseconds to seconds.

We also hypothesize that incorporating item based tags into a recommender system will improve its performance. We model such a system as a tri-partite graph of users, items and tags and use this graph to define a scoring function making use of graph-based proximity measures.

Exactly calculating the item scores is computationally expensive, so we use the proposed top-k algorithm to calculate the scores. The usefulness and efficiency of the approaches are compared to a simple, non-graph based, approach. We evaluate these approaches on a combination of the Netflix ratings data and the IMDb tag data.

Preface

This thesis is a result of a collaborative research with six other people, together with whom I prepared and submitted the following papers:

1. P. Esfandiar, F. Bonchi, D. F. Gleich, C. Greif, L.V.S. Lakshmanan, and B.-W. On, "Fast Katz and Commuters: Efficient Approximation of Social Relatedness over Large Networks," submitted to IEEE International Conference on Data Mining (ICDM'10), Australia, 2010.

2. P. Esfandiar, M. K. Tajer, D. F. Gleich, and L.V.S. Lakshmanan, "Evaluating graph-based proximity search for a recommender system with item tags," submitted to ACM International Conference on Web Search and Data Mining (WSDM'11), Hong Kong, 2011.

To put emphasis on the team work and acknowledge their efforts, I use plural pronouns when referring to the author(s). The contribution of each co-author is provided in details in the following.

Dr. Laks V.S. Lakshmanan was my supervisor in preparation of the papers and my master's thesis. The identification and design of the research that resulted in the first paper was mostly done by Laks V.S. Lakshmanan and Chen Greif.

Francesco Bonchi provided the required material and helped in conducting the experiments in the first paper. He also helped to finalize the manuscript. Byung-Won On helped in implementation of the first version of the experiments, and was involved in writing of older drafts of the first paper. Laks V.S. Lakshmanan mostly wrote up the data-mining-related parts of the first paper, and Chen Greif wrote the numerical-linear-algebraic parts.

David Gleich later added and implemented an idea for top-k queries in the first paper, which was made use of in the second paper. He also was involved in writing up both papers. Mohammad Khabbazhaye Tajer helped in the write up and implementation of experiments in the second paper, which was the extension of a course project done by me, Mohammad Khabbazhaye Tajer, and Nima Hazar under instruction of Laks V.S. Lakshmanan.

I implemented and ran experiments of both papers with help of David Gleich and Mohammad Khabbazhaye Tajer. I was in charge of performing surveys to make sure all related work is cited properly in both papers. I was involved in writing up both papers, especially the experimental results sections. I made sure the description of models and the implementation of them match. I was involved in analysis of the research data together with Laks V.S. Lakshmanan, Chen Greif, David Gleich, and Mohammad Khabbazhaye Tajer.

Parts of this thesis are verbatim or paraphrased duplicates of some paragraphs in the aforementioned papers, and most of the tables and figures are included in the papers too. The permission to reprint these materials is given by IEEE and ACM to the first author, provided that a proper copyright notice is included in the thesis. The following paragraphs are taken from IEEE and ACM websites without change.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor Dr. Laks Lakshmanan for his invaluable guidance and support during my studies. Were not his help to me, I could not finish this work. Special acknowledgements also go to Dr. Chen Greif and Dr. David Gleich who helped me a lot in preparation of the research papers. I also thank Dr. Greif for reading this thesis and for his insightful suggestions.

I finally appreciate all of my friends in data management and mining lab and staff in the department that created a supportive and friendly environment.

Pooya Esfandiar

The University of British Columbia
August 2010

To my beloved parents,
Faramarz and Mojgan,
and my little sister Shadi

Chapter 1

Introduction

1.1 Social Networks

A social network is "a network of social interactions and personal relationships" as defined by Oxford Dictionary [1]. Social networks have been studied and analyzed by sociologists since 1950s (e.g., see [6]).

The advent of large social networks (such as Facebook, MySpace, and Twitter) and the availability of large quantities of social interaction data (on movies, books, music, etc) have caused people to ask: what can we learn by mining this wealth of data? Measures of social relatedness play a fundamental role in answering this question. For example, Liben-Nowell and Kleinberg [26] identify a variety of topological measures as features for *link prediction*, the problem of predicting the likelihood of users/entities forming social ties in the future, given the current state of the network. The measures identified in [26] fall into one of two categories – neighborhood-based measures and path-based measures. It is known that the former are cheaper to compute, although the latter are more effective at link prediction. The best path based measurements from [26] are the Katz measure and the hitting time/commute time (HT/CT) measure. The details of these measures could be found shortly in the following subsections.

[1] http://www.oxforddictionaries.com

1.1.1 Katz Measure and Commute Time

The *Katz measure* (also called Katz status score, or just Katz) $K(x,y)$ between nodes x and y captures the connectivity between these nodes in terms of an ensemble of paths. The key intuition is that the more paths connecting x and y, the stronger their affinity, but also the shorter the paths, the more important the contribution of the paths to the affinity.

Katz [24] proposed a measure for gauging the importance of actors in a network. Given an undirected graph G, intuitively, the Katz status score between nodes x and y is a weighted sum over the ensemble of all paths between x and y. More precisely, let $path^i(x,y)$ denote the number of paths of length i between x and y. Then the Katz score is $score(x,y) = \sum_{i=1}^{\infty} \alpha^i \times paths^i(x,y)$, where $\alpha \in (0,1)$ is an attenuation constant. We can formulate the paths in terms of an $(n \times n)$ adjacency matrix of the graph. If $\alpha < 1$, this definition weighs shorter paths more heavily than longer paths. A given power k of the matrix A gives all the paths of length k. In matrix notation, the matrix of Katz scores between all pairs of nodes is given by

$$K = \alpha A + \alpha^2 A^2 + \cdots = (I - \alpha A)^{-1} - I.$$

Katz was interested in centrality measure of nodes, a measure signifying their global importance, so he proposed the column sums of the matrix K as the required centrality scores. More precisely, he defined $s(y) = \sum_{x=1}^{n} K_{x,y}$, where $s(y)$ denotes the Katz score of node y. These scores can be conveniently computed by solving the system of linear equations $(\frac{1}{\alpha} I - A^T)t = s$, where t is a vector of unknowns corresponding to the Katz scores to be computed and s is a vector with component s_i being the sum of entries in column i of A. Foster et al. [14] proposed an $O(n+m)$ algorithm for computing Katz centrality scores for nodes, where n (m) is the number of nodes (resp., edges) in the network.

The particular variant of Katz scores, adapted for the problem of link prediction, proposed by Newell and Kleinberg, goes back to the original matrix equation. Notice that according to this definition, the *Katz score* of a *pair of nodes* x and y is defined based on entries of this matrix. This is different from the node-wise centrality score defined by Katz in his paper. Computing the pairwise Katz score by explicitly computing matrix inverse takes $O(n^3)$ time which is impractical for

most real life networks. The computational time may be reduced if a column of the inverse is computed by solving a linear system with a standard basis vector on the right hand side. However, for computing the Katz score for a single pair of nodes, we can use a direct procedure, (i.e., one that does not produce a solution of a linear system as an intermediate step) and show that we can effectively use upper and lower bounds to estimate the error and converge to a result with a prescribed accuracy.

The hitting time from node x to y is the expected number of steps taken for a random walk started at x to reach y. At any current node v, the walk progresses to one of the neighbors chosen uniformly at random. The probability of transition is computed as follows. Let w_{ij} denote the weight of the edge (i, j) whenever the edge exists and is 0 otherwise (as a special case, all non-zero weights w_{ij} in the adjacency matrix may be 1). Then the probability of a transition from node i to j is $p_{ij} = w_{ij}/\sum_{\ell} w_{i\ell}$. Denote by P the resulting matrix. Notice that P is a stochastic matrix, i.e., every row sums to 1. The random walk characterized by P is described by $h_{ij} = 1 + \sum_{\ell} p_{i\ell} h_{\ell j}$, i.e., $H = I + PH$, where H denotes the matrix of hitting times. That is, $P = (I - P)^{-1}$. Note again the matrix inverse operation in this equation, similar to that for the Katz measure. However, unlike Katz, hitting time is not necessarily symmetric. A related quantity of interest is the commute time, which is defined as the sum of hitting times from x to y and from y to x and is symmetric. In this thesis, we have chosen commute time over hitting time for simplicity of computation, while it still serves as a proximity measure in social networks.

In previous research, both the Katz measure and the commute time measure have been shown to be effective at link prediction. A related task is to determine if a given link is peculiar or even anomalous. In [32], Rattigan and Jensen use a pairwise Katz score $K(x, y)$ to detect anomalous links, i.e., links with low likelihood of formation. More precisely, they treat a new link as anomalous if the Katz score between the endpoints is low. This motivates the problem of computing the score for a given pair of nodes.

Another recent paper [25] studies efficient computation of SimRank [22] for a given pair of nodes. Furthermore, commute time has also been applied in spectral clustering [30, 31]. There, the commute time between a node and a group of

nodes (e.g., a cluster) measures their affinity. This usage motivates computing the aggregate Katz score or commute time between a node and a set of other nodes.

Commute time has also been used for generating recommendations based on collaborative filtering [35], and for spectral clustering [30, 31]. For both link prediction and recommendation, another common task is: find the k best predictions or recommendations for a particular node. Thus, given a specific node, we explore computing the k-nearest-neighbors with respect to both Katz and a diffusion measure inspired by commute time.

In link prediction, anomalous link detection, and recommendation, the underlying graph is dynamic and evolving in time. These tasks require almost real-time computation because the results should reflect the latest state of the network, not the results of an offline cached computation. Therefore, calculation of these metrics must be as fast as possible. An alternative is to combine some offline processing with techniques to get fast online estimates of the scores. These techniques invariably involve a compromise between scalability of the approach (e.g., computing a matrix factorization offline) and the complexity of implementation (see [7, 23] for examples in personalized PageRank).

1.1.2 Bidirectional Diffusion Affinity

Another measure we investigate is inspired by commute time. In particular, we cannot state the top-k commute time scores as the solution of a single linear system (see Section 1.2 for details). Thus, we wanted a measure that was capturing something similar to commute-time, but could be computed with a single linear system.

Our new measure is defined by a diffusion in the graph. Consider the systems $(I-P)X = I$ and $(I-P^T)Y = I$ (See Section 1.1.1 for definitions). The matrices X and Y model a forward and backward diffusion from each node. (We discuss the singularity of these systems shortly.) Think of $X_{i,j}$ as the amount of "liquid" at node j when we inject 1 unit of "liquid" at node i and it diffuses according to P. Our new measure is $F = X + Y$.

Because bidirectional diffusion affinity is inspired by computing top-k commute times, we wanted to understand how the top-k commute time sets compared

with the top-k F scores. We decided to gauge this correspondence by computing exact commute times, which is of course feasible only on small graphs. For one small graph[2], we look at an empirical relationship between our bidirectional diffusion affinity F matrix and the exact commute time matrix C. In Figure 1.1, we explore the similarity in smallest commute time and highest bidirectional diffusion affinity; Left: For a line graph, we plot both the commute times and bidirectional diffusion affinity (F-measure) scores from a target node in the middle of the graph (in red). Right: Consider the k nodes with smallest commute time to a target node. Each box-plot shows the precision at k using the top-k nodes from our diffusion affinity measure. The plots aggregates 300 trials (for 300 different target nodes) and shows the median precision at k in red, and the 25th and 75th percentiles in the box. We exclude the neighbors of the target from the top-k set. This figure shows that, in the majority of experiments, the top-k sets of these two measures overlap by at least 75%, that is, the precision at top-k for most values of k is at least 75%.

1.1.3 PageRank

PageRank is a random-walk-based authority measure defined for nodes in a network. The PageRank score between two nodes is

$$R(u,v) = (1-\alpha)\sum_{\ell=0}^{\infty} \alpha^{\ell} \mathrm{Prob}_l(u,v)$$

where $\mathrm{Prob}_l(u,v)$ is the probability of random walk starting at u and ending at v in exactly l steps. If we fix v and look at the vector $R(\cdot,v)$, it is the vector of personalized PageRank scores where the reset step always moves to node v. Personalized PageRank (PPR) biases the authority towards a subset of nodes by defining a preference vector. The PPR vector v satisfies

$$v = \alpha P v + (1-\alpha)u$$

where $\alpha \in (0,1)$ is the damping or attenuation constant, and u is the preference vector. The matrix P has entries $P_{ji} = 1/d_i$ if node i links to node j, where d_i is

[2]The undirected connected component of Stanford CS webgraph, University of Florida sparse matrix collection.

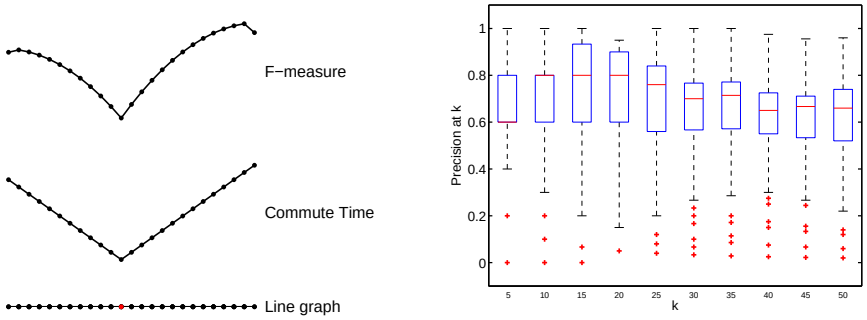


Figure 1.1: Bidirectional diffusion affinity plots

the degree of node i. If u has $\frac{1}{m}$ as m elements corresponding to m nodes, and 0 as other ones, the PPR score shows how the authority is propagated from those m nodes to each node. The propagation of authority could also be used to determine closeness of nodes [35]. Thus, we define $R(i,j) = v_i$ when $u = e_j$.

1.1.4 Other Measures

Works on other measures, mainly motivated by link prediction or collaborative filtering, abound. We just give a few examples. Spielman and Srivastava [39] develop a technique for computing the effective resistance (which is proportional to commute time) measure between nodes in amortized $O(\log n)$ time. Haveliwala [19] studies topic-sensitive PageRank, which is proposed as a useful measure for link prediction. Jeh and Widom [22] develop an algorithm for efficient computation of SimRank, a measure defined by a recurrence and is computationally intensive, while more recently, Li et al. [25] develop an efficient algorithm for computing the

SimRank for a given pair of nodes, avoiding the computation for pairs of nodes not needed for the query.

1.1.5 The Problems

As we explain shortly in Section 1.2, computing these measures is closely related to solving a linear system. Solution techniques for linear systems fall into two categories: iterative methods (e.g., conjugate gradient) or direct methods (e.g., Cholesky); see [16]. Each methodology leaves something to be desired in the context of this work.

Standard iterative methods solve a linear system with only matrix-vector products. These algorithms scale to large systems, but do not generally provide accuracy estimates of individual entries of the solution. Direct methods produce an easier-to-solve system by manipulating the entries of the matrix directly. However, computing these matrix factorizations requires $O(n^3)$ time and $O(n^2)$ memory in the worst case. Sparse factorization techniques improve performance empirically [12]; but for graphs with hundreds of thousands or millions of nodes, the time and memory required with these algorithms is prohibitive.

In this thesis, we seek approaches forming an attractive middle ground that at once provide a straightforward implementation and good scalability. We explore two cases where fast approximations of the proximity measures are possible:

- Given a pair of nodes x, y in a social network, find the proximity measure between x and y.
- Given a node x in a social network, compute the top-K nodes (in terms of proximity measures) associated with it.

In [41], the authors use a commute time kernel based approach to detect clusters and show that this method outperforms other kernel based clustering algorithms. The authors use commute time to define a distance measure between nodes, which in turn is used for defining a so-called intra-cluster inertia. Intuitively, this inertia measures how close nodes within a cluster are to each other. They follow a k-means approach to clustering and define the total intra-cluster inertia as follows. Let $\mathscr{C}$ be a clustering, i.e., an assignment of nodes to clusters. Then

$J(\mathscr{C}) = \sum_{C\in\mathscr{C}} \sum_{i\in C} ||x_i - g_C||^2$, where C is a cluster, x_i is the feature vector of node i, and g_C is the representative feature vector of cluster C and $||x_i - g_C||$ is the distance between the two vectors. The algorithm we propose for computing the Katz and commute time score for a given pair of nodes x, y extends to the case where one wants to find the aggregate score between a node x and a set of nodes S. Consequently, this has applications for finding the distance between a point and a cluster as well as for finding intra-cluster inertia.

1.2 Numerical Computation Remarks

In this section, computing the first three proximity measures (i.e. Katz score, commute time, and bidirectional diffusion affinity) is discussed. Please note that Personalized PageRank has been introduced in this thesis, but the approximation algorithms to calculate proximity measures are not applicable to it. It has only been used in our recommender system models. Lanczos and quadrature rules, as basic notions employed in our proposed algorithms are also introduced in the following.

1.2.1 Computing the Measures

Computing the pairwise Katz score by explicitly computing matrix inverse takes $O(n^3)$ time which is impractical for most real life networks. The computational time may be reduced if a column of the inverse is computed by solving a linear system with a standard basis vector on the right hand side.

Using a Neumann series expansion, for α sufficiently small, we have

$$(I - \alpha A)^{-1} = I + \alpha A + \alpha^2 A^2 + \cdots,$$

and hence K is component-wise positive. Based on this expansion, Foster et al. [14] proposed an $O(n+m)$ algorithm for computing Katz centrality scores for nodes, where n (m) is the number of nodes (resp., edges) in the network. This complexity is based on their claim, empirically verified, that their method converges in a small number of iterations.

The matrix K has a few useful properties. Consider the matrix that needs to be inverted, namely $B = I - \alpha A$. For α sufficiently small ($\alpha < 1/d_{\max}1$, where

$d_{\max} = \|A\|_1$ is the maximum degree or 1-norm of the matrix), B is a symmetric positive definite, diagonally dominant M-matrix [40], which asserts that (i) its diagonal is positive, (ii) its off-diagonal elements are non-positive, (iii) the diagonal elements in each row are larger than the absolute value of the sum of the remaining elements, and (iv) the inverse B^{-1} is component-wise positive. If $\alpha < (\max \lambda(A))^{-1}$, where $\lambda(A)$ is the set of eigenvalues of A, then B is a symmetric positive definite matrix. Computing a pairwise score amounts to computing a single entry of B^{-1}. Computing the k-largest scores involves computing the largest entries in a column of B^{-1}.

By definition, commute time is symmetric. Let C denote the commute time matrix of an undirected graph G, and $c_{i,j}$ be the commute time between a pair of nodes. We now review how to compute $c_{i,j}$. Let W be a weighted adjacency matrix. Typically, this matrix contains real values that hold the weight of an edge (zero if an edge does not exist). If D is the diagonal matrix holding the row sums of W, then the graph Laplacian is defined as $L = D - W$. Define the volume of a graph as $Vol(G) = \Sigma_i D_{ii}$. Then we have $c_{i,j} = Vol(G)(e_i - e_j)^T L^\dagger (e_i - e_j)$, where G is the graph, e_i and e_j are standard basis vectors, and $L^\dagger$ is the pseudo-inverse of the symmetric positive semidefinite graph Laplacian.

The graph Laplacian is singular since its row sums are all zero, which means it has a nontrivial null space that contains constant vectors. Standard techniques for dealing with this generate a shift that eliminates the zero eigenvalue, for example by projection onto the space orthogonal to the null space. Thus, one possible way of eliminating the null space (see, e.g., [34]) is by working with $(L + \frac{1}{n} ee^T)^{-1} - \frac{1}{n} ee^T$, where e is the vector of all 1s. That is, ee^T is a rank 1 correction associated with the null vectors, and the matrix to be inverted here is now positive definite. Another possible way of eliminating the singularity is simply by eliminating a row and a column of L.

We can compute bidirectional diffusion affinity with a single solve of the Laplacian system. Let $LZ = (D - W)Z = I$. Then $Y = DZ$ by direct substitution; and $X = ZD$ by direct substitution using $P = D^{-1}W$ and $WZ = DZ - I$. Furthermore, $F = DZ + (DZ)^T$ because Z is symmetric.

All of these linear systems are singular and moreover, none of the right-hand sides are compatible. Thus, "solutions" do not exist. We take each of these systems

as a least-squares problem instead. Consequently, $Z = L^{\dagger}$ corresponds to solving each least-squares problem with the minimum length solution [16]. This solution is unique, which allows us to uniquely define $F = DL^{\dagger} + (DL^{\dagger})^T$. We note, however, that this is a theoretical observation, and the pseudo-inverse is not explicitly computed in our experiments.

1.2.2 Lanczos Algorithm

A centerpiece in the methods we propose is the *Lanczos algorithm*. We therefore devote a special section to it. This algorithm (see, for example, [13, 16] for excellent and thorough descriptions) is a procedure of constructing an orthogonal sequence of vectors in a special linear space, which results in a *small* tridiagonal matrix that can be used in matrix computations related to eigenvalues and linear systems. Input for the algorithm is a matrix B, an initial vector q and a number of steps k. Upon exit, we have an $n \times (k+1)$ matrix Q_{k+1} with orthonormal columns and a $(k+1) \times k$ tridiagonal matrix $T_{k+1,k}$, that satisfy the relation

$$BQ_k = Q_{k+1}T_{k+1,k},$$

where Q_k is the $n \times k$ matrix that contains the first k columns of Q_{k+1}, and

$$T_{k+1,k} = \begin{bmatrix} \alpha_1 & \beta_1 & & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \ddots & \alpha_{k-1} & \beta_{k-1} \\ & & & & & \beta_{k-1} & \beta_{k-1} \\ & & & & & & \beta_k \end{bmatrix}.$$

The columns of Q_k form an orthogonal basis for the so called Krylov subspace

$$\mathscr{K}^k(B,q) = \text{span}\{q, Bq, B^2q, \ldots, B^{k-1}q\}.$$

where $span(v_1, ..., v_k)$ denotes the vector space spanned by the vectors v_i.

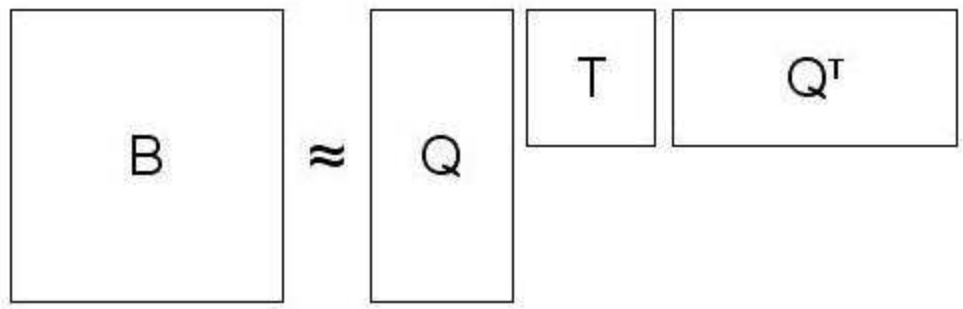


Figure 1.2: The Lanczos algorithm schema

It is worth explaining how the orthogonality relation is constructed. Basically, we perform the well known *Gram-Schmidt* process, and terminate it after a mere k steps. Starting off with the initial vector q, we normalize it: $q_1 = q\ /\|q\|$, so as to have a vector with unity norm. We then compute Bq_1, but instead of leaving it intact, we orthogonalize it against q_1 and normalize it to obtain q_2, so that q_1 and q_2 are now orthogonal, but span the exact same subspace as q and Bq. The process is now repeated, column by column. The symmetry of the matrix allows for constructing the orthonormal basis using short recurrence relations, which is reflected in the fact that $T_{k+1,k}$ is tridiagonal. (In the general nonsymmetric case we cannot expect to have this desirable situation; the tridiagonal matrix is replaced by an almost upper triangular matrix known as upper Hessenberg.)

Many modern methods in numerical linear algebra rely on the Lanczos algorithm for computing approximate eigenvalues or approximate solutions of linear systems. One way of understanding why this procedure can help to solve these problems, is by observing that in the limit $k = n$ we have an orthogonal similarity transformation of the form $B = QTQ^T$; see Fig. 1.2. (When $k < n$ we do not have equality.) Thus, the eigenvalues of B are preserved under this transformation, and if we were interested in a linear system solution, as follows from simple matrix algebra, a solution for $Bx = b$ could be obtained by solving $Ty = Q^T b$ and setting

$x = Qy$.

But what makes Lanczos attractive is *not* the exact similarity transformation that is obtained in the limit $k = n$. Rather, it is the good *approximation properties* that this procedure has for $k \ll n$. The matrix $T_{k+1,k}$ is small since $k \ll n$, but the spectrum if its $k \times k$ upper part (i.e. the same matrix, with its last row excluded) approximates that of the large $n \times n$ matrix B in a least squares sense; a detailed analysis is beyond the scope of this thesis, see, e.g., [13].

$1:\ q_1 = q/\|q\|_2, \beta_0 = q_0 = 0$
$2:\ \text{for } j = 1 \text{ to } k$
$3:\quad z = Bq_j$
$4:\quad \alpha_j = q_j^T z$
$5:\quad z = z - \alpha_j q_j - \beta_{j-1} q_{j-1}$
$6:\quad \beta_j = \|z\|_2$
$7:\quad \text{if } \beta_j = 0, \text{ quit}$
$8:\quad q_{j+1} = z/\beta_j$
$9:\ \text{end for}$

Figure 1.3: Lanczos(B, q, k).

Another crucial point here is that the matrix B does not necessarily have to be provided explicitly; a look at the algorithm (Figure 1.3) reveals that all we need here is a 'black box' routine that, given a vector x, returns a vector $y = Bx$. In other words, it is sufficient to have a routine that generates matrix-vector products, for any given vector. In any case, whether or not the matrix is available explicitly, a central point here is that the cost of matrix-vector products in $O(n)$. This is particularly important in the context of the problems discussed in this thesis, where the size of the social networks considered could easily be in the hundreds of thousands. Decomposition approaches are generally infeasible due to prohibitive storage and computational requirements. On the other hand, algorithms that rely on matrix-vector products are promising.

1.3 Social Search and Recommender Systems

Recommender systems provide recommendations (usually items from a large set that are hard to find otherwise) for users based on stored and/or acquired information such as their history, history of similar users, and explicit queries. The term social search is used to describe a type of web search that considers social information gathered from Web 2.0 applications [9].

Improving the usability of recommender systems and user satisfaction has always been one of the main concerns of the recommender systems community. The popularity of search engines suggests that, in general, users prefer easy-to-use ways of communicating with the system to express their current preferences.

In this thesis, we propose two models that integrate user history (of rating items) and keywords (a.k.a tags) assigned to items in order to make query-based recommendations. Our original motivation was to improve the usability of these systems by allowing users to issue keyword queries to a recommender system. For example, a user trying to find a family friendly movie would issue the query "family-friendly." In our system, we could use the tag "family-friendly" as the tag portion of the recommendation instead of the user's implied set of tags (as discussed in the previous two sections). Without this explicit information, we can only infer the current interests of users by complex statistical models, which are not necessarily accurate due to possible lack of sufficient information.

We identified three broad classes of related work for this section: social search, graph based recommendation, and query based hybrid recommendation, that appear in the following subsections.

1.3.1 Social Search

Many methods have been proposed to improve discovery of relationships from social data and enhance social search results. For example, [20] proposed FolkRank, a generalized link analysis approach (similar to PageRank), to compute strengths of each entity of the network. FolkRank computes the score of each entity based on its relationships with others and the strengths of the relationships that spread activation. Therefore, a tag stated to be strong by important users becomes strong, and a document strongly related to strong tags by strong users becomes strong itself.

In [5] the popularity of web pages, users, and annotations are captured simultaneously by SocialPageRank based on relationships of entities. The intuition behind this model is that the annotations are good summaries of web pages and popular web pages have higher annotation counts. The contribution of SocialSimRank [5] and SocialPageRank is that combining social score of a page with textual similarity of tags associating that web page to a query improves the quality of the results.

Chakrabarti [10] presents HubRank; a method for proximity searches in entity-relation (ER) graphs, which is fast and space efficient compared to previous proximity search algorithms. HubRank has a preprocessing phase which chooses a small fraction of nodes using query log statistics, and then computes and indexes certain random-walk fingerprints for that fraction of nodes in the multi-entity graph. At query time, a small active subgraph is identified and bordered by nodes with existing indexed fingerprints. These fingerprints are adaptively loaded and the remaining active nodes are then computed iteratively in order to calculate approximate personalized PageRank vectors.

Schenkel et al. [38] expand the scope of social search by collecting social information from LibraryThing[3]. They model social and semantic relationships among tags and items, and calculate the score of a document for a tag for each user based on these relations. The score of a document for a query, then, is produced by summing up the scores of that document for tags in the query. Similarly, [37] develop an incremental top-k algorithm considering strengths of relations among users and relations of different tags. They use a top-k threshold model and use social and semantic expansions in an incrementally on-demand manner to leverage social wisdom.

In [9], Carmel et al. try to take searcher's personal preferences into account by re-ranking search results. In order to re-rank search results for a user, they extract related users to that user and compute the similarity strength between them based on their social activity and re-rank the non-personalized search results. Therefore, documents that are strongly related to similar users get boosted in the personalized result.

Zhou et al. [43] combined language-modeling-based methods for information

[3]http://www.librarything.com

retrieval with social annotations in a unified framework to detect topical information in tags and integrate those information into traditional information retrieval techniques. In the first step they categorize users by domain and extract topics from contents and annotation of documents, and in the second step they incorporate user domain interests and topical background models to enhance document and query language models.

1.3.2 Graph-Based Recommendation Methods

Collaborative filtering is a popular approach to recommender systems in general [3]. Sarkar and Moore [35] motivated commute time in the context of collaborative filtering and proposed an interesting and efficient approach for finding approximate nearest neighbors with respect to a truncated version of the commute time measure. In [36], Sarkar et al. use their truncated commute time measure for link prediction over a collaboration graph and show that it outperforms personalized PageRank [22]. Saerens et al. [34] develop an application for spectral clustering based on principal component analysis of graphs, based on the Euclidean commute time distance between nodes, defined as the square root of the average commute time.

The relationships between users and items based on their rating preferences can be modeled as a bipartite graph. For example, in a movie recommender system, the nodes of the graph are users and movies, where a user is connected to a movie with a weighted edge if the user rated that movie and the rating is the weight of the edge. Gori et al. ([18]) present ItemRank, a random-walk based scoring algorithm that by using a similarity measure, ranks movies according to expected user preferences. Average commute time, PCA commute time distance, and elements of the graph Laplacian's pseudo-inverse are some of the measures characterizing similarity that Gori et al. used in ItemRank. The intuition behind ItemRank is that user preferences can spread through the correlation graph, so they used the PageRank algorithm because it has both propagation and attenuation properties.

In [15], the authors present a similar method for measuring similarity between any pair of nodes based on the number and length of the paths between them. They compute similarities based on a Markov chain model of random walk through the

graph by assigning transition probabilities to the edges and considering items as states of the Markov chain. They show that the pseudo-inverse of the Laplacian matrix of the graph is a valid kernel and can be considered as a similarity measure. Moreover, [8] present various measures and show that commute time is highly sensitive to nodes' degrees, which can be scaled to the stationary distribution of a simple random walk. They propose angular-based measures for recommendation and showed that its performance is much better than using commute time alone. An alternate approach proposed in [21] is to use link prediction measures, including Katz's score, to improve standard the accuracy of collaborative filtering.

Zhang et al. [42] model the label distribution for users and items and also pairwise relationships between users and items as a Gaussian Markov random field. They use this Gaussian semi-supervised model in order to solve the problem of top-k recommendations. Another contribution of their work is using an absorbing random-walk algorithm while considering degrees of nodes and directly generating top-k items without predicting ratings, just like our proposal.

1.3.3 Query Based Hybrid Recommenders

Many recent commercial movie recommendation systems[4] are designed around the idea of a "movie genome project" – a set of features describing the movies. These were most likely inspired by the success of Pandora's[5] "music genome project" used to build user customized radio-stations. The movie "genes" identified by these systems could easily serve as the tags in our approach. In particular, Jinni implements a query-based search and recommendation system similar to the future work we outline in Chapter 5.

To improve query based movie search results, [29] combines predicted user ratings with common search methods. In a more general setting, Cheng et al. [11] introduce a model for recommender system where attributes are added to item nodes as new nodes. Items are then sorted based on random walk proximity to a query, which could be a set of item nodes or attribute nodes. Multi-way clustering is used to reduce the amount of computation and hence the effectiveness and efficiency are

[4] See http://www.jinni.com, http://www.hellomovies.com, http://www.clerkdogs.com, and http://www.nanocrowd. com.

[5] http://www.pandora.com

improved.

1.4 Contributions

In this thesis, we make the following contributions:

- We propose a fast method for approximately computing the Katz score and the commute time score for a given pair of nodes based on the Lanczos/Stieltjes procedure [17]. Computing aggregate Katz or commute time scores between a node and a set of nodes is solved by similar algorithmic means. The algorithm we use produces lower and upper bounds on our measures.

- We provide algorithms to approximate the strongest ties (top-k) between a given source node and its neighbors, in terms of the Katz score and a diffusion measure. Our algorithms capitalize on the underlying graph structure and only access the out-links of a small set of vertices, producing good estimates of the final results.

- We propose and implement two models to integrate tags and ratings in a recommender system: the first is a straightforward combination of content scores (from tags) and predicted user score (from ratings); the second is novel and employs two commonly used graph proximity measures enhanced by a nearest-neighbor heuristic.

- We present an extensive experimental evaluation of the algorithms and models proposed in this thesis. Our experiments were conducted on five large real-world networks. We report the results of our evaluation in Chapter 4: our results attest to the scalability, effectiveness, and accuracy of our methods.

1.5 Thesis Structure

In this chapter, required background and related work were introduced as well as the problems being addressed, and the contributions made in this regard (see Section 1.4). In Chapter 2, our algorithms to approximate pairwise and top-k scores are discussed. In Chapter 3, a simple and a graph-based model for recommender

systems are introduced. The graph-based model makes use of the proposed top-k algorithm. Experiments and experimental results are provided in Chapter 4, and Chapter 5 gives the conclusions and future work.

Chapter 2

Approximation of Social Relatedness Measures

2.1 Algorithms for Pairwise Scores

We have earlier explained that computing the measure for a pair of given nodes boils down to computing the entry of an inverse of a matrix. Thus, let us define a matrix E, and for a given pair (i, j), we seek to approximate $E^{-1}(i, j)$.

Since E is symmetric positive definite, it admits an orthogonal spectral decomposition,

$$E = Q\Lambda Q^T,$$

where Q is an orthogonal matrix whose columns are eigenvectors of E with unity 2-norm, and Λ is a diagonal matrix with the eigenvalues of E along its diagonal.

Given this decomposition, we see that

$$u^T f(E)v = u^T Q f(\Lambda) Q^T v = \sum_{i=1}^{n} f(\lambda_i)\tilde{u}_i^T \tilde{v}_i,$$

where $\tilde{u}_i$ and $\tilde{v}_i$ are the components, respectively, of $\tilde{u} = Q^T u$ and $\tilde{v} = Q^T v$. The last sum can be thought of as a quadrature rule for computing integrals:

$$u^T f(E)v = \int_a^b f(\lambda) d\gamma(\lambda). \tag{2.1}$$

Here γ is a piecewise constant measure, which is monotonically increasing when $u = v$, and its values depend directly on the eigenvalues of E; λ denotes the set of all eigenvalues. γ is a discontinuous step function, each of whose pieces is a constant function. Specifically, $\gamma(\lambda)$ is identically zero if $\lambda < \min_i \lambda_i(E)$, is equal to $\sum_{j=1}^{i} \tilde{u}_j \tilde{v}_j$ if $\lambda_i \leq \lambda \leq \lambda_{i+1}$, and is equal to $\sum_{j=1}^{n} \tilde{u}_j \tilde{v}_j$ if $\lambda > \max_i \lambda_i(E)$.

Once we have identified that the problem may be posed as an approximation of an integral, we can apply a quadrature rule. In a few words, these are finite summation formulas that rely on the fact that computing a definite integral can be done by subdividing the given interval into small subintervals that are small enough so that each of them can be approximated by a function value. Sophisticated quadrature rules seek to evaluate exactly polynomials of order as high as possible; these are known as Gaussian rules and are fundamental in numerical computations; we use them in this thesis.

For any given vectors u and v and any symmetric matrix E, the following holds:

$$u^T E v \equiv \frac{1}{2}\left((u+v)^T E (u+v) - u^T E u - v^T E v\right).$$

Thus, without loss of generality, for computing the form $(E^{-1})_{i,j} = e_i^T E^{-1} e_j$ we can consider performing the easier computation of $u^T E^{-1} u$ with u being e_i, e_j and e_{i+j} in sequence.

In the case of computing elements of E^{-1} for the Katz score or Commute time, our function f is given by $f(E) = E^{-1}$. Since matrix operations are involved, it is convenient to approximate this function (or any other given smooth function for that matter) by a linear combination of polynomials. This gives the advantage of relying on matrix-vector products rather than the prohibitively costly operation of matrix inversion, though we settle for an approximation. In the context of our problem this works well for purposes of obtaining the value with prescribed accuracy of several decimal digits.

We need to compute an approximation for an integral of the form (2.1). An effective quadrature rule is

$$\int_a^b f(\lambda) d\gamma(\lambda) \approx \sum_{i=1}^{N} w_i f(t_i). \tag{2.2}$$

where $R[f]$ is the error and is given by

$$R[f] = \frac{f^{(2N)}(\eta)}{(2N)!} \int_a^b \left(\prod_{i=1}^{N} (\lambda - t_i) \right)^2 d\gamma(\lambda). \tag{2.3}$$

This formula is obtained by seeking to integrate exactly polynomials of as high a degree as possible. The *nodes* t_i and the *weights* w_i are unknown, and we set them to achieve this goal. (Note that these are not graph nodes but rather quadrature nodes.) If $N = 1$ then we have one node and one weight to determine. Linear functions are integrated exactly in this case. The more nodes and weights we have, the higher degree of polynomials we can integrate without error. But in the general case of an arbitrary function f, an exact formula cannot be developed. The formula for the error can then be obtained by the general theory of polynomial interpolation. In particular, observing that in general an integral of a function can be approximated by a polynomial, the error can be approximated by the integral of the error in polynomial interpolation. For the latter, it is possible to find an expression for the error by means in univariate calculus.

If *orthogonal* polynomials are used, then they admit a three-term recurrence relation that can be computationally exploited. Orthogonal polynomials satisfy the relation

$$\int_a^b p_i(x) p_j(x) \omega(x) dx = \delta_{i,j},$$

where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise. Here p_i and p_j are polynomials of degrees i and j respectively. The weight function $\omega(x)$ is nonnegative, and in the context of our problem it may be the measure $\gamma(\lambda)$.

Specifically, given orthogonal polynomials $\{p_j\}$ that are orthogonal with respect to the measure γ, we have the recurrence relation

$$\lambda_j p_j = (\lambda - \omega_j) p_{j-1} - \gamma_{j-1} p_{j-2}; \tag{2.4}$$

see, for example, [17, Section 3], or any textbook on numerical integration or orthogonal polynomials. As a result, we can iterate and be assured that as we progress along with the iteration, the recurrence relations stay short (i.e. three term recurrences); this presents an attractive feature in terms of required storage space.

If the nodes of the Gauss quadrature, namely t_j, are the eigenvalues of a tridiagonal matrix whose elements are exactly the γ_i and ω_i values; in other words the symmetric matrix is given by

$$T = \text{tri}(\gamma_i, \omega_i, \gamma_i).$$

The weights of the Gaussian quadrature rule are the squares of the first elements of the eigenvalues of T.

A central question that remains is how to construct the orthogonal polynomials in an efficient manner. Here is where the Lanczos algorithm [13, 16] comes in handy. It turns out that if orthogonal polynomials are used to compute the integrals, the Lanczos procedure can be used to construct a tridiagonal matrix one step at a time, whose eigenvalues are the nodes that are required. This is accomplished by using recurrence relations that are identical to recurrence relations that arise in the computation of the Gauss integrals for bilinear forms. Therefore, the iterates of Lanczos are vectors of the form

$$q_j = p_j(E)q_0,$$

where p_j are precisely the orthogonal polynomials defined in the quadrature rule. Hence, constructing approximations for $e_i^T E^{-1} e_i$ can be done by applying k steps of Lanczos and using the coefficients of the underlying tridiagonal matrix, to estimate the value of the quadratic form.

An important feature of the formula is that since $f(\lambda) = \frac{1}{\lambda}$ is a simple function, computing its derivatives is an easy task, and in fact we can get a precise idea of the error in the computation. Indeed, for this function f, we have that

$$f^{(2n)}(\lambda) = (2n)!\lambda^{-(2n+1)},$$

and therefore the sign of the error is readily available. We can use variants of the Gaussian integration formula to obtain both lower and upper bounds and 'trap' the value of the element of the inverse that we seek, between these bounds. The ability to estimate bounds for the value is powerful and allows also for effective stopping criteria for the algorithm. It is important to note that such bounds cannot

be obtained if we were to extract the value of the element from a column of the inverse, by solving the corresponding linear system.

Algorithm 1 shows the procedure. Input is a matrix A, a vector u, and estimates of extremal eigenvalues of A, a and b. The algorithm computes $\underline{b_j}$ and $\overline{b_j}$, lower and upper bounds for $u^T A^{-1} u$. The core of the algorithm are steps 3–6, which are nothing but the Lanczos algorithm. Notice in particular that ω_j and γ_j are the coefficients of the triangular matrix, what we called $T_{k+1,k}$ in Section 1.2. The values a and b are the endpoints of the quadrature interval, and may be difficult to compute, but in our case a can be taken as zero (a lower bound on the eigenvalues of A) and b can be taken as the maximum of the sum of absolute values of all rows of A; this gives an upper bound on the maximal eigenvalues. In line 7 we apply the summation for the quadrature formula. The computation needs to be done for upper bound as well as the lower bound; see lines 10 and 11, as well as 12 and 13. The remaining lines provide the actual values that 'trap' from above and below the required quadratic form. Lines 14 and 15 compute the required bounds.

2.1.1 Computational Complexity

The algorithm is based on the Lanczos procedure. It takes time $O(n\eta)$ where η is the number of iterations. In our experiments we found the number of iterations needed for convergence to be several orders of magnitude smaller than n. It is linear in the size of the matrix, but the number of iterations needs to be taken into account. Recall that our approach to Problem 2 is generic and can make use of a linear solver or an eigensolver and use it once until convergence. Given this, our algorithm for Problem 2 inherits the complexity of the solver used. For example, if we use the linear solver by Foster et al. [14], we will inherit their complexity of $O(n+m)$, where m is the number of edges in the graph. Finally, our algorithm for Problem 3 is based on invoking a computation similar to that for Problem 2 $(K+1)$ times. This is done in Steps 1 and 2 of that algorithm. In Step 3, the kn scores generated thus far are sorted, contributing an additional cost of $KnlogKn$. Thus this last algorithm has a time complexity of $O(K(n+m)+KnlogKn)$.

Algorithm 1 *GQL* – Method for Pairwise Problem

Initial step: $h_1 = 0$, $h_0 = u$, $\omega_1 = u^T A u$, $\gamma_1 = ||(A - \omega_1 I)u||$, $b_1 = \omega_1^{-1}$, $d_1 = \omega_1$, $c_1 = 1$, $\overline{d_1} = \omega_1 - a$, $\underline{d_1} = \omega_1 - b$, $h_1 = \frac{(A-\omega_1 I)u}{\gamma_1}$.
for $j = 2, ...l$ **do**
  $\omega_j = h_{j-1}^T A h_{j-1}$
  $\widetilde{h_j} = (A - \omega_j I)h_{j-1} - \gamma_{j-1}h_{j-2}$
  $\gamma_j = \left|\left|\widetilde{h_j}\right|\right|$
  $h_j = \frac{\widetilde{h_j}}{\gamma_j}$
  $b_j = b_{j-1} + \frac{\gamma_{j-1}^2 c_{j-1}^2}{d_{j-1}(\omega_j d_{j-1} - \gamma_{j-1}^2)}$
  $d_j = \omega_j - \frac{\gamma_{j-1}^2}{d_{j-1}}$.
  $c_j = c_{j-1}\frac{\gamma_{j-1}}{d_{j-1}}$
  $\overline{d_j} = \omega_j - a - \frac{\gamma_{j-1}^2}{d_{j-1}}$
  $\underline{d_j} = \omega_j - b - \frac{\gamma_{j-1}^2}{\underline{d_{j-1}}}$
  $\overline{\omega_j} = a + \frac{\gamma_j^2}{d_j}$
  $\underline{\omega_j} = b + \frac{\gamma_j^2}{\underline{d_j}}$
  $\overline{b_j} = b_j + \frac{\gamma_j^2 c_j^2}{d_j(\overline{\omega_j} d_j - \gamma_j^2)}$
  $\underline{b_j} = b_j + \frac{\gamma_j^2 c_j^2}{d_j(\underline{\omega_j} d_j - \gamma_j^2)}$
end for

2.2 Top-k Algorithms

In this section, we show how to adapt techniques for rapid personalized PageRank computation [4, 7, 27] to the problem of computing the top-k largest Katz scores and the bidirectional diffusion affinity measure F. These algorithms exploit the graph structure by accessing the edges of individual vertices, instead of accessing the graph via a matrix-vector product. They are "local" because they only access the outlinks of a small set of vertices and need not explore the majority of the graph.

The basis of these algorithms is a variant on the Richardson stationary method for solving a linear system [40]. Given a linear system $Ax = b$, the Richardson

iteration is $x^{(k+1)} = x^{(k)} + \omega r^{(k)}$, where $r^{(k)} = b - Ax^{(k)}$ is the residual vector at the kth iteration and ω is an acceleration parameter. While updating $x^{(k+1)}$ is a linear time operation, computing the next residual requires another matrix-vector product. To take advantage of the graph structure, the personalized PageRank algorithms [4, 7, 27] propose the following change: do not update $x^{(k+1)}$ with the entire residual, and instead change only a single component of x. Formally, $x^{(k+1)} = x^{(k)} + \omega r_j^{(k)} e_j$, where e_j is a vector of all zeros, except for a single 1 in the jth position, and $r_j^{(k)}$ is the jth component of the residual vector. Now, computing the next residual involves accessing a single column of the matrix A:

$$r^{(k+1)} = b - Ax^{(k+1)} = b - A(x^{(k)} + \omega r_j^{(k)} e_j) = r^{(k)} + \omega r_j^{(k)} A e_j.$$

Suppose that r, x, and Ae_j are sparse, then this update introduces only a small number of new nonzeros into both x and the new residual r. Each column of A is sparse for most graphs, and thus keeping the solution and residual sparse is a natural choice for graph algorithms where the solution x is localized (i.e., many components of x can be rounded to 0 without dramatically changing the solution). By choosing the element j based on the largest entry in the sparse residual vector (maintained in a heap), this algorithm often finds a good approximation to the largest entries of the solution vector x while exploring only a small subset of the graph. Let d be the maximum degree of a node in the graph, then each iteration takes $O(d \log n)$ time. We now discuss a few details of these algorithms for Katz and bidirectional diffusion affinity scores.

2.2.1 Katz Scores

For a particular node i in the graph, the Katz scores to the other nodes are given by $k_i = [(I - \alpha A)^{-1} - I]e_i$. Let $(I - \alpha A)x = e_i$. Then $k_i = x - e_i$. We use the above algorithm with $\omega = 1$ to compute x. For this system, x and r are always positive, and the residual converges to 0 geometrically if $\alpha < 1/\|A\|_1$. For larger α, we can show convergence if $\alpha < 1/\|A\|_2$. This result follows from the relationship between the Richardson iteration and gradient descent on the problem $\min_x x^T Ax - x^T b$. The update with the maximum value of $r_j^{(k)}$ always maintains a sufficient

decrease of $1/\sqrt{n}$. To terminate our algorithm, we wait until the largest element in the residual is smaller than a specified tolerance, for example 10^{-4}.

2.2.2 Bidirectional Diffusion Affinity Scores

As in previous sections, let D be the diagonal matrix of row-sums and d_i be the degree of node i. The diffusion scores from a single node are given by $Fe_i = Dx + d_i x$, where $x = L^{\dagger} e_i$, we now address how to compute x. Recall that $(D-A)x = e_i$ up to an unknown constant. Let $y = Dx$. We now have $(I - AD^{-1})y = e_i$. Solving this system instead of the Laplacian allows us to work with bounded quantities – everything is smaller than 1, for instance. However, both systems have a singularity and the residual will not converge to 0. Even given a solution $x^{\star} = (D - A - \frac{1}{n}ee^T)^{-1}e_i$, the residual in our system is $e_i - (I - AD^{-1})(Dx^{\star}) = (1/n)e$. (This follows from directly substituting into the residual equation and further showing that $e^T x^{\star} = -1$.) Using $\omega = 1$ in the top-k algorithm, the 1-norm of the residual is always 1. Consequently, we run the iteration until each element of the residual is smaller than τ/n, for values of τ larger than 1. Due to the nature of the iteration, values of τ much smaller than 2 often will not converge.

Chapter 3

Recommender System Models

3.1 A Simple Model Integrating Tags

There has been much work on keyword searching within the information retrieval literature. In the most popular scenario, items are ranked via a combination of query-dependent features and document-importance features. The idea is that a less precise match in a highly important document may trump a great match in a total stinker. Common query-based features are the TF-IDF score or the BM25 score [33] between a document and a query. Document-importance features take many forms. Possibly the most well-known are the PageRank scores associated with pages on the web, but domain specific heuristics, such as the number of document views, are equally valid.

Our proposed model for combining tags and recommender systems takes a similar approach to an information retrieval search. Instead of a query, we assume there is a set of tags associated with each user. In our experiments, these are the set of tags on all items the user has rated. Our problem setting is still different from classic keyword search in two main ways. First, our input data is different. In our case we are dealing with two matrices M_U (user/item rating matrix) and M_W (item/keyword occurrence matrix). Second, the ranking must be done with respect to the individual user issuing the query. In this setting, it has more in common with personalized search than standard keyword search, but as pointed out in Section 2, our problem is considerably different from personalized search in taking user

ratings into account.

Similar to the ranking described above, we use two main components in our score formula, one of which represents the score of every item regardless of the tag set but *with respect to the user issuing the query*. Second, we use the TF-IDF score of every item with respect to each tag. Let S_Q represent the TF-IDF score and S_C represent the predicted content score from a collaborative filtering approach. Both score values are then scaled to the $[0,1]$ interval and linearly combined through a β parameter. Let T represent the set of tags for the user, we define the score associated with user u and item i to be

$$\text{score}(u,i;T) = \beta \cdot S_Q(T,i) + (1-\beta) \cdot S_C(u,i). \tag{3.1}$$

This model, however, does not take into account indirect tag similarities. Therefore, it is especially sensitive to tag sparsity. This is due to the fact that every item's score with respect to the tags is calculated only based on those tags that appear in item's set. This might cause problems in situations where the set of tags assigned to every item is not expressive enough to describe all aspects of the item. In the next section, we proposed a more sophisticated and principled approach for combining the query relevance of an item with its user preference.

3.2 A Graph-Based Model Integrating Tags

3.2.1 The Data Structure

The data available to our proposed system is, as mentioned earlier, the user-item rating matrix and a list of tags for each item. We now model the input as a tripartite graph of user, item, and tag nodes. Edges connect users and items based on the available ratings, while edges between items and tags are simply defined using the item-tag matrix – the M_W matrix. The intuition behind this model is that similarity between items, induced by users or tags, is reinforced.

Let $U = \{u_1, u_2, \ldots, u_m\}$ be the set of users, $I = \{i_1, i_2, \ldots, i_n\}$ be the set of items, and $W = \{w_1, w_2, \ldots, w_k\}$ be the set of tags. Therefore, we can define an undirected graph $G = (V,E)$, where $V = U \cup I \cup W$ is the set of nodes and the

adjacency matrix is defined as:

$$A_{i,j} = \begin{cases} 1 & i \in U, j \in I, r_{ij} \geq \mu \text{ OR } i \in I, j \in U, r_{ji} \geq \mu \\ 1 & i \in I, j \in W, t_{ij} = 1 \text{ OR } i \in W, j \in I, t_{ji} = 1 \\ 0 & \text{otherwise} \end{cases}$$

where r_{ij} is the rating of item i_j by user u_i, t_{ij} is the element in i^{th} row and j^{th} column of M_W matrix. This means we add an edge between a user and an item only if the rating of the user on that particular item is at least μ, which in our case is set to 3 (on a scale of 1-5). This setting corresponds to picking all movies that the user "liked", "really-liked", or "loved" according to the Netflix rating scale. Defining edges between items and tags is done in the obvious way using the M_W matrix. Notice that we do not define any edges between users and tags, in keeping with our desire to make our model as general as possible in assuming as little information as possible.

3.2.2 The Proximity Measures

We are not the first to suggest graph based proximity measures for a recommender system. Bao et al. [5] propose a matrix-algebraic method to propagate similarity of users, items, and tags iteratively until convergence. Sarkar and Moore suggest that an approximated version of commute time could be used to measure similarity of users and items in a recommender system [35]. In our proposed graph-based model, we too assume that the proximity of nodes indicates similarity. From the variety of proximity measures proposed in social networks [26], we have selected the pair-wise Katz measure and the personalized PageRank [28] measure.

The Katz measure between vertices u and v is defined as

$$K(u,v) = \sum_{\ell=1}^{\infty} \alpha^{\ell} \text{Paths}_l(u,v),$$

where $\text{Paths}_l(u,v)$ is the number of paths of length l between two vertices and $0 < \alpha < 1$ is an attenuation factor. Likewise, PageRank is a random-walk-based authority measure defined for nodes in a network. The PageRank score between

two nodes is

$$R(u,v) = (1-\alpha)\sum_{\ell=0}^{\infty} \alpha^{\ell} \text{Prob}_l(u,v)$$

where $\text{Prob}_l(u,v)$ is the probability of random walk starting at u and ending at v in exactly l steps. If we fix v and look at the vector $R(\cdot,v)$, it is the vector of personalized PageRank scores where the reset step always moves to node v.

Recall our setting from the simple model, we assume each user is associated with a set of tags T based on their items. Therefore, the returned items should be as close to those tags as possible, as well as being close to the user itself. In order to define a score for each item, suppose that set of user tags is $u_t = \{t_1, t_2, \ldots, t_l\}$, we may define:

$$\text{score}(u,i;T) = \beta \cdot S(u,i) + (1-\beta) \cdot \sum_{t \in u_t} S(i,t), \tag{3.2}$$

where $S(j,k)$ is either of the two graph similarity measures defined above.

In order to compute pairwise Katz scores between the user and different items, we use the following approach. The pairwise Katz score between a user (u_i) and all other nodes including all items is a vector (x) found by $x = (I - \alpha \cdot A)^{-1} \cdot e_i$, where e_i is a standard basis vector whose i^{th} element (corresponding to user node u_i) is set to 1 and all other elements to 0, and A is the adjacency matrix of the graph. After finding x, we only use those similarity values that correspond to item nodes and normalize the vector. Calculating the similarity between every tag node and every item can be done in the same way.

Unfortunately, computing Katz score using the previous formula with matrix inversion is too expensive for large matrices. Instead, we approximate the solution of the linear system. That is, we define $B = (I - \alpha \cdot A)$, and the Katz score of all nodes with respect to u_i can be computed by solving the following linear system

$$B \cdot x = e_i.$$

However, even this is too expensive. In the next section we describe a technique to approximate the solution of this system by adapting methods for personalized PageRank [7, 27]. This technique only *explores a small set of nodes nearby vertex*

u_i to approximate the largest results.

Personalized PageRank scores satisfy a similar linear system. The PageRank scores between a user (u_i) and all other nodes satisfy

$$(I - \alpha P) \cdot x = (1 - \alpha) e_i,$$

where the matrix P has entries $P_{ji} = 1/d_i$ if node i links to node j, where d_i is the degree of node i. The algorithms in [7, 27] efficiently estimate the largest elements in x by only exploring a small set of nodes nearby u_i.

3.2.3 The Algorithm

To approximate the vector of Katz scores, $B \cdot x = e_i$, we use the top-k algorithm in Section 2.2. We briefly summarize its features here. Standard iterative methods for large linear systems employ a sequence of matrix-vector products to approximate a solution x. The algorithm employs a sequence of column queries from B instead. It is inspired by fast techniques for personalized PageRank computation [7, 27] where column queries correspond to out-link queries for a node of the graph. When applied to approximating Katz scores, column queries also correspond to out-link queries.

Results from personalized PageRank computation and Katz score computation demonstrate that when solutions of the linear system are localized (meaning only a few entries of x have non-trivial values), these algorithms only access a small set of distinct columns, although they may repeatedly query any particular column. In practice, the computation time for an approximate solution may be only slightly larger than *a single matrix-vector product*. Furthermore, this algorithm does not involve any preprocessing of the graph. It can be implemented straightforwardly whenever there is a technique to access the out-links from a node.

3.2.4 Nearest Neighbors Heuristic

Since we are only interested in the items that have not been rated by the querying user, those items are connected to the user through paths of at least 3 edges. Therefore, the contribution of these paths to Katz or PPR scores could be insignificant and perhaps lost in numerical computations or approximation. A heuristic

method to raise the score of items of potential interest to the user is adding edges from the querying user to a few other user nodes. A common measure for finding these nodes in neighbor-based recommender systems approaches is Pearson correlation coefficient. We add these edges before calculating the proximity measure, and remove them afterwards for queries in the future. The number of non-zeros added to the matrix is negligible, but it can significantly improve the quality. We only find nearest neighbors of the user among those users that have rated at least one common item with the querying user. Moreover, the number of nearest neighbors should be empirically determined, so we took 10 nearest neighbors based on smaller experiments.

Chapter 4

Experiments

In this chapter we report the experimental analysis we conducted. Our goals are: (i) to test the convergence speed; (ii) measure the accuracy and scalability of our algorithms; (iii) compare our algorithms against the standard conjugate gradient (CG) approach; and (iv) validate our recommender system models.

4.1 Experiment Settings and Networks Used

We implemented our methods in MATLAB and MATLAB mex codes. We used five real-world networks for our experiments: two citation-based networks based on publications databases, and three social network. The dataset statistics are reported in Table 4.1.

Table 4.1: Basic statistics about our datasets.

Graph	Nodes	Edges	Average Degree	Max Singular Value	2-core Size
dblp	93,156	178,145	3.82	39.5753	76,578
arxiv	86,376	517,563	11.98	99.3319	45,342
flickr	513,969	3,190,452	12.41	663.3587	233,395
netflix&imdb	200,000	25,554,966	255.55	2672.4586	187,890

DBLP coauthor graph - We extracted the DBLP coauthors graph from a recent snapshot (2005-2008) of the DBLP database (http://www.informatik.uni-trier.

de/~ley/db/index.html). We considered only nodes (authors) that have at least three publications in the snapshot. There is an undirected edge between two authors if they have coauthored a paper. From the resulting set of nodes, we randomly chose a sample of $100,000$ nodes, extracted the largest connected component, and discarded any weights on the edges.

arXiv coauthor graph - This dataset contains another coauthorship graph extracted by a snapshot (1990-2000) of arXiv (http://arxiv.org/), which is an e-print service owned, operated and funded by Cornell University, and which contains bibliographies in many fields including computer science and physics. This graph is much denser than DBLP. Again, we extracted the largest connected component of this graph and only work with that subset.

Flickr contacts - Flickr (http://flickr.com) is a popular online-community for sharing photos, with millions of users. The first graph we construct is representative of its social network, in which the node set V represents users, and the edge set E is such that $(u,v) \in E$ if and only if a user u has added user v as his/her contact. We start with a crawl extracted from Flickr in May 2006. This crawl began with a single user and continued until the total personalized PageRank on the set of uncrawled nodes was less than 0.0001. The result of the crawl was a graph with $820,878$ nodes and $9,837,214$ edges. In order to create a sub-graph suitable for our experimentation we performed the following steps. First, we created a graph from Flickr by taking all the contact relationships that were reciprocal, and second, we again took the largest connected component.

Netflix and IMDb -In order to run experiments to validate our recommender system models, we selected Netflix [2] and IMDb [1] datasets. Netflix provides a collection of user-movie ratings, but it lacks keywords or tags for each movie. Therefore, we joined movies from Netflix to IMDb's database to find a set of keywords for each movie. Our final data contains almost 1 million ratings from 175000 users on 5000 movies, and there are 20000 tags linked to these movies.

4.2 Experiments

4.2.1 Pairwise Approximation

We begin by studying the accuracy of the pairwise algorithms for Katz scores and commute times. For this task, we first compute an exact answer using the MINRES method [16] to solve the corresponding linear systems: $(I - \alpha A)x = e_i$ for Katz and $(L + \frac{1}{n}ee^T)x = (e_i - e_j)$ for commute time. We used a tolerance of 10^{-8} in these exact solutions. As a point of reference, these solutions typically required more than a thousand matrix-vector products to converge. Next, we run our pairwise method. Recall that using Algorithm 1 requires a lower-bound on the smallest eigenvalue of the matrix A. We use 10^{-4} for this bound. We terminate our algorithms when the relative change in the upper and lower bounds is smaller than 10^{-4} or the upper and lower bounds cross each other. We evaluate the accuracy at each iteration of Algorithm 1. Because our approach to compute Katz scores requires two applications of Algorithm 1, the work at each iteration takes two matrix-vector products. As described in previous chapters, our pairwise algorithm is closely related to iterative methods for linear systems, but with the added benefit of providing lower and upper bounds. As such, it's convergence closely tracks that of the conjugate gradient method, a standard iterative method. We demonstrate the parallels between the convergence of conjugate gradient and our techniques in the subsequent figures. We terminate conjugate gradient when the norm of the residual is smaller than 10^{-4}. The results are shown in Section 4.3.

4.2.2 Top-k Approximation

We now proceed to a similar investigation of the top-k algorithms for Katz scores and our diffusion measure. In this section, we are concerned with the convergence of the set of top-k results. Thus, we evaluate each algorithm in terms of the precision between the top-k results generated by our algorithms and the exact top-k set produced by solving the linear system. Here, natural alternatives are other iterative methods and specialized direct methods that exploit sparsity, and we again use conjugate gradient (CG) as an example of iterative methods. The latter are beyond the scope of this work, since they require a different computational treatment in terms of caching and parallelization.

Let T_k^{alg} be the top-k set from our algorithm and T_k^* be the exact top-k set. The precision at k is $|T_k^{\text{alg}} \cap T_k^*|/k$. We also look at the Kendall-τ correlation coefficient between our algorithm's results and the exact top-k set. This experiment will let us evaluate whether the algorithm is ordering the true set of top-k results correctly. Let $x_{k^*}^{\text{alg}}$ be the scores from our algorithm on the exact top-k set, and let $x_{k^*}^*$ be the true top-k scores. The τ coefficients are computed between $x_{k^*}^{\text{alg}}$ and $x_{k^*}^*$. Both of these measures should tend to 1 as we increase the work in our algorithms. However, some of the exact top-k results contain tied values. Our algorithm has trouble capturing precisely tied values and the effect is that our Kendall-τ score does not always tend to 1 exactly.

Recall that the basic element of work in our top-k algorithms is an operation that accesses the neighbors of a single vertex in the graph. To quantify this work, we evaluate the algorithm in terms of the total number of edges it evaluates. However, the algorithms in the previous section used iteration count. Each iteration involves examining each edge in the graph twice. Thus,

To compare our results with those in the previous section, we present the algorithm performance in *effective matrix-vector products*. An effective matrix-vector product corresponds to our algorithm examining the same number of edges as a matrix-vector product. In other words, suppose the algorithm accesses a total of 80 neighbors in a graph with 16 edges. Then this instance corresponds to 2.5 effective matrix vector products.

For our first set of tests, we let the algorithm run for a prescribed number of steps and evaluate the results at the end. In our runtime tests, we describe stopping criteria more precisely.

4.2.3 Query Enabled Recommender System

We implement two different methods; first the simple method using Pearson correlation as a similarity measure and use a weighted average of ratings of 10 most similar users who have rated the item whose score is being predicted. We have also implemented the proposed model exactly as described in Section 3.2 using Katz and PPR scores, both with and without the nearest neighbors heuristic. We use $\alpha = 10^{-4}$ as the attenuation factor of Katz, $\alpha = 10^{-1}$ for PPR. We vary β from 0 to 1 in increments of 0.1.

Missing Link Test

We randomly select 1000 users, and for each remove the edge between the user and a random item that has been rated 5. We then apply our approaches to the dataset to retrieve an ordered list of items. For each user, we form the set of tags of all movies they liked. Then we randomly permute this set. We perform tests using the first 1, 4, 7, and 10 tags in the randomly permuted set. By design then, the test with 7 tags includes all the tags used in the test with 4.

Ideally, our algorithm will place the removed item into the top set of results. Thus, we look at the top 10 and 25 items with the highest scores from our method. (These thresholds were chosen because they are common result set sizes on the web.) We then calculate the ratio of the number of times that the removed item appears in the top-k list to the number of trials and call it the *hit rate*. We also compare the run time of different approaches.

Hybrid Recommender Test

We randomly separate 90% of the ratings for the training set and the remaining 10% for the testing set. From the testing set, we choose 1000 users who have more than 20 ratings in the testing set. For each user, we make a query using all tags of all rated items by the user. We thus have an ordered list of items rated by the users in the test set, according to the actual ratings ($L1$), and retrieve an ordered list of same items according to the scores ($L2$).

We take $L1$ as the ground truth and measure the effectiveness of approaches by comparing $L2$ to it. An effective hybrid recommender system should be able to return an $L2$ list as similar to $L1$ as possible. Among different measures to compare these two list, we have used precision@k, mean average positions (MAP), mean reciprocal ranks (MRR), and normalized discounted cumulative gain (nDCG). We briefly recall these measures below:

Precision@k This is simply the ratio of relevant items in the retrieved list to all retrieved items. In calculation of precision@k, we assume items rated 3 and more are relevant.

MAP While precision is a well-known metric for evaluating an ordered list of

items returned by a query, MAP is the average of precision@k values for all possible k values.

MRR For each user in the test set, the rank of first true relevant item in the list returned to the user is desired to be lower. The average of reciprocal of these ranks is thus a proper metric for this study. Items rated 3 or more are considered relevant.

nDCG Discounted cumulative gain is defined as:

$$DCG = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log_2 i},$$

where $rel_i \in 0,1$ indicates if i^{th} item is relevant. Normalized DCG is calculated by dividing DCG by the maximum DCG score an algorithm could achieve given the relevancy information beforehand.

4.3 Results

4.3.1 Pairwise Approximation

Katz scores For convergence of the Katz scores, we use a value of α that makes $B = I - \alpha A$ nearly indefinite. Such a value produces the slowest convergence in our experience. The particular value we use is $\alpha = 1/(\|A\|_2 + 1)$. For a single pair of nodes in arxiv, we show how the upper and lower bounds "trap" the pairwise Katz scores in Figure 4.1 (left). At iteration 13, the lower bound approaches the upper bound. Beyond this point the algorithm converges quickly. In Figure 4.1 (right), we show the convergence of both bounds to the exact solution. Both the lower and upper bounds converge identically. We show similar convergence results for the other graphs in Figure 4.2.

In comparison with the conjugate gradient method, our pairwise algorithm is slower to converge. For these problems, we also evaluated techniques based on the Neumann series for $I - \alpha A$, but those took over 100 times as many iterations as CG or our pairwise approach. While the conjugate gradient method appears

to outperform our pairwise algorithms here, the experiments with commute time illustrate a case where it is difficult to terminate CG early.

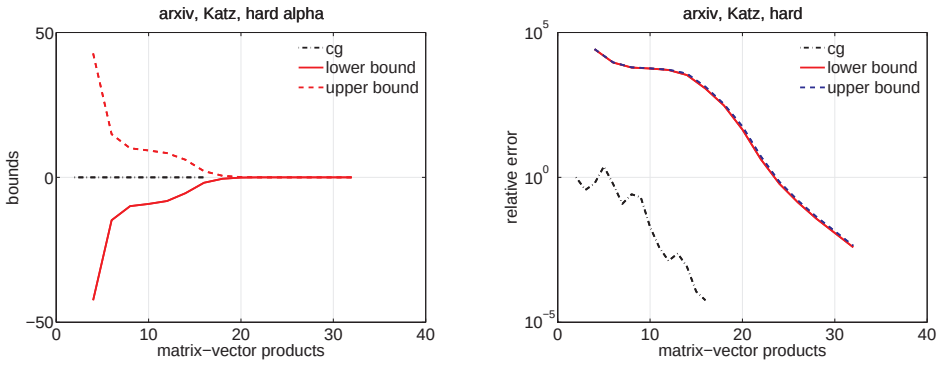


Figure 4.1: Convergence results for pairwise Katz on ArXiv.

Commute time In Figure 4.3, we show how commute time converges for the same pair of nodes in arxiv. Again, the left figure shows the convergence of the upper and lower bounds, and the right shows the convergence of the error. Whereas Katz took only a few iterations, computing pairwise commute times requires a few hundred iterations. A notable result is that the lower-bound from the quadrature rule provides a more accurate estimate of commute time than does the upper bound. See the curve of the lower bound in Figure 4.3(right); and also see the additional results in Figure 4.4 for other cases when this occurs. This observation suggests that using the lower bound as an approximate solution is probably better for commute time.

Note that the relative error in the lower-bound produced by our algorithm is almost identical to the relative error from CG. This behavior is expected in cases

where the largest eigenvalue of the matrix is well-separated from the remaining eigenvalues – a fact that holds for the Laplacians of our graphs. When this happens, the Lanczos procedure underlying both our technique and CG quickly produces an accurate estimate of the true largest eigenvalue, which in turn eliminates any effect due to our initial overestimate of the largest eigenvalue. (Recall from Algorithm 1 that the estimate of b is present in the computation of the lower-bound $\underline{b_j}$.)

Here, the conjugate gradient method suffers two problems. First, because it does not provide bounds on our score, we are unable to terminate the algorithm until the residual is small. Thus, the conjugate gradient method requires about twice as many iterations as our pairwise algorithms. Note, however, this result is simply a matter of detecting when to stop – both conjugate gradient and our lower-bound produce similar relative errors for the same work. Second, the relative error for conjugate gradient displays erratic behavior. Such behavior is not unexpected, because conjugate gradient optimizes the A-norm of the solution error and it is not guaranteed to provide smooth convergence in our norm. These oscillations make early termination of the CG algorithm problematic, whereas no such issues occur for the upper and lower bounds from our pairwise algorithms.

Runtime Finally, we present the runtime of our pairwise method in Table 4.2. We explore two cases for Katz:

$$\begin{aligned} \text{easy-}\alpha \quad & 1/(10\|A\|_1 + 10) \text{ and} \\ \text{hard-}\alpha \quad & 1/(\max(\lambda(A)) + 1). \end{aligned}$$

The former should converge more quickly than the latter. We note that estimating either of these values is computationally inexpensive. For each graph, we evaluate the runtime on three pairs of nodes. These pairs were chosen such that there was a high degree-high degree pair, a high degree-low degree pair, and a low degree-low degree pair. The results show the impact of these choices. As expected, the easy-α cases converged faster and commute time converged slower than either Katz score. In this small sample, the degree of the pairs played a role. On Flickr, for example, the low-low pair converged fastest for Katz, whereas the high-low pair converged fastest for commute time. The solution tolerance was 10^{-4}. We do not report separate computation times for the conjugate gradient method, but note that

Table 4.2: Runtime (in seconds) of the pairwise algorithms for Katz scores and commute time. See the text for a description of the cases.

Graph	Pairs	Score		
		Katz		Commute
		easy-α	hard-α	
arxiv	High, high	0.6081	2.6902	24.8874
	High, low	0.6068	2.3689	19.7079
	Low, low	0.3619	0.5842	10.7421
dblp	High, high	0.3266	1.7044	10.3836
	High, low	0.3436	1.3010	8.8664
	Low, low	0.2133	0.5458	8.3463
flickr	High, high	5.1061	12.7508	227.2851
	High, low	4.2578	11.0659	82.0949
	Low, low	2.6037	3.4782	172.5125

the previous experiments suggest it should take about half the time for the Katz problems and about twice as long for the commute time experiments.

4.3.2 Top-k Approximation

Katz scores We first present the convergence results for computing Katz scores. In Figure 4.5, we plot the convergence of the top-k set for $k = 10, 25, 100$, and 1000 for a single node in arxiv. The left figure plots the precision at k, and the right figure plots the Kendall-τ correlation with the exact top-k set. Both of these measures trend to 1 quickly. In fact, the top-25 set is nearly converged after the equivalent of a single matrix-vector product – equivalent to just one iteration of the CG algorithm. We show results from the conjugate gradient method for the top-25 set after $2, 5, 10, 15, 25$, and 50 matrix-vector products.

Figure 4.6 presents examples from the other graphs where we observe convergence even more quickly. On the dblp graph, the top-k algorithm produces almost the exact Katz top-k set with just slightly more than 1 effective matrix-vector product. For flickr, we see a striking transition around 1 effective matrix-vector product,

when it seems to suddenly "lock" the top-k sets, then slowly adjust their order.

In all of the experiments, the CG algorithm does not provide any useful information until it converges. Our top-k algorithm produces useful partial information in much less work and time.

Diffusion affinity We next investigate the convergence of the diffusion affinity scores. See Figure 4.7 for the same figure plotted with the diffusion affinity scores. These values take longer to converge. Our top-k algorithm begins identifying the correct set after the equivalent of one matrix-vector product of work, and improves from that starting point. See Figure 4.8 for additional convergence examples. Note that top-1000 set for dblp degrades with additional work. We conjecture this occurs once the residual vector becomes mostly uniform. At this point, our algorithm has actually converged. If we continue to run it, it introduces small errors into the uniform residual, which then cause other small errors. Over many steps, these small errors erode the quality of the solution. We observe no decay in the top-10 and top-25. Thus, the errors are concentrated in smaller values. For the flickr experiment in the same figure, we observe slower convergence of the top-k sets. But, just like in the case for Katz, the top-k sets seem to converge slightly before the order.

For the arxiv experiment, we see that CG does provide useful intermediate information; whereas for both dblp and flickr, it does not and only shows useful information after, or close to, the convergence point of the algorithm. In comparison with CG, we are able to estimate the top-25 set much faster. For dblp, we get it right with about one-tenth the work on CG.

Runtime We conclude our empirical evaluation with the runtime of each method. We terminate the Katz algorithm when the largest element in the residual vector is smaller than $10^{-4}\alpha d_u$ where d_u is the degree of the source node. For most of the experiments, this setting produced a 2-norm residual smaller than 10^{-4} – the same convergence criteria for CG. We terminate the diffusion affinity top-k algorithm when the largest element is smaller than $3/n$ (where n is the number of nodes in the graph). Just as in the previous section with the pairwise algorithm, we use the

Table 4.3: Runtime (in seconds) of the top-k algorithms for Katz with an easy α, a hard α, and for the diffusion affinity measure.

Graph	Degree	Score		
		Katz		Diffuse
		easy-α	hard-α	
arxiv	High	0.0027	0.2334	4.4566
	Low	0.0003	0.2815	4.4889
	Low	0.0004	0.5315	4.5609
dblp	High	0.0012	0.0163	0.8112
	Low	0.0011	0.0161	0.8037
	Low	0.0007	0.0173	1.0654
flickr2	High	0.0741	0.0835	137.8290
	Low	0.0036	36.2140	137.3796
	Low	0.0040	0.0063	89.4775

same easy-α and hard-α. In this case, we characterize each source node as low or high degree. The timing results in Table 4.3 match the convergence results and show that the diffusion scores require more time to compute. Interestingly, it seems the hard-α for Flickr does not affect the convergence of all nodes equally. The first low-degree node required 36 seconds whereas the second required 0.0063 seconds.

4.3.3 Query Enabled Recommender System

Figure 4.9 shows that the best β value for combining the scores due to collaborative filtering and keyword or tag relevance (when query size is 1, at which the plot is more demonstrative) is neither 0 nor 1. Therefore, their combination is showing a better performance than the parts alone. Several observations can be made from the figure. The nearest neighbor heuristic improves hit rate significantly in case of PPR but hardly makes a difference in case of Katz. The most effective value of β is about 0.8.

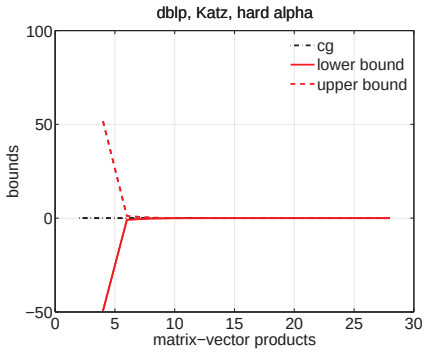
Figures 4.10 and 4.11 report top-k hit rate at k = 10 and 25 for a few query sizes. As is seen in these figures, the algorithm very well captures the removed item when the query size is around 4 words, but for larger queries, the returned results may be too broad to include the removed item. In scarce queries, PPR, enhanced PPR, and especially the simple model do better than Katz approaches.

	MAP	MRR	Precision	nDCG
Simple	0.0463	0.1606	0.2077	0.0829
Katz	0.1382	0.5874	0.3306	0.2406
PPR	0.0333	0.1002	0.1693	0.0358
Katz+NN	0.1400	0.5912	0.3330	0.2435
PPR+NN	0.0357	0.1051	0.1713	0.0385

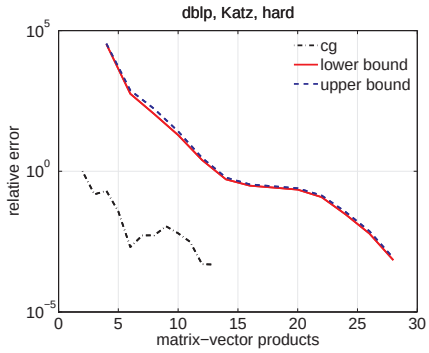
Table 4.4: Comparison of mean average precision (MAP), mean reciprocal rank (MRR), precision@k (k=25), and normalized discounted cumulative gain (nDCG) for different approaches.

Figure 4.12 shows the run time of different approaches in seconds. Although PPR has a rather better performance, it takes more time to reach the same accuracy as Katz does. It is also obvious that enhancing the algorithm with the nearest neighbors heuristic does not cause a significantly longer run time. Please note that shorter run times of the simple approach is a result of preprocessing the item similarities (which took a few hours; that means it is not practical for query time) is not very flexible with dynamically changing data, unlike graph-based models that simply use the latest state of graphs.

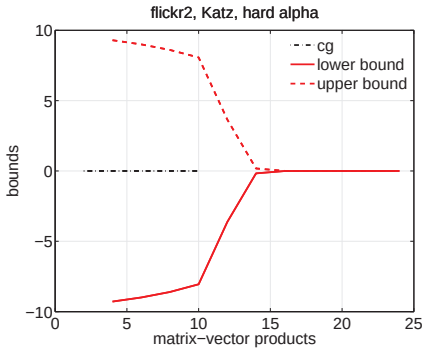
Results from the hybrid recommender system experiment are shown in Table 4.4. The nearest neighbors heuristic has slightly improved both Katz and PPR approaches in all the measures used in the experiment. The difference between Katz and PPR is significantly large, and thus Katz is showing a much better quality of returned items compared to the simple model and PPR.



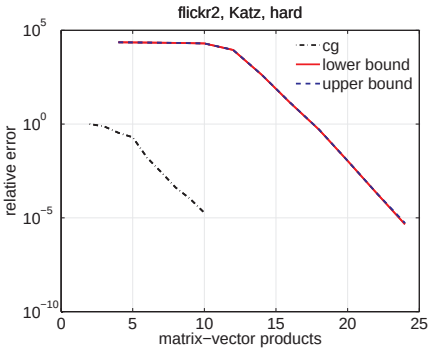
(a) dblp bounds



(b) dblp error



(c) flickr bounds



(d) flickr error

Figure 4.2: More convergence results for pairwise Katz in the hard α case on DBLP and Flickr

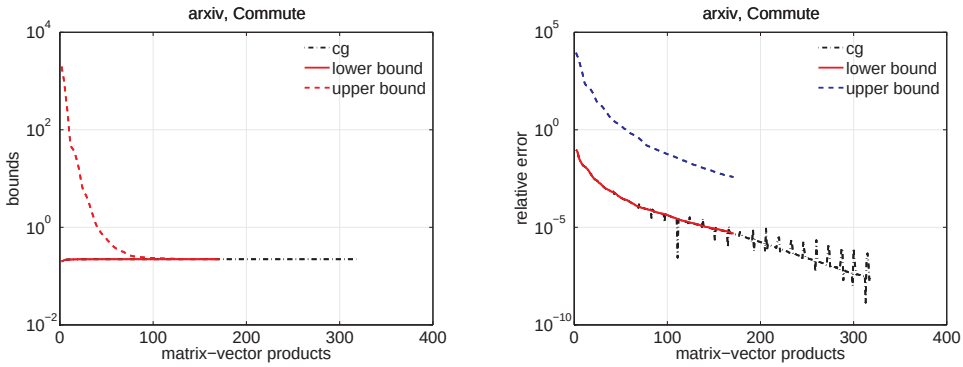
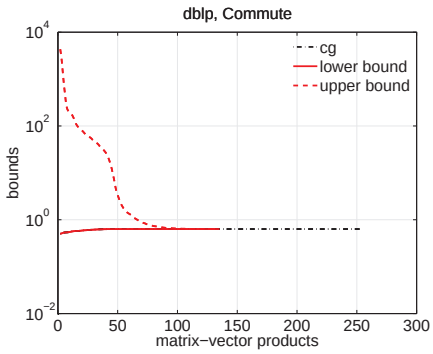
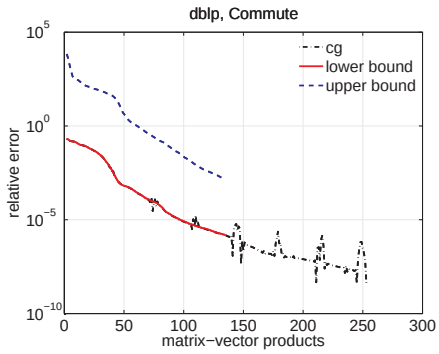


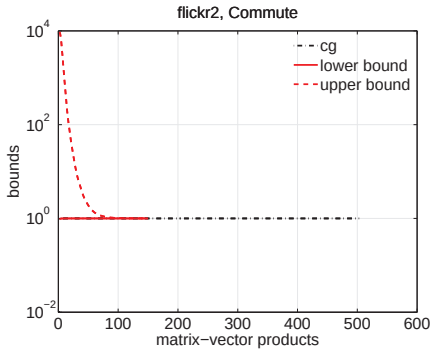
Figure 4.3: Convergence results for pairwise commute time on ArXiv.



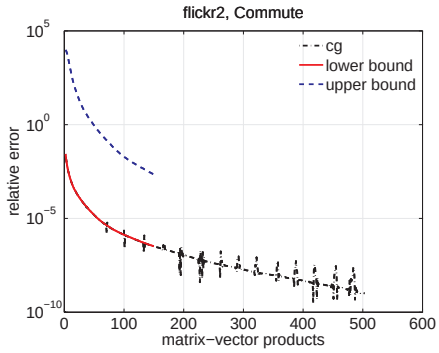
(a) dblp bounds



(b) dblp error



(c) flickr bounds



(d) flickr error

Figure 4.4: More convergence results for pairwise commute time case on DBLP and Flickr.

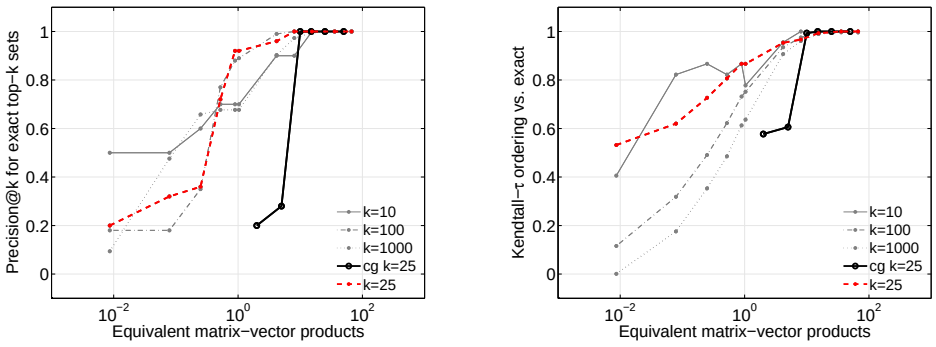
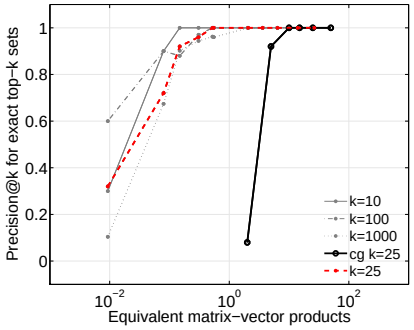
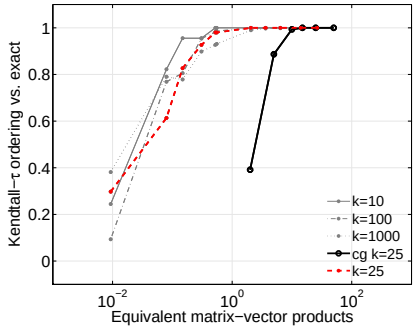


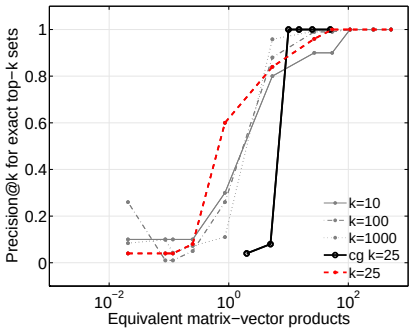
Figure 4.5: Convergence of the our top-k algorithm for the top-k Katz neighborhood of a single node in arxiv using the same value of α as Figure 4.1.



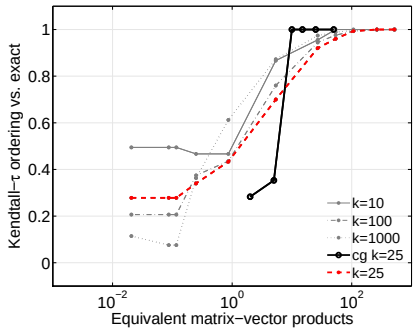
(a) dblp precision



(b) dblp τ



(c) flickr precision



(d) flickr τ

Figure 4.6: More convergence results for top-k Katz in a hard α case on DBLP and Flickr.

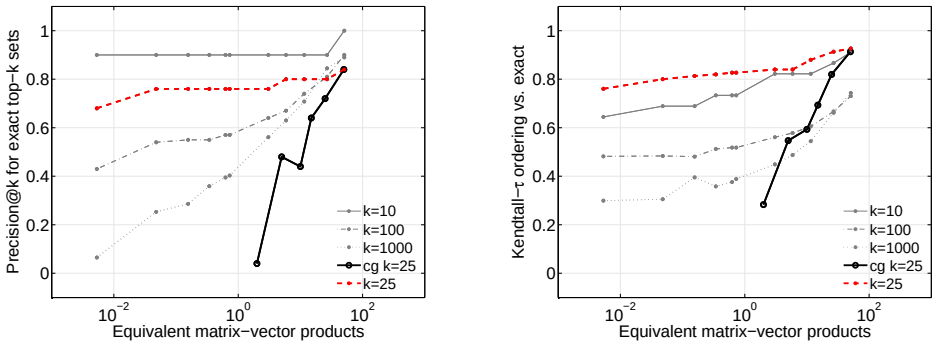
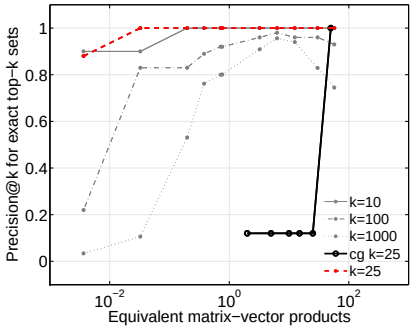
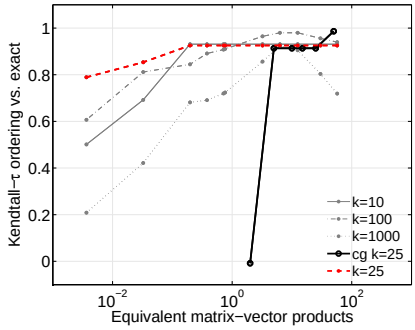


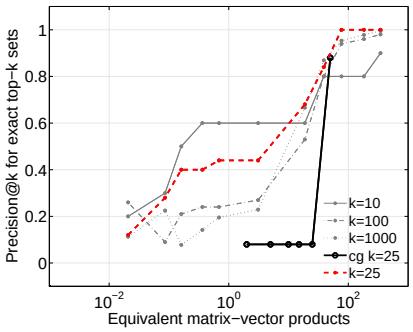
Figure 4.7: Convergence of the our top-k algorithm for the top-k diffusion affinity neighborhood of a single node in arxiv.



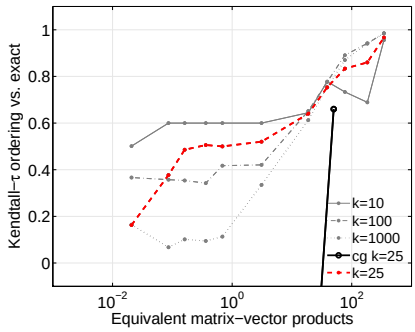
(a) dblp precision



(b) dblp τ



(c) flickr precision



(d) flickr τ

Figure 4.8: Convergence results for top-k diffusion affinity on dblp and flickr.

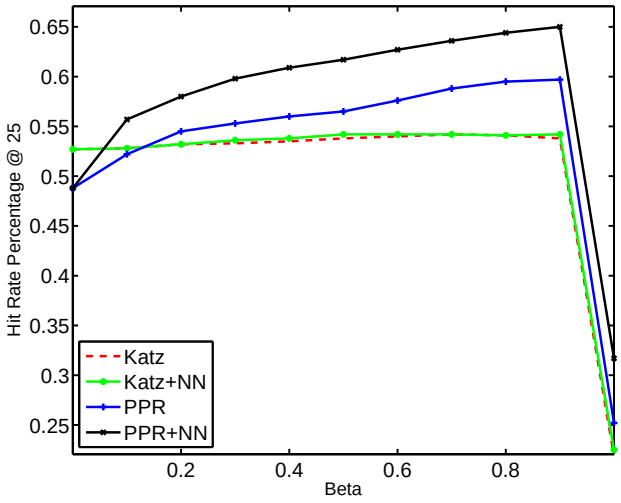


Figure 4.9: Hit rate vs. β

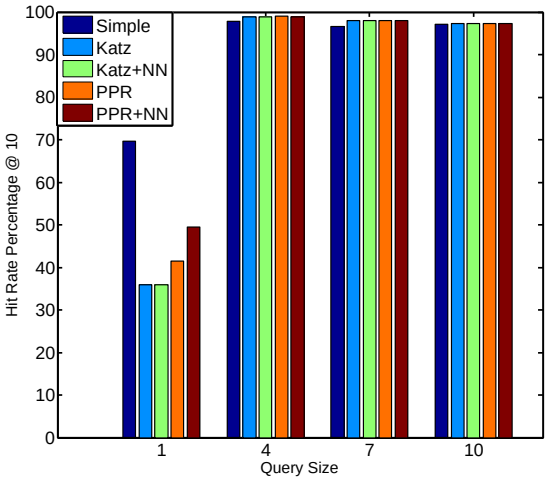


Figure 4.10: Hit rate of approaches at top-10

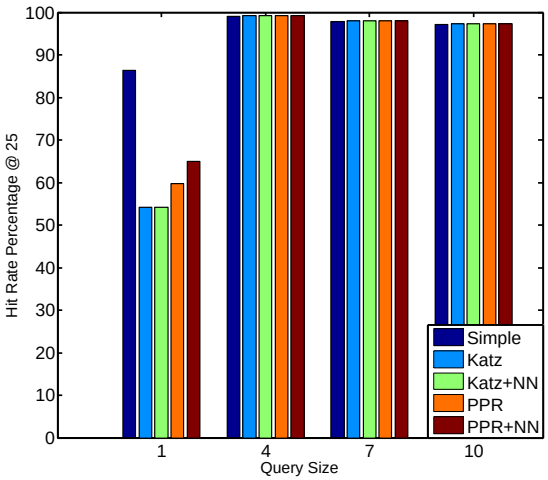


Figure 4.11: Hit rate of approaches at top-25

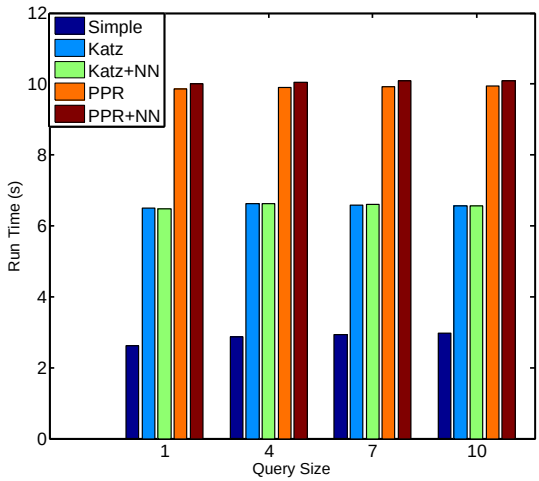


Figure 4.12: Run time of approaches

Chapter 5

Conclusion and Future Work

5.1 Conclusions

Measures based on ensembles of paths such as the Katz and the commute time have been found useful in several applications such as link prediction, anomalous link detection, and collaborative filtering. In this thesis, motivated by applications, we focused on two problems related to fast approximations for these scores.

- Finding the score between a specified pair of nodes: We have proposed an efficient algorithm to compute it and also obtain upper and lower bounds, making use of a technique for computing bilinear forms using a Lanczos-Stieltjes procedure – a combination of the Lanczos procedure for partial reduction to tridiagonal matrices with Gauss/Stieltjes quadrature rules. It is based on matrix-vector products and is linear in the dimension of the problem as long as the number of iterations is small (which we found was the case in our experiments). Our algorithm readily extends to the case of finding the aggregate score between a node and a set of nodes.

- Finding the top-k nodes that have the highest scores with respect to a given source node: Here, we used a bidirectional diffusion affinity measure inspired by commute time. We proposed a top-k algorithm based on a variant of the Richardson stationary method for solving a linear system.

We have conducted a comprehensive set of experiments on three real-world datasets and obtained many encouraging results. Our experiments demonstrate the good scalability of the proposed method to very large networks, without giving up much accuracy with respect to the exact methods (that are infeasible on such networks).

We also proposed the idea of combining tags and collaborative filtering in order to improve usability of recommender systems. We first described a simple model and then proposed a graph-based model based on the proposed top-k algorithm.

We empirically evaluated the approaches in terms of their capacities of performing as hybrid recommender systems using a combination of two real-world datasets and two common proximity measures. We identified the weaknesses and strengths of our approach and will provide concrete ideas in order to improve them for the future in the following section.

5.2 Future Work

Our future work will explore further improvements to the proposed approximation algorithms and extensions to non-symmetric measures such as hitting time. Also, our algorithms easily adapt to graphs stored in highly-scalable link databases or map-reduce environments and we hope to investigate applications in these settings.

Moreover, it is useful to investigate whether any of our techniques can be adapted to solve nonsymmetric problems, such as hitting time. In the nonsymmetric case the ability to use short recurrence relations is lost, and one may need to replace the Lanczos process by more costly approaches that entail higher memory requirements. It is challenging, but new results might lead to a set of tools that will help design efficient approximation algorithms for a suite of measures for random walk models.

The area of hybrid recommender systems is ripe for future work. The current thesis is a stepping stone on the path towards the vision for a query enabled recommender system. Currently, we use all of the tags in the user profile (via the liked movies) as tags used for recommendation. There are certainly better ways of choosing a subset of the most important tags for every user in order to design a better system. One possible way of doing this would be grouping items into two different categories for the user, like and dislike. We could as well group them into

multiple categories according to rating levels. Having done this, each of the categories could be considered as a class and the ***mutual information*** between every tag and every class could be used in order to select the most important tags of the user for every class. Using tags that have a high mutual information with "like" class should improve the performance.

Similarly, we can define the keyword query in such a way that it results in diversification of recommendations. In order to do this, first we can either cluster the tags or alternatively discover some topics in tags using their co-occurrences in items. Then tag selection could be done again based on the mutual information between the tag and different topics or clusters. Defining a keyword query that contains the best representatives from each topic has the effect of giving items of different types (according to their contents) the opportunity to get the chance of being recommended to the user. On the other hand, the collaborative filtering component of the scoring function will only assign high scores to the items that the user will like and altogether a diverse set of items which are also of user's interest will be returned.

A final limitation of the current system is that there is no way to incorporate information on the movies that a user dislikes. Such information is easy to include in our formulation by subtracting graph proximity scores associated with disliked movies.

We hope to investigate all of these ideas in the future.

Bibliography

[1] The Internet Movie Database. http://www.imdb.com.

[2] The Netflix Challenge. http://www.netflixprize.com.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[4] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006.

[5] S. Bao, G. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su. Optimizing web search using social annotations. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 501–510, New York, NY, USA, 2007. ACM.

[6] J. A. Barnes. Class and committees in a norwegian island parish. *Human Relations*, 7:39-58, 1954.

[7] P. Berkhin. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Mathematics*, 3(1):41–62, 2007.

[8] M. Brand. A random walks perspective on maximizing satisfaction and profit. In *Proceedings of the Fifth SIAM International Conferenceon Data Mining (SDM2005)*, pages 12–19, 2005.

[9] D. Carmel, N. Zwerdling, I. Guy, S. Ofek-Koifman, N. Har'el, I. Ronen, E. Uziel, S. Yogev, and S. Chernov. Personalized social search based on the user's social network. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, pages 1227–1236, New York, NY, USA, 2009. ACM.

[10] S. Chakrabarti. Dynamic personalized PageRank in entity-relation graphs. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 571–580, New York, NY, USA, 2007. ACM.

[11] H. Cheng, P.-N. Tan, J. Sticklen, and W. F. Punch. Recommendation via query centered random walk on k-partite graph. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 457–462, Washington, DC, USA, 2007. IEEE Computer Society.

[12] T. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006.

[13] J. W. Demmel. *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), 1997.

[14] K. C. Foster, S. Q. Muth, J. J. Potterat, and R. B. Rothenberg. A faster katz status score algorithm. *Comput. & Math. Organ. Theo.*, 7(4):275–285, 2001.

[15] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with applicaiton to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, March 2007.

[16] G. H. Golub and C. F. V. Loan. *Matrix Computations*. Third Edition, Johns Hopkins Univ. Press, Baltimore, MD, 1996.

[17] G. H. Golub and G. Meurant. Matrices, moments and quadrature. In *Numerical analysis 1993 (Dundee, 1993)*, volume 303 of *Pitman Res. Notes Math. Ser.*, pages 105–156. Longman Sci. Tech., Harlow, 1994.

[18] M. Gori and A. Pucci. ItemRank: a random-walk based scoring algorithm for recommender engines. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2766–2771, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

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

[20] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Information retrieval in folksonomies: Search and ranking. In Y. Sure and J. Domingue, editors, *Proceedings of the 3rd European Semantic Web Conference*, volume 4011 of *Lecture Notes in Computer Science*, pages 411–426, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[21] Z. Huang, X. Li, and H. Chen. Link prediction approach to collaborative filtering. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 141–142, New York, NY, USA, 2005. ACM.

[22] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proc. of the 8th ACM Intl. Conf. on Know. Discov. and Data Mining (KDD'02)*.

[23] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on the World Wide Web*, pages 271–279, Budapest, Hungary, 2003. ACM.

[24] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18:39–43, 1953.

[25] P. Li, H. Liu, J. X. Yu, J. He, and X. Du. Fast single-pair simrank computation. In *Proc. of the SIAM Intl. Conf. on Data Mining (SDM2010)*, Columbus, OH.

[26] D. Liben-Nowell and J. M. Kleinberg. The link prediction problem for social networks. In *Proc. of the ACM Intl. Conf. on Inform. and Knowlg. Manage. (CIKM'03)*.

[27] F. McSherry. A uniform approach to accelerated PageRank computation. In *Proceedings of the 14th international conference on the World Wide Web*, pages 575–582, New York, NY, USA, 2005. ACM Press.

[28] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, November 1999.

[29] S.-T. Park and D. M. Pennock. Applying collaborative filtering techniques to movie search for better ranking and browsing. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 550–559, New York, NY, USA, 2007. ACM.

[30] H. Qiu and E. R. Hancock. Commute times for graph spectral clustering. In *Proc. of the 11th Intl. Conf. on Comp. Anal. of Images and Patterns (CAIP'05)*.

[31] H. Qiu and E. R. Hancock. Clustering and embedding using commute times. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1873–1890, 2007.

[32] M. J. Rattigan and D. Jensen. The case for anomalous link discovery. *SIGKDD Explor. Newsl.*, 7(2):41–47, 2005.

[33] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In D. K. Harman, editor, *Proceedings of the Third Text REtrieval Conference*, TREC, NIST Special Publication 500-226, pages 109–126, Gaithersburg, MD, November 1994. National Institute of Standards and Technology.

[34] M. Saerens, F. Fouss, L. Yen, and P. Dupont. The principal components analysis of a graph, and its relationships to spectral clustering. In *Proc. of the 15th Euro. Conf. on Mach. Learn. (ECML'04)*.

[35] P. Sarkar and A. W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *Proceedings of the 23rd conference on Uncertainty in Artificial Intelligence (UAI2007)*, 2007.

[36] P. Sarkar, A. W. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *Proc. of the 25th Intl. Conf. on Mach. Learn. (ICML'08)*.

[37] R. Schenkel, T. Crecelius, M. Kacimi, S. Michel, T. Neumann, J. X. Parreira, and G. Weikum. Efficient top-k querying over social-tagging networks. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 523–530, New York, NY, USA, 2008. ACM.

[38] R. Schenkel, T. Crecelius, M. Kacimi, T. Neumann, J. Xavier Parreira, M. Spaniol, and G. Weikum. Social wisdom for search and recommendation. *IEEE Data Engineering Bulletin*, 31(2):40–49, June 2008.

[39] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proc. of the 40th Ann. ACM Symp. on Theo. of Comput. (STOC'08)*, pages 563–568.

[40] R. Varga. *Matrix Iterative Analysis*. Prentice-Hall, 1962.

[41] L. Yen, F. Fouss, C. Decaestecker, P. Francq, and M. Saerens. Graph nodes clustering based on the commute-time kernel. In *Proc. of the 11th Pacific-Asia Conf. on Knowled. Disc. and Data Mining (PAKDD 2007)*. Lecture Notes in Computer Science (LNCS), 2007.

[42] Y. Zhang, J. Q. Wu, and Y. T. Zhuang. Random walk models for top-n recommendation task. *Journal of Zhejiang University - Science A*, 10(7):927–936, July 2009.

[43] D. Zhou, J. Bian, S. Zheng, H. Zha, and C. L. Giles. Exploring social annotations for information retrieval. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 715–724, New York, NY, USA, 2008. ACM.