Internet Mathematics Vol. 8, Nos. 1–2: 73–112

Fast Matrix Computations for Pairwise and Columnwise Commute Times and Katz Scores

Francesco Bonchi, Pooya Esfandiar, David F. Gleich, Chen Greif, and Laks V. S. Lakshmanan

Abstract. We explore methods for approximating the commute time and Katz score between a pair of nodes. These methods are based on the approach of matrices, moments, and quadrature developed in the numerical linear algebra community. They rely on the Lanczos process and provide upper and lower bounds on an estimate of the pairwise scores. We also explore methods to approximate the commute times and Katz scores from a node to all other nodes in the graph. Here, our approach for the commute times is based on a variation of the conjugate gradient algorithm, and it provides an estimate of all the diagonals of the inverse of a matrix. Our technique for the Katz scores is based on exploiting an empirical localization property of the Katz matrix. We adapt algorithms used for personalized PageRank computing to these Katz scores and theoretically show that this approach is convergent. We evaluate these methods on 17 real-world graphs ranging in size from 1000 to 1,000,000 nodes. Our results show that our pairwise commute-time method and columnwise Katz algorithm both have attractive theoretical properties and empirical performance.

ISSN: 1542-7951 print

1. Introduction

Commute times [Göbel and Jagers 74] and Katz scores [Katz 53] are two topological measures defined between any pair of vertices in a graph that capture their relationship due to the link structure. Both of these measures have become important because of their use in social network analysis as well as applications such as link prediction [Liben-Nowell and Kleinberg 03], anomalous link detection [Rattigan and Jensen 05], recommendation [Sarkar and Moore 07], and clustering [Saerens et al. 04].

For example, in [Liben-Nowell and Kleinberg 03], the authors identify a variety of topological measures as features for link prediction: the problem of predicting the likelihood of users/entities forming new connections in the future, given the current state of the network. The measures they studied fall into two categories—neighborhood-based measures and path-based measures. The former are cheaper to compute, yet the latter are more effective at link prediction. Katz scores were among the most effective path-based measures studied in [Liben-Nowell and Kleinberg 03], and the commute time also performed well.

Standard algorithms to compute these measures between all pairs of nodes are often based on direct solution methods and require cubic time and quadratic space in the number of nodes of the graph. Such algorithms are impractical for large-scale networks, which may have at least a million vertices and several million edges. We explore algorithms to compute a targeted subset of scores that do scale to modern networks.

Katz scores measure the affinity between nodes via a weighted sum of the number of paths between them. Formally, the Katz score between node i and j is

$$K_{i,j} = \sum_{\ell=1}^{\infty} \alpha^{\ell} \operatorname{paths}_{\ell}(x, y),$$

where $\operatorname{paths}_{\ell}(x, y)$ denotes the number of paths of length ℓ between i and j and $\alpha < 1$ is an attenuation parameter. Now let $\boldsymbol{A}$ be the symmetric adjacency matrix, corresponding to an undirected and connected graph, and recall that $(\boldsymbol{A}^{\ell})_{i,j}$ is the number of paths between nodes i and j. Then computing the Katz scores for all pairs of nodes is equivalent to the following computation:

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

Hereinafter, we refer to $\boldsymbol{K}$ as the *Katz matrix*. We shall study this problem only in the case that $\boldsymbol{I} - \alpha \boldsymbol{A}$ is positive definite. This occurs when $\alpha < 1/\sigma_{\max}(\boldsymbol{A})$, where $\sigma_{\max}(\boldsymbol{A})$ is the largest singular value of $\boldsymbol{A}$, and also corresponds to the case that the series expansion converges.

In order to define the *commute time* between nodes, we must first define the *hitting time* between nodes. Formally, the hitting time from node i to j is the expected number of steps for a random walk started at i to visit j for the first time. The commute time is the round-trip time between nodes and is defined as the sum of hitting times from i to j and from j to i. The hitting time is computed via first-transition analysis on the random walk transition matrix associated with a graph. To be precise, let $\boldsymbol{A}$ again be the symmetric adjacency matrix. Let $\boldsymbol{D}$ be the diagonal matrix of degrees:

$$D_{i,j} = \begin{cases} \sum_v A_{i,v} & i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The random walk transition matrix is given by $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}$. Let $H_{i,j}$ be the hitting time from node i to node j. Based on the Markovian nature of a random walk, $H_{i,j}$ must satisfy

$$H_{i,j} = 1 + \sum_v H_{i,v} P_{v,j} \quad \text{and} \quad H_{i,i} = 0.$$

That is, the hitting time between i and j is 1 more than the hitting time between i and v, weighted by the probability of transitioning between v and j, for all v. The minimum nonnegative solution $\boldsymbol{H}$ that satisfies this equation is thus the matrix of hitting times. The commute time between nodes i and j is then

$$C_{i,j} = H_{i,j} + H_{j,i}.$$

As a matrix, $\boldsymbol{C} = \boldsymbol{H} + \boldsymbol{H}^T$, and we refer to $\boldsymbol{C}$ as the *commute-time matrix*. An equivalent expression follows from exploiting a few relationships with the combinatorial graph Laplacian matrix: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ [Fouss et al. 07]. Each element $C_{i,j}$ is given by

$$C_{i,j} = \mathrm{Vol}(G)\left(L^\dagger_{i,i} - 2L^\dagger_{i,j} + L^\dagger_{j,j}\right),$$

where $\mathrm{Vol}(G)$ is the sum of elements in $\boldsymbol{A}$ and $\boldsymbol{L}^\dagger$ is the pseudoinverse of $\boldsymbol{L}$. The null space of the combinatorial graph Laplacian has a well-known expression in terms of the connected components of the graph G. This relationship allows us to write

$$\boldsymbol{L}^\dagger = \Big(\underbrace{\boldsymbol{L} + \frac{1}{n}\mathbf{e}\mathbf{e}^T}_{\tilde{\boldsymbol{L}}}\Big)^{-1} - \frac{1}{n}\mathbf{e}\mathbf{e}^T$$

for *connected* graphs [Saerens et al. 04], where $\mathbf{e}$ is the vector of all 1's, and n is the number of nodes in the graph. The commute time between nodes in different connected components is infinite, and thus we need to consider only connected

$\boldsymbol{A}$	the symmetric adjacency matrix for a connected undirected graph
$\boldsymbol{D}$	the diagonal matrix of node degrees
n	the number of vertices in $\boldsymbol{A}$
$\mathbf{e}$	the vector of all ones
$\mathbf{e}_i$	a vector of zeros with a 1 in the ith position
$\boldsymbol{L}$	the combinatorial Laplacian matrix of a graph, $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$
$\tilde{\boldsymbol{L}}$	the adjusted combinatorial Laplacian, $\tilde{\boldsymbol{L}} = \boldsymbol{L} + \frac{1}{n}\mathbf{e}\mathbf{e}^T$
α	the damping parameter in the Katz score
$\boldsymbol{K}$	the Katz matrix, $\boldsymbol{K} = (\boldsymbol{I} - \alpha\boldsymbol{A})^{-1}$
$\boldsymbol{C}$	the commute-time matrix
$\boldsymbol{Z}$	a "general" matrix, usually $\boldsymbol{I} - \alpha\boldsymbol{A}$ or $\tilde{\boldsymbol{L}}$

Table 1. Notation.

graphs. We summarize the notation thus far and a few subsequent definitions in Table 1.

Computing either Katz scores or commute times between all pairs of nodes involves the inverse of a matrix:

$$(\boldsymbol{I} - \alpha\boldsymbol{A})^{-1} \quad \text{or} \quad \left(\boldsymbol{L} + \frac{1}{n}\mathbf{e}\mathbf{e}^T\right)^{-1}.$$

Standard algorithms for a matrix inverse require $O(n^3)$ time and $O(n^2)$ memory. Both of these requirements are inappropriate for a large network (see Section 2 for a brief survey of existing alternatives).

Inspired by applications in anomalous link detection and recommendation, we focus on computing only a single Katz score or commute time and on approximating a column of these matrices. In the former case, our goal is to find the score for a given pair of nodes, and in the latter, it is to identify the most closely related nodes for a given node. In our vision, the pairwise algorithms should help in cases in which random pairwise data are queried, for instance when checking random network connections, or evaluating user similarity scores as a user explores a website. For the columnwise algorithms, recommending the most-similar nodes to a query node or predicting the most likely links to a given query node are both obvious applications.

One way to compute a single score—what we term the *pairwise problem*—is to find the value of a bilinear form

$$\mathbf{u}^T \boldsymbol{Z}^{-1}\mathbf{v},$$

where $\boldsymbol{Z} = (\boldsymbol{I} - \alpha\boldsymbol{A})$ or $\boldsymbol{Z} = \tilde{\boldsymbol{L}}$. An interesting approach to estimating these bilinear forms and to deriving computable upper and lower bounds on the

value arises from the relationship between the Lanczos/Stieltjes procedure and a quadrature rule [Golub and Meurant 94]. This relationship and the resulting algorithm for a quadratic form ($\mathbf{u}^T \boldsymbol{Z}^{-1}\mathbf{u}$) are described in Section 4.1. Prior to that, and because it will form the basis of a few algorithms that we use, Section 3 first reviews the properties of the Lanczos method. We state the pairwise procedure for commute times and Katz scores in Sections 4.2 and 4.3.

The *columnwise problem* is to compute, or approximate, a column of the matrix $\boldsymbol{C}$ or $\boldsymbol{K}$. A column of the commute-time matrix is

$$\mathbf{c}_i = \boldsymbol{C}\mathbf{e}_i = \mathrm{vol}(G)[(\mathbf{e}_i - \mathbf{e}_v)^T \tilde{\boldsymbol{L}}^{-1}(\mathbf{e}_i - \mathbf{e}_v) : 1 \leq v \leq n].$$

A difficulty with this computation is that it requires *all* of the diagonal elements of $\tilde{\boldsymbol{L}}^{-1}$, as well as the solution of the linear system $\tilde{\boldsymbol{L}}^{-1}\mathbf{e}_i$. We can use a property of the Lanczos procedure and its relationship with the conjugate gradient algorithm to solve $\tilde{\boldsymbol{L}}^{-1}\mathbf{e}_i$ *and* estimate all of the diagonals of the inverse simultaneously [Paige and Saunders 75, Chantas et al. 08].

A column of the Katz matrix is $\boldsymbol{K}\mathbf{e}_i$, which corresponds to solving a single linear system:

$$\mathbf{k}_i = \boldsymbol{K}\mathbf{e}_i = (\boldsymbol{I} - \alpha\boldsymbol{A})^{-1}\mathbf{e}_i - \mathbf{e}_i.$$

Empirically, we observe that the solutions of the Katz linear system are often localized. That is, there are only a few large elements in the solution vector, and many negligible elements. See Table 2 for an example of this localization in a few graphs. In order to capitalize on this phenomenon, we use a generalization of "push"-style algorithms for personalized PageRank computing [McSherry 05, Andersen et al. 06, Berkhin 07]. These methods access the adjacency information for only a limited number of vertices in the graph. In Section 5.2, we explain the generalization of these methods and the adaptation to Katz scores, and we utilize the theory of coordinate descent optimization algorithms to establish convergence. As we argue in that section, these techniques might also be called "Gauss–Southwell" methods, based on historical precedents.

One of the advantages of Lanczos-based algorithms is that the convergence is often much faster than a worst-case analysis would suggest. This means that studying their convergence by empirical means and on real data sets is important. We do so for 17 real-world networks in Section 6, ranging in size from approximately 1000 vertices to 1,000,000 vertices. These experiments highlight both the strengths and weaknesses of our approaches, and should provide a balanced picture of our algorithms. In particular, our algorithms run in seconds or milliseconds—significantly faster than many techniques that use preprocessing to estimate all of the scores simultaneously, which can take minutes.

Straightforward approaches based on the conjugate gradient technique are often competitive with our techniques. However, our algorithms have other desirable properties, such as upper and lower bounds on the solution or exploiting sparsity *in the solution vector*, which conjugate gradient does not. These experiments also shed light on a recent result from [von Luxburg et al. 10] on the relationship between commute time and the degree distribution.

Literature directly related to the problems we study and the techniques we propose is discussed throughout the paper, in context. However, we have isolated a small set of core related papers and discuss them in the next section.

2. Related Work

This paper is about algorithms for computing commute times and Katz scores over networks with hundreds of thousands to millions of nodes. Most existing techniques determine the scores among all pairs of nodes simultaneously [Acar et al. 09, Wang et al. 07, Sarkar and Moore 07] (discussed below). These methods tend to involve some preprocessing of the graph using a one-time, rather expensive, computation. We instead focus on quick estimates of these measures between a single pair of nodes and between a single node and all other nodes in the graph. In this vein, a recent paper [Li et al. 10] studies efficient computation of SimRank [Jeh and Widom 02] for a given pair of nodes.

A highly related paper is [Benzi and Boito 10], in which the authors investigate entries in functions of the adjacency matrix, such as the exponential, using quadrature-based bounds. A priori upper and lower bounds are obtained by employing a few Lanczos steps, and the bounds are effectively used to observe the exponential decay behavior of the exponential of an adjacency matrix.

In [Sarkar and Moore 07], an interesting and efficient approach is proposed for finding approximate nearest neighbors with respect to a truncated version of the commute-time measure. In [Spielman and Srivastava 08], the authors develop a technique for computing the effective resistance of all edges (which is proportional to commute time) in $O(m \log n)$ time. Both of these procedures involve some preprocessing.

Standard techniques to approximate Katz scores include truncating the series expansion to paths of length less than $\ell_{\max}$ [Foster et al. 01, Wang et al. 07] and low-rank approximation [Liben-Nowell and Kleinberg 03, Acar et al. 09]. Only the former technique, when specialized to compute only a pair or top-k set, has performance comparable to our algorithms. However, when we tested an adapted algorithm based on the Neumann series expansion, it required much more work than the techniques we propose.

As mentioned in the introduction, both commute times and Katz scores were studied in [Liben-Nowell and Kleinberg 03] for the task of link prediction, and were found to be effective. Beyond link prediction, in [Yen et al. 07], 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 intracluster inertia. Intuitively, this inertia measures how close nodes within a cluster are to each other. The algorithm we propose for computing the Katz and commute-time scores for a given pair of nodes x, y extends to the case that one wants to find the aggregate score between a node x and a set of nodes S. Consequently, this work has applications for finding the distance between a point and a cluster as well as for finding intracluster inertia. For applications to recommender systems, the authors of [Sarkar et al. 08] used their truncated commute-time measure for link prediction over a collaboration graph and showed that it outperforms personalized PageRank [Page et al. 99].

3. The Lanczos Process

The Lanczos algorithm [Lanczos 50] is a procedure applied to a symmetric matrix that works particularly well when the given matrix is large and sparse. A sequence of Lanczos iterations can be thought of as "truncated" orthogonal similarity transformations. Given an $n \times n$ matrix $\boldsymbol{Z}$, we construct a matrix $\boldsymbol{Q}$ with orthonormal columns, one at a time, and perform only a small number of steps, say k, where $k \ll n$. The input for the algorithm is the matrix $\boldsymbol{Z}$, an initial vector $\mathbf{q}$, and a number of steps k. Upon exit, we have an $n \times (k+1)$ matrix $\boldsymbol{Q}_{k+1}$ with orthonormal columns and a $(k+1) \times k$ tridiagonal matrix $\boldsymbol{T}_{k+1,k}$ that satisfy the relationship

$$\boldsymbol{Z}\,\boldsymbol{Q}_k = \boldsymbol{Q}_{k+1}\,\boldsymbol{T}_{k+1,k},$$

where $\boldsymbol{Q}_k$ is the $n \times k$ matrix that contains the first k columns of $\boldsymbol{Q}_{k+1}$, and $\boldsymbol{T}_{k+1,k} = \mathrm{tri}(\beta_i, \alpha_i, \beta_i)$.

What makes the Lanczos procedure attractive is the good approximation properties that it has for $k \ll n$. The matrix $\boldsymbol{T}_{k+1,k}$ is small when $k \ll n$, but the eigenvalues of its $k \times k$ upper part—a matrix we will refer to as $\boldsymbol{T}_k$ in the subsequent section—approximate the extremal eigenvalues of the large $n \times n$ matrix $\boldsymbol{Z}$. This can be exploited not only for eigenvalue computations but also for solving a linear system [Lanczos 53, Paige and Saunders 75]. Another attractive feature is that the matrix $\boldsymbol{Z}$ does not necessarily have to be provided explicitly; the algorithm uses $\boldsymbol{Z}$ only via matrix–vector products.

The Lanczos procedure is given in Algorithm 1. For expositional purposes we define the core of the algorithm as Algorithm 2. We will later incorporate that part into other algorithms; see Section 4.

4. Pairwise Algorithms

Consider the commute time and Katz score between a single pair of nodes:

$$C_{i,j} = \mathrm{Vol}(G)(\mathbf{e}_i - \mathbf{e}_j)^T \boldsymbol{L}^\dagger(\mathbf{e}_i - \mathbf{e}_j),$$
$$K_{i,j} = \mathbf{e}_i^T (\boldsymbol{I} - \alpha \boldsymbol{A})^{-1}\mathbf{e}_j - \delta_{i,j}.$$

In these expressions, $\mathbf{e}_i$ and $\mathbf{e}_j$ are vectors of zeros with a 1 in the ith and jth positions, respectively, and $\delta_{i,j}$ is the Kronecker delta function. A straightforward means of computing them is to solve the linear systems

$$\tilde{\boldsymbol{L}}^{-1}\mathbf{y} = \mathbf{e}_i - \mathbf{e}_j \quad \text{and} \quad (\boldsymbol{I} - \alpha \boldsymbol{A})\mathbf{x} = \mathbf{e}_j.$$

Then $C_{i,j} = \mathrm{Vol}(G)(\mathbf{e}_i - \mathbf{e}_j)^T\mathbf{y}$ and $K_{i,j} = \mathbf{e}_i^T\mathbf{x} - \delta_{i,j}$. It is possible to compute the pairwise scores by solving these linear systems. In what follows, we show how a technique combining the Lanczos iteration and a quadrature rule [Golub and Meurant 94, Golub and Meurant 97] produces the pairwise commute-time score or the pairwise Katz score *as well as* upper and lower bounds on the estimate.

Algorithm 1 Lanczos$(\boldsymbol{Z}, \mathbf{q}, k)$.

$\mathbf{q}_1 = \mathbf{q}/\|\mathbf{q}\|_2, \beta_0 = 0, \mathbf{q}_0 = 0$
for $j = 1$ to k **do**
  $\mathbf{z} = \boldsymbol{Z}\mathbf{q}_j$
  $\alpha_j = \mathbf{q}_j^T\mathbf{z}$
  $\mathbf{z} = \mathbf{z} - \alpha_j\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1}$
  $\beta_j = \|\mathbf{z}\|_2$
  if $\beta_j = 0, \mathbf{q}_{j+1} = 0$ **and quit**
  else $\mathbf{q}_{j+1} = \mathbf{z}/\beta_j$

Algorithm 2 LanczosStep$(\boldsymbol{Z}, \mathbf{q}^{(-)}, \mathbf{q}, \beta^{(-)})$.

$\mathbf{z} = \boldsymbol{Z}\mathbf{q}$
$\alpha = \mathbf{q}^T\mathbf{z}$
$\mathbf{z} = \mathbf{z} - \alpha\mathbf{q} - \beta^{(-)}\mathbf{q}^{(-)}$
$\beta = \|\mathbf{z}\|_2$
if $\beta = 0, \mathbf{q}^{(+)} = 0$
else $\mathbf{q}^{(+)} = \mathbf{z}/\beta$
Return $(\mathbf{q}^{(+)}, \alpha, \beta)$

4.1. Matrices, Moments, and Quadrature

Both of the pairwise computations above are instances of the general problem of estimating a bilinear form

$$\mathbf{u}^T f(\boldsymbol{Z})\mathbf{v},$$

where $\boldsymbol{Z}$ is symmetric positive definite (for Katz, this occurs by restricting the value of α, and for commute times, the adjusted Laplacian $\tilde{\boldsymbol{L}}$ is always positive definite), and $f(x)$ is an analytic function on the region containing the eigenvalues of $\boldsymbol{Z}$. The only function $f(x)$ we use in this paper is $f(x) = 1/x$, although we treat the problem more generally for part of this section.

In [Golub and Meurant 94, Golub and Meurant 97], the authors introduced elegant computational techniques for evaluating such bilinear forms. They provided a solid mathematical framework and a rich collection of possible applications. These techniques are well known in the numerical linear algebra community, but they do not seem to have been used in data-mining problems. We adapt this methodology to the pairwise score problem, and explain how to do so in an efficient manner in a large-scale setting. The algorithm has two main components: Gauss-type quadrature rules for evaluating definite integrals and the Lanczos algorithm for partial reduction to symmetric tridiagonal form. In the following discussion, we treat the case $\mathbf{u} = \mathbf{v}$. This form suffices, thanks to the identity

$$\mathbf{u}^T f(\boldsymbol{Z})\mathbf{v} = \frac{1}{4}\left[(\mathbf{u}+\mathbf{v})^T f(\boldsymbol{Z})(\mathbf{u}+\mathbf{v}) - (\mathbf{u}-\mathbf{v})^T f(\boldsymbol{Z})(\mathbf{u}-\mathbf{v})\right].$$

Because $\boldsymbol{Z}$ is symmetric positive definite, it has a unitary spectral decomposition $\boldsymbol{Z} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, where $\boldsymbol{Q}$ is an orthogonal matrix whose columns are eigenvectors of $\boldsymbol{Z}$ with unit 2-norms, and $\boldsymbol{\Lambda}$ is a diagonal matrix with the eigenvalues of $\boldsymbol{Z}$ along its diagonal. We use this decomposition only for the derivation that follows; it is never computed in our algorithm. Given this decomposition, for any analytic function f,

$$\mathbf{u}^T f(\boldsymbol{Z})\mathbf{u} = \mathbf{u}^T\, \boldsymbol{Q} f(\boldsymbol{\Lambda})\, \boldsymbol{Q}^T \mathbf{u} = \sum_{i=1}^{n} f(\lambda_i)\tilde{u}_i^2,$$

where $\tilde{\mathbf{u}} = \boldsymbol{Q}^T\mathbf{u}$. Let $\underline{\lambda}$ and $\overline{\lambda}$ be values that are respectively lower and higher than the extremal eigenvalues of $\boldsymbol{Z}$. The last sum is equivalent to the Stieltjes integral

$$\mathbf{u}^T f(\boldsymbol{Z})\mathbf{u} = \int_{\underline{\lambda}}^{\overline{\lambda}} f(\lambda)\, d\omega(\lambda). \tag{4.1}$$

Here $\omega(\lambda)$ is a piecewise constant measure, which is monotonically increasing, and its values depend directly on the eigenvalues of $\boldsymbol{Z}$. Let

$$0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

be the eigenvalues of $\boldsymbol{Z}$. Note that $\underline{\lambda} < \lambda_1$ and $\overline{\lambda} > \lambda_n$. Now $\omega(\lambda)$ takes the following form:

$$\omega(\lambda) = \begin{cases} 0, & \lambda < \lambda_1, \\ \sum_{j=1}^{i} \tilde{u}_j^2, & \lambda_i \leq \lambda < \lambda_{i+1}, \\ \sum_{j=1}^{n} \tilde{u}_j^2, & \lambda_n \leq \lambda. \end{cases}$$

The first of Golub and Meurant's key insights is that we can compute an approximation for an integral of the form (4.1) using a quadrature rule

$$\int_{\underline{\lambda}}^{\overline{\lambda}} f(\lambda)\, d\omega(\lambda) \approx \sum_{j=1}^{N} f(\eta_j)\omega_j,$$

where η_j, ω_j are the nodes and weights of a Gaussian quadrature rule. The second insight is that the Lanczos procedure *constructs the quadrature rule itself*. Since we use a quadrature rule, an estimate of the error is readily available; see, for example, [Davis and Rabinowitz 84]. More importantly, we can use variants of the Gaussian quadrature 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 provides effective stopping criteria for the algorithm—we shall see this in the experiments in Section 6.2. It is important to note that such componentwise bounds could not be easily obtained if we were to extract the value of the element from a column of the inverse, by solving the corresponding linear system, for example. Indeed, typically for the solution of a linear system, normwise bounds are available, but obtaining bounds pertaining to the components of the solution is significantly more challenging, and results of this sort are harder to establish. It should also be noted that bounds of the sort discussed here cannot be obtained for general nonsymmetric matrices.

Returning to the procedure, let $f(\lambda)$ be a function whose $(2k+1)$st derivative has a negative sign for all $\underline{\lambda} < \lambda < \overline{\lambda}$. Note that $f(\lambda) = 1/\lambda$ satisfies this condition because all odd derivatives are negative when $\lambda > 0$. As a high-level algorithm, the Golub–Meurant procedure for estimating bounds

$$\underline{b} \leq \mathbf{u}^T f(\boldsymbol{Z})\mathbf{u} \leq \overline{b}$$

is given by the following steps:

1. Let $\sigma = \|\mathbf{u}\|_2$.

Algorithm 3 MMQStep [Golub and Meurant 97, Algorithm GQL].

Input: $\alpha, \beta_{-1}, \beta, b_{-1}, c_{-1}, d_{-1}, \underline{d}_{-1}, \overline{d}_{-1}$

1: $b = b_{-1} + \frac{\beta_{-1}^2 c_{-1}^2}{d_{-1}(\alpha d_{-1} - \beta_{-1}^2)}$; $\quad c = c_{-1}\frac{\beta_{-1}}{d_{-1}}$; $\quad d = \alpha - \frac{\beta_{-1}^2}{d_{-1}}$

2: $\overline{d} = \alpha - \underline{\lambda} - \frac{\beta_{-1}^2}{\overline{d}_{-1}}$; $\quad \underline{d} = \alpha - \overline{\lambda} - \frac{\beta_{-1}^2}{\underline{d}_{-1}}$

3: $\overline{\omega} = \underline{\lambda} + \frac{\beta^2}{\overline{d}}$; $\quad \underline{\omega} = \overline{\lambda} + \frac{\beta^2}{\underline{d}}$

4: $\overline{b} = b + \frac{\beta^2 c^2}{d(\overline{\omega} d - \beta^2)}$; $\quad \underline{b} = b + \frac{\beta^2 c^2}{d(\underline{\omega} d - \beta^2)}$

Output: $(\overline{b}, \underline{b})$ and $(b, c, d, \overline{d}, \underline{d})$

2. Compute $\boldsymbol{T}_k$ from k steps of the Lanczos procedure applied to $\boldsymbol{Z}$ and $\mathbf{u}/\sigma$.

3. Compute $\underline{\boldsymbol{T}}_k$, which is the matrix $\boldsymbol{T}_k$ extended with another row and column crafted to add the eigenvalue $\overline{\lambda}$ to the eigenvalues of $\boldsymbol{T}_k$. This new matrix is still tridiagonal.

4. Set $\underline{b} = \sigma^2 \mathbf{e}_1^T f(\underline{\boldsymbol{T}}_k)\mathbf{e}_1$. This estimate corresponds to a $(k+1)$-point Gauss–Radau rule with a prescribed point of $\overline{\lambda}$.

5. Compute $\overline{\boldsymbol{T}}_k$, which is the matrix $\boldsymbol{T}_k$ extended with another row and column crafted to add the eigenvalue $\underline{\lambda}$ to the eigenvalues of $\boldsymbol{T}_k$. Again, this new matrix is still tridiagonal.

6. Set $\overline{b} = \sigma^2 \mathbf{e}_1^T f(\overline{\boldsymbol{T}}_k)\mathbf{e}_1$. This estimate corresponds to a $(k+1)$-point Gauss–Radau rule with a prescribed point of $\underline{\lambda}$.

Based on the theory of Gauss–Radau quadrature, the fact that these are lower and upper bounds on the quadratic form $\mathbf{u}^T f(\boldsymbol{Z})\mathbf{u}$ follows because the *sign* of the error term changes when a node is prescribed in this fashion. See [Golub and Meurant 10, Theorem 6.4] for more information. As k increases, the upper and lower bounds converge.

While this form of the algorithm is convenient for understanding the high-level properties and structure of the procedure, it is not computationally efficient. If $f(\lambda) = 1/\lambda$ and if we want to compute a more accurate estimate by increasing k, then we need to solve two inverse eigenvalue problems (steps 3 and 5), and solve two linear systems (steps 4 and 6). Each of these steps involves $O(k)$ work because the matrices involved are tridiagonal. However, a *constant-time* update procedure is possible. The set of operations to efficiently update $\underline{b}$ and $\overline{b}$ after a Lanczos step (Algorithm 2) is given by Algorithm 3. Please see [Golub and Meurant 97] for an explanation of this procedure. Using Algorithms 2 and 3 as

Algorithm 4 Pairwise score bounds for commute time.

Input: $\boldsymbol{L}$ (Laplacian matrix); i, j (pairwise coordinate); $\underline{\lambda}, \overline{\lambda}$ (bounds where $\underline{\lambda} < \lambda(\boldsymbol{L}) < \overline{\lambda}$); τ (stopping tolerance)

Output: $\underline{\kappa}, \overline{\kappa}$ where $\underline{\kappa} < (\mathbf{e}_i - \mathbf{e}_j)^T \boldsymbol{L}^\dagger (\mathbf{e}_i - \mathbf{e}_j) < \overline{\kappa}$

(Initialize Lanczos) $\sigma = \sqrt{2}, \mathbf{q}_{-1} = 0, \mathbf{q}_0 = (\mathbf{e}_i - \mathbf{e}_j)/\sigma, \beta_0 = 0$

(Initialize MMQStep) $b_0 = 0, c_0 = 1, d_0 = 1, \overline{d}_0 = 1, \underline{d}_0 = 1$

for $j = 1, \ldots$ **do**

  Set $(\mathbf{q}_j, \alpha_j, \beta_j)$ from LanczosStep$(\tilde{\boldsymbol{L}}, \mathbf{q}_{j-2}, \mathbf{q}_{j-1}, \beta_{j-1})$

  Set $(\overline{b}, \underline{b})$ and $(b_j, c_j, d_j, \overline{d}_j, \underline{d}_j)$ from MMQStep$(\alpha_j, \beta_{j-1}, \beta_j, b_{j-1}, c_{j-1}, d_{j-1}, \underline{d}_{j-1}, \overline{d}_{j-1})$.

  $\underline{\kappa} = \sigma^2 \underline{b}$; $\overline{\kappa} = \sigma^2 \overline{b}$

  if $\overline{\kappa} - \underline{\kappa} < \tau$, **stop**

subroutines, it is now straightforward to state the pairwise commute-time and Katz procedures.

4.2. Pairwise Commute Scores

The bilinear form that we need for estimating a commute time is

$$b = (\mathbf{e}_i - \mathbf{e}_j)^T \tilde{\boldsymbol{L}}^{-1} (\mathbf{e}_i - \mathbf{e}_j).$$

For this problem, we apply Algorithm 2 to step through the Lanczos process and then use Algorithm 3 to update the upper and lower bounds on the score. This combination is explicitly described in Algorithm 4. Note that we do not need to apply the final correction with $\frac{1}{n}\mathbf{e}\mathbf{e}^T$ because $\mathbf{e}^T(\mathbf{e}_i - \mathbf{e}_j) = 0$.

4.3. Pairwise Katz Scores

The bilinear form that we need to estimate for a Katz score is

$$b = \mathbf{e}_i^T (\boldsymbol{I} - \alpha \boldsymbol{A})^{-1} \mathbf{e}_j.$$

Recall that we use the identity

$$b = \frac{1}{4}\Big[\underbrace{(\mathbf{e}_i + \mathbf{e}_j)^T (\boldsymbol{I} - \alpha \boldsymbol{A})^{-1} (\mathbf{e}_i + \mathbf{e}_j)}_{=g} - \underbrace{(\mathbf{e}_i - \mathbf{e}_j)^T (\boldsymbol{I} - \alpha \boldsymbol{A})^{-1} (\mathbf{e}_i - \mathbf{e}_j)}_{=h}\Big].$$

In this case, we apply the combination of LanczosStep and MMQStep to estimate $\underline{g} \le g \le \overline{g}$ and $\underline{h} \le h \le \overline{h}$. Then $\frac{1}{4}(\underline{g} - \overline{h}) \le b \le \frac{1}{4}(\overline{g} - \underline{h})$. Algorithm 5 describes the entire procedure.

Algorithm 5 Pairwise score bounds for Katz.

Input: $\boldsymbol{A}$ (adjacency matrix); α (the Katz damping factor); i, j (pairwise coordinate); $\underline{\lambda}, \overline{\lambda}$ (bounds where $\underline{\lambda} < \lambda(\boldsymbol{I} - \alpha\boldsymbol{A}) < \overline{\lambda}$); τ (stopping tolerance)
Output: $\underline{\rho}, \overline{\rho}$ where $\underline{\rho} < (\boldsymbol{I} - \alpha\boldsymbol{A})^{-1}_{i,j} < \overline{\rho}$

(Initialize Lanczos for g) $\sigma = \sqrt{2}, \mathbf{q}_{-1} = 0, \mathbf{q}_0 = (\mathbf{e}_i + \mathbf{e}_j)/\sigma, \beta_0^g = 0$
(Initialize Lanczos for h) $\mathbf{u}_{-1} = 0, \mathbf{u}_0 = (\mathbf{e}_i - \mathbf{e}_j)/\sigma, \beta_0^h = 0$
(Initialize MMQStep for g) $b_0^g = 0, c_0^g = 1, d_0^g = 1, \overline{d}_0^g = 1, \underline{d}_0^g = 1$
(Initialize MMQStep for h) $b_0^h = 0, c_0^h = 1, d_0^h = 1, \overline{d}_0^h = 1, \underline{d}_0^h = 1$
for $j = 1, \ldots$ **do**
  Set $(\mathbf{q}_j, \alpha_j^g, \beta_j^g)$ from LanczosStep$((\boldsymbol{I} - \alpha\boldsymbol{A}), \mathbf{q}_{j-2}, \mathbf{q}_{j-1}, \beta_{j-1}^f)$
  Set $(\mathbf{u}_j, \alpha_j^h, \beta_j^h)$ from LanczosStep$((\boldsymbol{I} - \alpha\boldsymbol{A}), \mathbf{u}_{j-2}, \mathbf{u}_{j-1}, \beta_{j-1}^h)$
  Set $(\overline{g}, \underline{g})$ and $(b_j^g, c_j^g, d_j^g, \overline{d}_j^g, \underline{d}_j^g)$ from MMQStep$(\alpha_j^g, \beta_{j-1}^g, \beta_j^g, b_{j-1}^g, c_{j-1}^g, d_{j-1}^g, \underline{d}_{j-1}^g, \overline{d}_{j-1}^g)$.
  Set $(\overline{h}, \underline{h})$ and $(b_j^h, c_j^h, d_j^h, \overline{d}_j^h, \underline{d}_j^h)$ from MMQStep$(\alpha_j^h, \beta_{j-1}^h, \beta_j^h, b_{j-1}^h, c_{j-1}^h, d_{j-1}^h, \underline{d}_{j-1}^h, \overline{d}_{j-1}^h)$.
  $\underline{\rho} = \sigma^2/4(\underline{g} - \overline{h})$; $\overline{\rho} = \sigma^2/4(\overline{g} - \underline{h})$
  if $\overline{\rho} - \underline{\rho} < \tau$, **stop**

5. Columnwise Algorithms

Whereas the last section used a single procedure to derive two algorithms, in this section, we investigate two different procedures: one for commute time and a different procedure for Katz scores. The reason behind this difference is that, as mentioned in the introduction, computing a column of the commute-time matrix cannot be stated as the solution of a single linear system

$$\mathbf{c}_i = \boldsymbol{C}\mathbf{e}_i = \mathrm{vol}(G)\left[(\mathbf{e}_i - \mathbf{e}_v)^T \tilde{\boldsymbol{L}}^{-1}(\mathbf{e}_i - \mathbf{e}_v) : 1 \le v \le n\right].$$

Computing this column requires *all* of the diagonal elements of the inverse. In contrast, a column of the Katz matrix is just the solution of a linear system

$$\mathbf{k}_i = \boldsymbol{K}\mathbf{e}_i = (\boldsymbol{I} - \alpha\boldsymbol{A})^{-1}\mathbf{e}_i - \mathbf{e}_i.$$

For this computation, we exploit an empirical localization property of these columns.

5.1. Columnwise Commute Times

A straightforward way to compute an entire column of the commute-time matrix would require solving n separate linear systems: one to get both $\tilde{\boldsymbol{L}}^{-1}\mathbf{e}_i$ and $\tilde{\boldsymbol{L}}^{-1}_{i,i}$, and the other $n-1$ to get $\tilde{\boldsymbol{L}}^{-1}_{j,j}$ for $i \neq j$. Neither solving each system independently nor using a *multiple right-hand side* algorithm [O'Leary 80] will easily

yield an efficient procedure. Both of these approaches generate far too much extraneous information. In fact, the only information we need are the solution of a linear system, as well as the diagonal elements of the pseudoinverse. Thus, any procedure to compute or estimate $\mathrm{diag}(\boldsymbol{L}^{\dagger})$ provides a practical algorithm.

One such procedure arises again from the Lanczos method. It was originally described in [Paige and Saunders 75], and is explained in more detail in [Chantas et al. 08]. Suppose we want to compute $\mathrm{diag}(\tilde{\boldsymbol{L}}^{-1})$. If the Lanczos algorithm runs to completion in exact arithmetic, then we have

$$\tilde{\boldsymbol{L}} = \boldsymbol{Q}\,\boldsymbol{T}\,\boldsymbol{Q}^T \quad \text{and} \quad \tilde{\boldsymbol{L}}^{-1} = \boldsymbol{Q}\,\boldsymbol{T}^{-1}\,\boldsymbol{Q}^T.$$

Let $\boldsymbol{T} = \boldsymbol{R}\boldsymbol{R}^T$ be a Cholesky factorization of $\boldsymbol{T}$. If we substitute this factorization into the expression for the inverse, then $\tilde{\boldsymbol{L}}^{-1} = \boldsymbol{V}\,\boldsymbol{R}^{-T}\,\boldsymbol{R}^{-1}\,\boldsymbol{V}^T$. Now let $\boldsymbol{W} = \boldsymbol{V}\,\boldsymbol{R}^{-T}$. Note that $\tilde{\boldsymbol{L}}^{-1} = \boldsymbol{W}\,\boldsymbol{W}^T$. As a notational convenience, let $\mathbf{w}_k$ be the kth column of $\boldsymbol{W}$. Consequently,

$$\mathrm{diag}(\boldsymbol{E}^{-1}) = \sum_{k=1}^{n} \mathbf{w}_k \circ \mathbf{w}_k,$$

where $\mathbf{w}_k \circ \mathbf{w}_k$ is the Hadamard (elementwise) product $[\mathbf{w}_k \circ \mathbf{w}_k]_i = \mathbf{w}_{k,i}^2$. If we implement CG based on the Lanczos algorithm as explained in [Paige and Saunders 75], then the vector $\mathbf{w}_k$ is computed as part of the standard algorithm and is available at no additional cost. This idea is implemented in the `cgLanczos.m` code [Saunders 07], which we use in our experiments. Please see [Chantas et al. 08] for a detailed account of this derivation including the diagonal estimate.

Based on advice from the author of the `cgLanczos` code, we added local reorthogonalization to the Lanczos procedure. This addition requires a few extra vectors of memory, but ensures greater orthogonality in the computed Lanczos vectors $\mathbf{q}_k$. Also, based on advice from the author, we use the following preconditioned linear system:

$$\boldsymbol{D}^{-1/2}\,\tilde{\boldsymbol{L}}\boldsymbol{D}^{-1/2}\mathbf{y} = \boldsymbol{D}^{-1/2}\mathbf{e}_i.$$

If $\mathbf{f}$ is the estimate of the diagonals of $(\boldsymbol{D}^{-1/2}\,\tilde{\boldsymbol{L}}\boldsymbol{D}^{-1/2})^{-1}$, then $\boldsymbol{D}^{-1}\mathbf{f}$ is the estimate of the diagonals of $\tilde{\boldsymbol{L}}^{-1}$. Using this preconditioned formulation, the algorithm converges much more quickly than without preconditioning. In summary, this approach to estimate the columnwise commute times $\mathbf{c}_i$ is as follows:

1. Solve $\boldsymbol{D}^{-1/2}\,\tilde{\boldsymbol{L}}\boldsymbol{D}^{-1/2}\mathbf{y} = \boldsymbol{D}^{-1/2}\mathbf{e}_i$ using `cgLanczos.m` to get both $\mathbf{y}$ and $\mathbf{f} \approx \mathrm{diag}\left((\boldsymbol{D}^{-1/2}\,\tilde{\boldsymbol{L}}\boldsymbol{D}^{-1/2})^{-1}\right)$.
2. Set $\mathbf{x} = \boldsymbol{D}^{-1/2}\mathbf{y} - \frac{1}{n}\mathbf{e} \approx \boldsymbol{L}^{\dagger}\mathbf{e}_i$.

3. Set $\mathbf{g} = \boldsymbol{D}^{-1}\mathbf{f} - \frac{1}{n}\mathbf{e} \approx \mathrm{diag}(\boldsymbol{L}^{\dagger})$.

4. Output $\mathbf{c}_i \approx \mathbf{g} + x_i\mathbf{e} - 2\mathbf{x}$.

We refrain from stating this as a formal algorithm because the majority of the work is in the `cgLanczos.m` routine.

5.2. Columnwise Katz Scores

In this section, we show how to adapt techniques for rapid personalized PageRank computation [McSherry 05, Andersen et al. 06, Berkhin 07] to the problem of computing a column of the Katz matrix. Recall that such a column is given by the solution of a single linear system:

$$\mathbf{k}_i = \boldsymbol{K}\mathbf{e}_i = (\boldsymbol{I} - \alpha\boldsymbol{A})^{-1}\mathbf{e}_i - \mathbf{e}_i.$$

The algorithms for personalized PageRank 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 access the adjacency information of only a small set of vertices and need not explore the majority of the graph. Such a property is useful when the solution of a linear system is localized on a small set of elements.

Localization is a term with a number of interpretations. Here, we use it to mean that the vector becomes sparse after small elements are rounded to 0. A nice way of measuring this property is to look at the *participation ratios* [Farkas et al 01]. Let $\mathbf{k}$ be a column of the Katz matrix. Then the participation ratio of $\mathbf{k}$ is

$$p = \frac{\left(\sum_j k_j^2\right)^2}{\sum_j k_j^4}.$$

This ratio measures the number of effective nonzeros of the vector. If $\mathbf{k}$ is a uniform vector, then $p = n$, the size of the vector. If $\mathbf{k}$ has only a single element, then $p = 1$, the number of states occupied. For a series of graphs we describe more formally in Section 6.1, we show the statistics of some participation ratios in Table 2. We pick columns of the matrix in two ways: (i) randomly and (ii) from the degree distribution to ensure we choose both high-, medium-, and low-degree vertices. See Section 6.7 for a more formal description about how we pick columns; we use the "hard alpha" value of Katz described in the experiments section. The results show that number of effective nonzeros is always less than 10,000, even when the graph has 1,000,000 vertices. Usually, it is even smaller. Our forthcoming algorithms exploit this property.

The basis of these personalized PageRank algorithms is a variant on the Richardson stationary method for solving a linear system [Varga 62]. Given a

Graph	Vertices	Avg. Deg.	Participation Ratios			
			Min	Mean	Median	Max
tapir	1024	5.6	4.2	12.0	11.8	35.8
stanford-cs-sym	2759	7.4	1.0	26.3	23.5	274.1
ca-GrQc	4158	6.5	1.0	27.4	34.0	84.2
wiki-Vote	7066	28.5	1.2	248.8	291.6	342.6
ca-HepTh	8638	5.7	1.0	23.5	29.8	82.1
ca-HepPh	11204	21.0	1.0	160.7	256.1	268.5
Stanford3	11586	98.1	1.1	1509.5	1657.8	1706.4
ca-AstroPh	17903	22.0	1.0	167.5	219.2	290.8
ca-CondMat	21363	8.5	1.0	71.0	85.6	204.6
email-Enron	33696	10.7	1.0	203.0	262.5	598.6
soc-Epinions1	75877	10.7	1.0	299.2	455.6	526.0
soc-Slashdot0811	77360	12.1	1.0	320.4	453.3	495.8
arxiv	86376	12.0	1.0	121.1	137.9	508.6
dblp	93156	3.8	1.0	50.0	25.2	258.9
email-EuAll	224832	3.0	1.0	237.7	276.7	7743.7
flickr2	513969	12.4	1.0	592.3	1104.9	1414.9
hollywood-2009	1069126	105.3	2.0	1696.0	2433.8	3796.0

Table 2. Participation ratios for Katz scores. These results demonstrate that the columns of the Katz matrix are highly localized. In the worst case, there are only a few thousand large elements in a vector, compared with the graph size of a few hundred thousand vertices.

linear system $\boldsymbol{Z}\mathbf{x} = \mathbf{b}$, the Richardson iteration is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{r}^{(k)},$$

where $\mathbf{r}^{(k)} = \mathbf{b} - \boldsymbol{Z}\mathbf{x}^{(k)}$ is the residual vector at the kth iteration. While updating $\mathbf{x}^{(k+1)}$ 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 [McSherry 05, Andersen et al. 06, Berkhin 07] propose the following change: do not update $\mathbf{x}^{(k+1)}$ with the entire residual, and instead change only a single component of $\mathbf{x}$. Formally, $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + r_j^{(k)}\mathbf{e}_j$, where $\mathbf{e}_j$ 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 $\boldsymbol{Z}$:

$$\mathbf{r}^{(k+1)} = \mathbf{b} - \boldsymbol{Z}\mathbf{x}^{(k+1)} = \mathbf{b} - \boldsymbol{Z}(\mathbf{x}^{(k)} + r_j^{(k)}\mathbf{e}_j) = \mathbf{r}^{(k)} + r_j^{(k)}\boldsymbol{Z}\mathbf{e}_j.$$

Algorithm 6 Columnwise Katz scores (via the Gauss–Southwell algorithm).

Input: $\boldsymbol{A}$ (the adjacency matrix), α (the Katz damping factor), i (the desired column), τ (a stopping tolerance).
Output: $\mathbf{x}$ (an approximate solution of $(\boldsymbol{I} - \alpha \boldsymbol{A})^{-1}\mathbf{e}_i$)

Set $\mathbf{x} = 0$, $\mathbf{r} = 0$
Let $\mathcal{H}$ be a heap over the non-zero entries of $\mathbf{r}$ larger than τ.
Set $r_i = 1$, update $\mathcal{H}$
while $\mathcal{H}$ is not empty **do**
  Set j as the index of the largest element in $\mathcal{H}$
  if $r_j < \tau$ **then** quit.
  $\eta = r_j$
  $x_j \leftarrow x_j + \eta$
  $r_j \leftarrow 0$, remove j from $\mathcal{H}$
  for u where $A_{j,u} > 0$ **do**
    $r_u \leftarrow r_u + \alpha\eta$
    if $r_u > \tau$ **then** insert j in $\mathcal{H}$ or update $\mathcal{H}$.
$x_i \leftarrow x_i - 1$

Suppose that $\mathbf{r}$, $\mathbf{x}$, and $\boldsymbol{Z}\mathbf{e}_j$ are sparse. Then this update introduces only a small number of new nonzeros into both $\mathbf{x}$ and the new residual $\mathbf{r}$. If $\boldsymbol{Z} = (\boldsymbol{I} - \alpha\boldsymbol{A})$, as in the case of Katz, then each column is sparse, and thus keeping the solution and residual sparse is a natural choice for graph algorithms in which the solution $\mathbf{x}$ is localized (i.e., many components of $\mathbf{x}$ 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 $\mathbf{x}$ while exploring only a small subset of the graph. The resulting procedure is presented in Algorithm 6. For reasons that will become clear below, we call this procedure the *Gauss–Southwell* algorithm. While experimenting with this method, we found that sorting the heap by $\boldsymbol{D}^{-1}\mathbf{r}$ instead of $\mathbf{r}$ yielded convergence with fewer total edges explored, mirroring the results in [Andersen et al. 06]. We use this version in all of our experiments, although we state all the formal convergence results for the simple choice of residual $\mathbf{r}$.

Let $d_{\max}$ be the maximum degree of a node in the graph. Then each iteration takes $O(d_{\max} \log n)$ time. We analyze the convergence of this algorithm for Katz scores in two stages. In the first case, when $\alpha < 1/d_{\max}$, the convergence theory of this method for personalized PageRank also shows that it converges for Katz scores. This fortunate occurrence results from the equivalence of Katz scores and the general formulation of PageRank adopted by [McSherry 05] in this setting. In the second case, when $\alpha < 1/\sigma_{\max}(\boldsymbol{A})$, then $(\boldsymbol{I} - \alpha\boldsymbol{A})$ is still symmetric positive definite, and the Richardson algorithm converges. To show convergence in this

case, we will utilize an equivalence between this algorithm and a coordinate descent method.

For completeness, we prove a precise convergence result when $\alpha < 1/d_{\max}$. The key observations here is that the residual $\mathbf{r}$ is always nonnegative and that the sum of the residual ($\mathbf{e}^T\mathbf{r}$) is monotonically decreasing. To show convergence, we need to bound this sum by a function that converges to 0.

Consider the algorithm applied to $(\boldsymbol{I} - \alpha\boldsymbol{A})\mathbf{x} = \mathbf{e}_i$. From step k to step $k+1$, the algorithm sets

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \eta\mathbf{e}_j, \quad \mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} + \eta(\boldsymbol{I} - \alpha\boldsymbol{A})\mathbf{e}_j.$$

First note that $\alpha < 1/d_{\max}$ implies $r_i^{(k+1)} \geq 0$ given $r_i^{(k)} \geq 0$. This bound now implies that $\mathbf{x}_i^{(k+1)} \geq 0$ when $\mathbf{x}_i^{(k)} \geq 0$. Since these conditions hold for the initial conditions $\mathbf{x}^{(0)} = 0$ and $\mathbf{r}^{(0)} = \mathbf{e}_q$, they remain true throughout the iteration. Consequently, we can use the *sum* of $\mathbf{r}^{(k)}$ as the 1-norm of this vector; that is, $\mathbf{e}^T\mathbf{r}^{(k+1)} = \|\mathbf{r}^{(k+1)}\|_1$. It is now straightforward to analyze the convergence of this sum:

$$\mathbf{e}^T\mathbf{r}^{(k+1)} = \mathbf{e}^T\mathbf{r}^{(k)} - \eta + \alpha\eta\mathbf{e}^T\boldsymbol{A}\mathbf{e}_i.$$

At this point, we need the bound that $\eta = r_j^{(k)} \geq (1/n)\mathbf{e}^T\mathbf{r}^{(k)}$, which follows immediately from the fact that $r_j^{(k)}$ is the largest element in $\mathbf{r}^{(k)}$. Also, $\mathbf{e}^T\boldsymbol{A}\mathbf{e}_i \leq d_{\max}$. Thus, we draw the following conclusion:

Remark 5.1. If $\alpha < 1/d_{\max}$, then the 1-norm of the residual in the Gauss–Southwell iteration applied to the Katz linear system satisfies

$$\left\|\mathbf{r}^{(k+1)}\right\|_1 \leq \left(1 - \frac{1 - \alpha d_{\max}}{n}\right)\left\|\mathbf{r}^{(k)}\right\|_1 \leq \left(1 - \frac{1 - \alpha d_{\max}}{n}\right)^k.$$

In the second case, when $1/d_{\max} < \alpha < 1/\sigma_{\max}(\boldsymbol{A})$, then the Gauss–Southwell iteration in Algorithm 6 still converges. However, the result is more intricate than in the previous case because the sum of the residual does not converge monotonically. As we shall see, treating this linear system as an optimization problem provides a way to handle this case.

Let $\boldsymbol{Z}$ be symmetric positive definite. We first show that the Gauss–Southwell algorithm is a coordinate descent method for the convex problem

$$\text{minimize } \tfrac{1}{2}\mathbf{x}^T\boldsymbol{Z}\mathbf{x} - \mathbf{x}^T\mathbf{b} = f(\mathbf{x}).$$

The gradient of this problem is $\boldsymbol{Z}\mathbf{x} - \mathbf{b}$; hence a stationary point is the solution of the linear system and the global minimizer. In this framework, the Richardson method is a gradient descent method. If $\mathbf{g}^{(k)}$ is the gradient at step k, $\mathbf{g}^{(k)} =$

$\boldsymbol{Z}\mathbf{x}^{(k)} - \mathbf{b}$, then

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{g}$$

is exactly the Richardson step.

Now consider a standard coordinate descent method. Such methods usually minimize the function in the jth coordinate *exactly*. Formally, they find

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma^{(k)}\mathbf{e}_j,$$

where

$$\gamma^{(k)} = \text{argmin}_\gamma\, f(\mathbf{x}^{(k)} + \gamma\mathbf{e}_j).$$

Solving this system produces the choice

$$\gamma^{(k)} = \frac{b_j - (\boldsymbol{Z}\mathbf{e}_j)^T\mathbf{x}^{(k)}}{Z_{j,j}}.$$

Note that in terms of the optimization problem, the Gauss–Southwell algorithm generates

$$\gamma_S^{(k)} = r_j^{(k)} = (b_j - \mathbf{z}_j^T\mathbf{x}^{(k)}).$$

The two methods are equivalent if the diagonals of $\boldsymbol{A}$ are 1. Consequently, we have the following result.

Lemma 5.2. *The Gauss–Southwell method for $\boldsymbol{Z}\mathbf{x} = \mathbf{b}$ with $Z_{i,i} = 1$ is equivalent to a coordinate gradient descent method for the function $f(\mathbf{x}) = (1/2)\mathbf{x}^T\boldsymbol{Z}\mathbf{x} - \mathbf{x}^T\mathbf{b}$.*

To produce a convergent algorithm, we must now specify how to choose the descent direction j.

Theorem 5.3. *Let $\boldsymbol{Z}$ be symmetric positive definite with $Z_{i,i} = 1$. Then the Gauss–Southwell method for $\boldsymbol{Z}\mathbf{x} = \mathbf{b}$ and $j^{(k)} = \text{argmax}_i \left|r_i^{(k)}\right|$ or with $j^{(k)}$ chosen cyclically ($j^{(1)} = 1, j^{(k+1)} = j^{(k)} + 1 \mod n$) is convergent.*

Proof. This result follows from the convergence of the coordinate descent method [Luo and Tseng 92, Theorem 2.1] with these two update rules. The first is also known as the Gauss–Southwell rule. □

This proof demonstrates that as long as $A_{i,i} = 0$ for all the diagonal entries of the adjacency matrix, then Algorithm 6 will converge when $(\boldsymbol{I} - \alpha\boldsymbol{A})$ is positive definite, that is, when $\alpha < 1/\sigma_{\max}(\boldsymbol{A})$. We term this algorithm a Gauss–Southwell procedure because the choice of j in the algorithm is given by the Gauss–Southwell rule.

6. Experimental Results

The previous sections showed three algorithms based on the Lanczos method, and showed the theoretical convergence of the columnwise Katz algorithm. In this section, we investigate these algorithms numerically. Algorithms based on the Lanczos method, in general, are arguably best studied empirically because their worst-case convergence properties are often conservative. These experiments are designed to shed light on two key questions:

1. How do these iterative algorithms converge to the exact solution?
2. Are the techniques faster than a conjugate gradient-based algorithm?

Note that columnwise commute-time measure is a special case for reasons we discuss below, and we investigate the accuracy of our procedure only for that problem.

Experimental settings. We implemented our methods in MATLAB and MATLAB mex codes. All computations and timings were done in Linux on a desktop computer with a Core i7-960 processor (4 core, 2.8 GHz) with 24 GB of memory. As mentioned in the introduction, all of the experimental code is available from our website.

We first describe the data used in the experiments. These data were also used in the experiment about localization in the Katz scores from the previous section.

6.1. Data

We use three publicly available sources and three graphs we collected ourselves. The majority of the data comes from the SNAP collection [Leskovec 10], of which we use ca-GrQc, ca-HepTh, ca-CondMat, ca-AstroPh, email-Enron, email-EuAll [Leskovec et al. 07], wiki-Vote [Leskovec et al. 10], soc-Epinions1 [Richardson et al. 03], and soc-Slashdot0811 [Leskovec et al. 09]. Besides these, the graph tapir is from [Gilbert and Teng 02], the graph Stanford3 is from [Traud et al. 11], and both graphs stanford-cs [Hirai et al. 00] and hollywood-2009 [Boldi et al. 11] are distributed via the webgraph framework [Boldi and Vigna 04]. The graph stanford-cs is actually a subset of the webbase-2001 graph [Hirai et al. 00], restricted to pages in the domain cs.stanford.edu. All graphs are symmetrized (if nonsymmetric) and stripped of any self-loops, edge weights, and extraneous connected components.

DBLP. We extracted the DBLP coauthors graph from a recent snapshot (2005–2008) of the DBLP database. 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. This data set contains another coauthorship graph extracted by a snapshot (1990–2000) of arXiv, 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 worked only with that subset.

Flickr contacts. Flickr is a popular online community for sharing photos, with millions of users. The node set represents users, and the directed edges are between contacts. 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 subgraph 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. (This network is now available from the University of Florida sparse matrix collection [Davis and Hu 10]).

Table 3 presents some elementary statistics about these graphs. We also include the time to compute the truncated singular value decomposition for the first 200 singular values and vectors using the ARPACK library [Lehoucq et al. 97] in MATLAB's `svds` routine. This time reflects the work it would take to use the standard low-rank preprocessing algorithm for Katz scores on the network [Liben-Nowell and Kleinberg 03].

6.2. Pairwise Commute Scores

From this data, we now study the performance of our algorithm for pairwise commute scores, and compare it against solving the linear system $\tilde{\boldsymbol{L}}\mathbf{x} = (\mathbf{e}_i - \mathbf{e}_j)$ using the conjugate gradient method (CG). At each step of CG, we use the approximation $(\mathbf{e}_i - \mathbf{e}_j)^T \mathbf{x}^{(k)}$, where $\mathbf{x}^{(k)}$ is the kth iterate. The convergence check in CG was that either the pairwise element value changed by less than the tolerance, checked by taking a relative difference between steps, or the 2-norm of the residual fell below the tolerance.

Graph	Nodes	Edges	Avg. Deg.	Max Deg.	$\|\boldsymbol{A}\|_2$	SVD (sec.)
tapir	1024	2846	5.56	24	6.9078	2.2
stanford-cs	2759	10270	7.44	303	39.8213	8.9
ca-GrQc	4158	13422	6.46	81	45.6166	16.2
ca-HepTh	8638	24806	5.74	65	31.0348	31.5
ca-CondMat	21363	91286	8.55	279	37.8897	78.6
wiki-Vote	7066	100736	28.51	1065	138.1502	28.5
ca-HepPh	11204	117619	21.00	491	244.9349	49.5
dblp	93156	178145	3.82	260	33.6180	391.0
email-Enron	33696	180811	10.73	1383	118.4177	119.5
ca-AstroPh	17903	196972	22.00	504	94.4296	62.3
email-EuAll	224832	339925	3.02	7636	102.5365	935.3
soc-Epinions1	75877	405739	10.69	3044	184.1751	324.6
soc-Slashdot0811	77360	469180	12.13	2539	131.3418	359.1
arxiv	86376	517563	11.98	1253	99.3319	241.2
Stanford3	11586	568309	98.10	1172	212.4606	48.8
flickr2	513969	3190452	12.41	4369	663.3587	3418.7
hollywood-2009	1069126	56306653	105.33	11467	2246.5596	5998.9

Table 3. The networks studied in the experiments. The first five columns are self-explanatory. The last two columns show the largest singular value of the network, which is also the matrix 2-norm, and the time taken to compute the largest 200 singular values and vectors.

The first figure we present shows the result of running Algorithm 4 on a single pairwise commute-time problem for a few graphs (Figure 1). The upper row of figures shows the actual bounds themselves. The bottom row of figures shows the relative error that would result from using the bounds as an approximate solution. We show the same results for CG. The exact solution was computed using MINRES [Paige and Saunders 75] to solve the same system as CG to a tolerance of 10^{-10}. For all of the graphs, we used $\underline{\lambda} = 10^{-4}$ and $\overline{\lambda} = \|\tilde{\boldsymbol{L}}\|_1$. Again using ARPACK, we verified that the smallest eigenvalue of each of the Laplacian matrices was larger than $\underline{\lambda}$. We chose the vertices for the pair from among the high-degree vertices for no particular reason. Both Algorithm 4 and CG used a tolerance of 10^{-4}.

In the figure, the upper bounds and lower bounds "trap" the solution from above and below. These bounds converge smoothly to the final solution. For

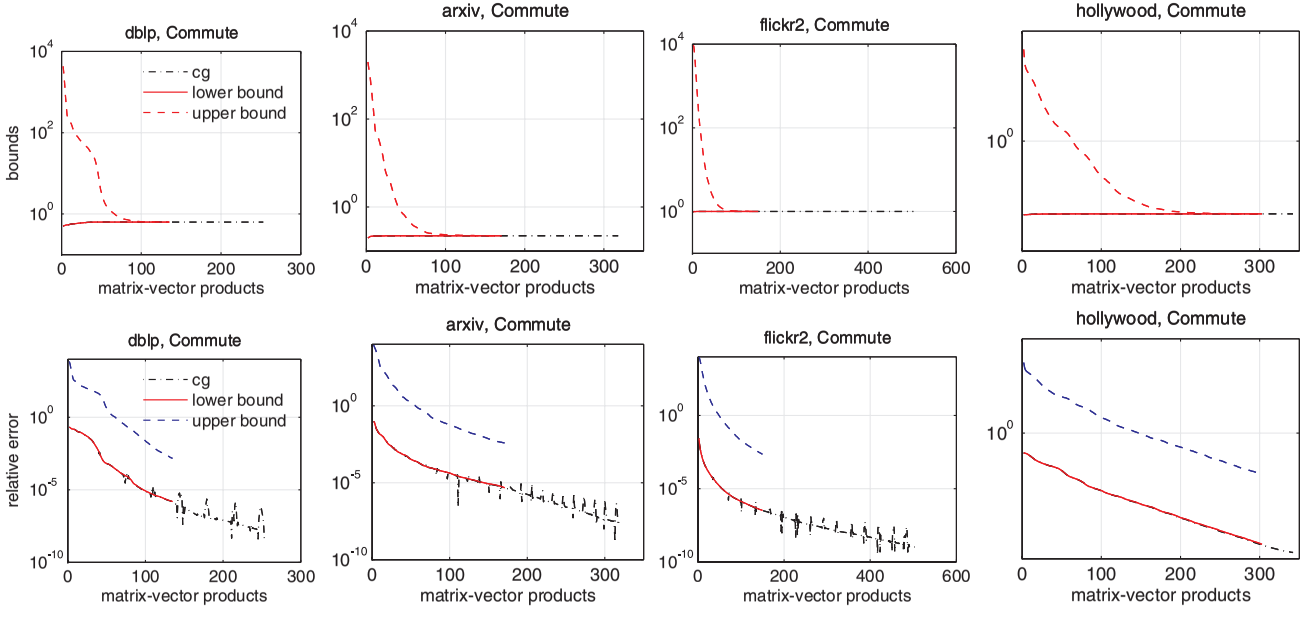

Figure 1. Convergence results for pairwise commute times. Top row: Each figure shows the upper and lower bounds at each iteration of Algorithm 4 for the graphs dblp, arxiv, flickr2, and hollywood-2009. Bottom row: For the same graphs, each figure shows the relative size of the error, $|v_{\mathrm{alg}} - v_{\mathrm{exact}}|/|v_{\mathrm{exact}}|$, in the upper and lower bounds at each iteration. In both cases, we also show the same data from the conjugate gradient algorithm. See Section 6.2 for our discussion (color figure available online).

these experiments, the lower bound has smaller error, and also, this error tracks the performance of CG quite closely. This behavior is expected in cases in which the largest eigenvalue of the matrix is well separated from the remaining eigenvalues—a fact that holds for the Laplacians of our graphs; see [Mihail and Papadimitriou 02] and [Chung et al. 03] for some theoretical justification. 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 4 that the estimate of $\overline{\lambda}$ 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 the score, it is not possible to terminate it until the residual is small. Thus, the conjugate gradient method requires more iterations than our pairwise algorithm. Note, however, that 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 error and it is not guaranteed to provide smooth convergence for an element of the solution. These oscillations make early termination of the CG algorithm problematic, whereas no such issues occur for the upper and lower bounds from our algorithm. We speculate that the seemingly smooth convergence behavior that we observe for the upper and lower bound estimates may be rooted in the convergence behavior of the largest Ritz value of the tridiagonal matrix associated with Lanczos, but a better understanding of this issue will require further exploration.

6.3. Pairwise Katz Scores

We next show the same type of figure but for the pairwise Katz scores instead; see Figure 2. We use a value of α that makes $\boldsymbol{I} - \alpha \boldsymbol{A}$ nearly indefinite. Such a value produces the slowest convergence in our experience. The particular value we use is $\alpha = 1/(\|\boldsymbol{A}\|_2 + 1)$, which we call "hard alpha" in some of the figure titles. For all of the graphs, we again used $\underline{\lambda} = 10^{-4}$ and $\overline{\lambda} = \|\tilde{\boldsymbol{L}}\|_1$. This value of $\underline{\lambda}$ is smaller than the smallest eigenvalue of $\boldsymbol{I} - \alpha \boldsymbol{A}$ for all the graphs. Also, the vertex pairs are the same as those used for Figure 1.

For pairwise Katz scores, the baseline approach involves solving the linear system $(\boldsymbol{I} - \alpha \boldsymbol{A})\mathbf{x} = \mathbf{e}_j$, again using the conjugate gradient method (CG). At each step of CG, we use the approximation $\mathbf{e}_i^T \mathbf{x}^{(k)}$, where $\mathbf{x}^{(k)}$ is the kth iterate. We use the same convergence check as in the CG baseline for commute time.

For these problems, we also evaluated techniques based on the Neumann series for $\boldsymbol{I} - \alpha \boldsymbol{A}$, but those took over 100 times as many iterations as CG or our pairwise approach. The Neumann series is the same algorithm used in [Wang et al. 07] but customized for the linear system, not matrix inverse, which is a more appropriate comparison for the pairwise case. Finally, the exact solution was again computed using MINRES [Paige and Saunders 75] to solve the same system as CG to a tolerance of 10^{-14}.

A distinct difference from the commute-time results is that both the lower and upper bounds converge similarly and have similar errors. This occurs because of the symmetry in the upper and lower bounds that results from using the MMQ algorithm twice on the form

$$\frac{1}{4}\left[(\mathbf{e}_i + \mathbf{e}_j)^T (\boldsymbol{I} - \alpha \boldsymbol{A})^{-1} (\mathbf{e}_i + \mathbf{e}_j) - (\mathbf{e}_i - \mathbf{e}_j)^T (\boldsymbol{I} - \alpha \boldsymbol{A})^{-1} (\mathbf{e}_i - \mathbf{e}_j)\right].$$

In comparison with the conjugate gradient method, our pairwise algorithm is slower to converge. While the conjugate gradient method appears to outperform our pairwise algorithms here, recall that it does not provide any approximation guarantees. Also, the two matrix–vector products in Algorithm 5 can easily be merged into a single "combined" matrix–vector product algorithm. As we discuss further in the conclusion, such an implementation would reduce the difference in running time between the two methods.

6.4. Relative Matrix–Vector Products in Pairwise Algorithms

Thus far, we have detailed a few experiments describing how the pairwise algorithms converge. In these cases, we compared against the conjugate gradient algorithm for a single pair of vertices on each graph. In this experiment, we examine the number of matrix–vector products that each algorithm requires for a much larger set of vertex pairs. Let us first describe how we picked the vertices for the pairwise comparison. There were two types of vertex pairs chosen: purely random, and degree-correlated. The purely random choices are simple: pick a random permutation of the vertex numbers; then use pairs of vertices from this ordering. The degree-correlated pairs were picked by first sorting the vertices by degree in decreasing order, then picking the 1st, 2nd, 3rd, 4th, 5th, 10th, 20th, 30th, 40th, 50th, 100th ... vertices from this ordering, and finally, using all vertex pairs in this subset. Note that for commute time, we used only the 1st, 5th, 10th, 50th, 100th ... vertices to reduce the total computation time. For the pairwise commute times, we used 20 random pairs and used 100 random pairs for pairwise Katz scores.

In Figure 3, we show the matrix–vector performance ratio between our pairwise algorithms and conjugate gradient. Let k_{cg} be the number of matrix–vector

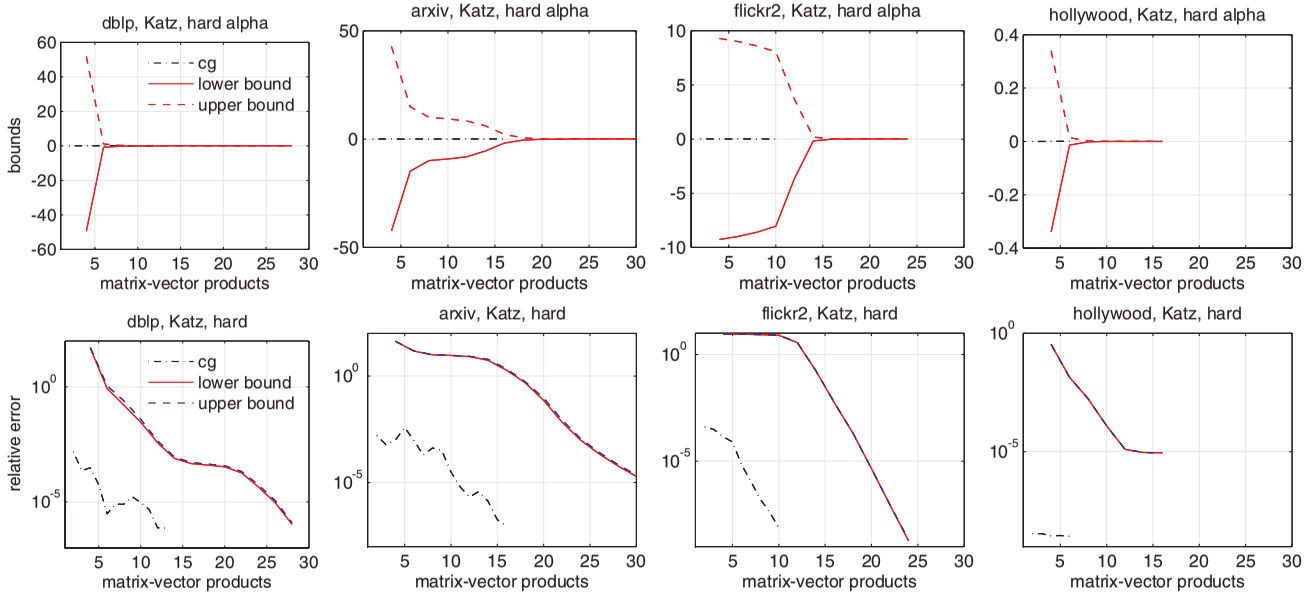

Figure 2. Convergence results for pairwise Katz scores. Top row: Each figure shows the upper and lower bounds at each iteration of Algorithm 5 for the graphs dblp, arxiv, flickr2, and hollywood-2009. Bottom row: For the same graphs, each figure shows the relative size of the error, $(v_{\mathrm{alg}} - v_{\mathrm{exact}})/v_{\mathrm{exact}}$, in the upper and lower bounds at each iteration. In both cases, we also show the same data from the conjugate gradient algorithm. See Section 6.3 for discussion (color figure available online).

products until CG converges to a tolerance of 10^{-4} (as in previous experiments), and let k_{alg} be the number of matrix–vector products until our algorithm converges. The performance ratio is

$$\frac{k_{\text{cg}} - k_{\text{alg}}}{k_{\text{cg}}},$$

which has a value of 0 when the two algorithms take the same number of matrix–vector products, the value 1 when our algorithm takes zero matrix–vector products, and the value -1 (or -2) when our algorithm takes two (or three) times as many matrix–vector products as CG. We display the results as a box plot of the results from all trials. There was no systematic difference in the results between the two types of vertex pairs (random or degree correlated).

These results show that the small sample in the previous section is fairly representative of the overall performance difference. In general, our commute-time algorithm uses fewer matrix–vector products than conjugate gradient. We suspect that this result is due to the ability to stop early as explained in Section 6.2. And as also observed in Section 6.3, our pairwise Katz algorithm tends to take two to three times as many matrix–vector products as conjugate gradient. These results again used the same "hard alpha" value.

6.5. Columnwise Commute Times

Our next set of results concerns the precision of our approximation to the columnwise commute-time scores. Because the output of our columnwise commute-time algorithm is based on a coarse approximation of the diagonal elements of the inverse, we do not expect these scores to converge to their exact values as we increase the work in the algorithm. Consequently, we study the results in terms of the *precision at k* measure. Recall that the motivation for studying these columnwise measures is not to get the column scores precisely correct, but rather to identify the closest nodes to a given query or target node. That is, we are most interested in the smallest elements of a column of the commute-time matrix. Given a target node i, let S_k^{alg} be the k closest nodes to i in terms of our algorithm. Also, let S_k^* be the k closest nodes to i in terms of the exact commute time. (See below for how we compute this set.) The precision at k measure is

$$|S_k^* \cap S_k^{\text{alg}}|/k.$$

In words, this formula computes the fraction of the true set of k nodes that our algorithm identifies.

We ran the algorithm from Section 5.1 with a tolerance of 10^{-16} to evaluate the maximum accuracy possible with this approach. We choose two sets of 50 target nodes. The first set was picked uniformly at random, the second set was picked

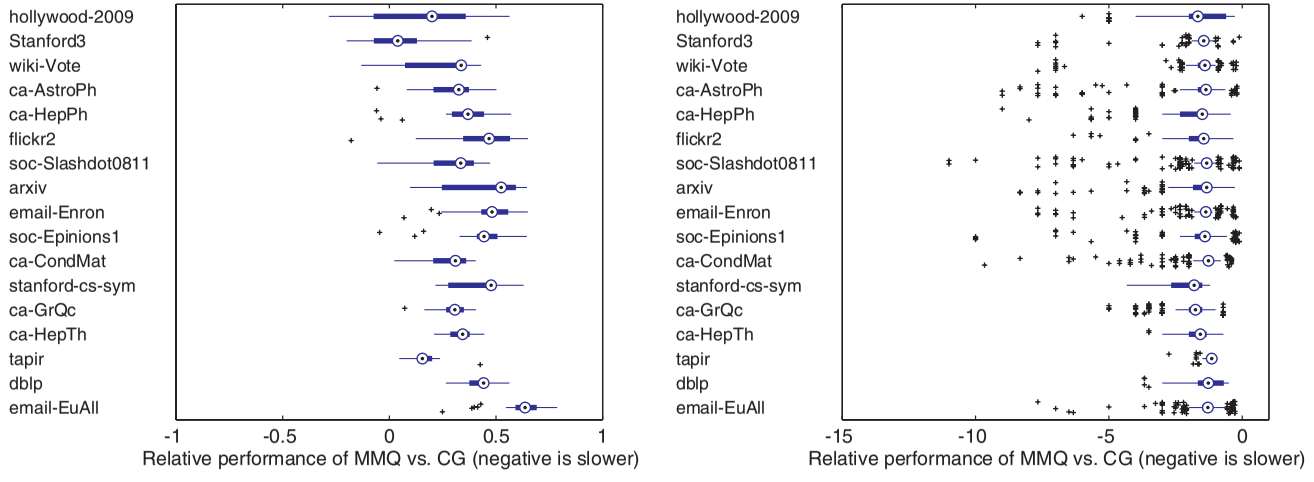

Figure 3. (Left) Relative performance between Algorithm 4 and conjugate gradient for pairwise commute times. (Right) Relative performance between Algorithm 5 and conjugate gradient for pairwise Katz scores. The relative performance measure is $(k_{\mathrm{cg}} - k_{\mathrm{alg}})/k_{\mathrm{cg}}$, where k is the number of matrix–vector products taken by each approach (color figure available online).

based on the same degree-sequence sampling mentioned in the last section. For values of k between 5 and 100, we show a box plot of the precision at k scores for four networks in Figure 4. In the same figure, we also show the result of using the heuristic $C_{i,j} \approx \frac{1}{D_{i,i}} + \frac{1}{D_{j,j}}$ suggested by [von Luxburg et al. 10]. This heuristic is called "inverse degree" in the figure, because it shows that the set S_k^* should look like the set of k nodes with highest degree or smallest inverse degree.

These results show that our approach for estimating a column of the commute-time matrix provides only partial information about the true set. However, these experiments reinforce the theoretical discussion in [von Luxburg et al. 10] that commute time provides little information beyond the degree distribution. Consequently, the results from our algorithm may provide more useful information in practice, although such a conclusion would require us to formalize the nature of the approximation error in this algorithm and involve a rather different kind of study.

Exact commute times. Computing commute times is challenging. As part of a separate project, the third author of this paper wrote a program to compute the exact eigenvalue decomposition of a combinatorial graph Laplacian in a distributed computing environment using the MPI and the ScaLAPACK libraries [Blackford et al. 96]. This program ignores the sparsity in the matrix and treats the problem as a dense matrix. We adapted this software to compute the pseudoinverse of the graph Laplacian as well as the commute times. We were able to run this code on graphs up to 100,000 nodes using approximately 10 to 20 nodes of a larger supercomputer. (The details varied by graph, and are not relevant for this paper.) For graphs with fewer than 20,000 nodes, the same program will compute all commute times on the previously mentioned desktop computer. Thus, we computed the exact commute times for all graphs except email-euAll, flickr2, and hollywood-2009.

6.6. Columnwise Katz Scores

We now come to evaluate the local algorithm for Katz scores. As with the pairwise algorithms, we first study the empirical convergence of the algorithm. However, the evaluation for the convergence here is rather different. Recall again that the point of the columnwise algorithms is to find the most closely related nodes. For Katz scores, these are the largest elements in a column (whereas for commute time, they were the smallest elements in the column). Thus, we again evaluate each algorithm in terms of the precision at k for the top-k set generated by our algorithms and the exact top-k set produced by solving the linear system. Natural alternatives are other iterative methods and specialized direct methods that

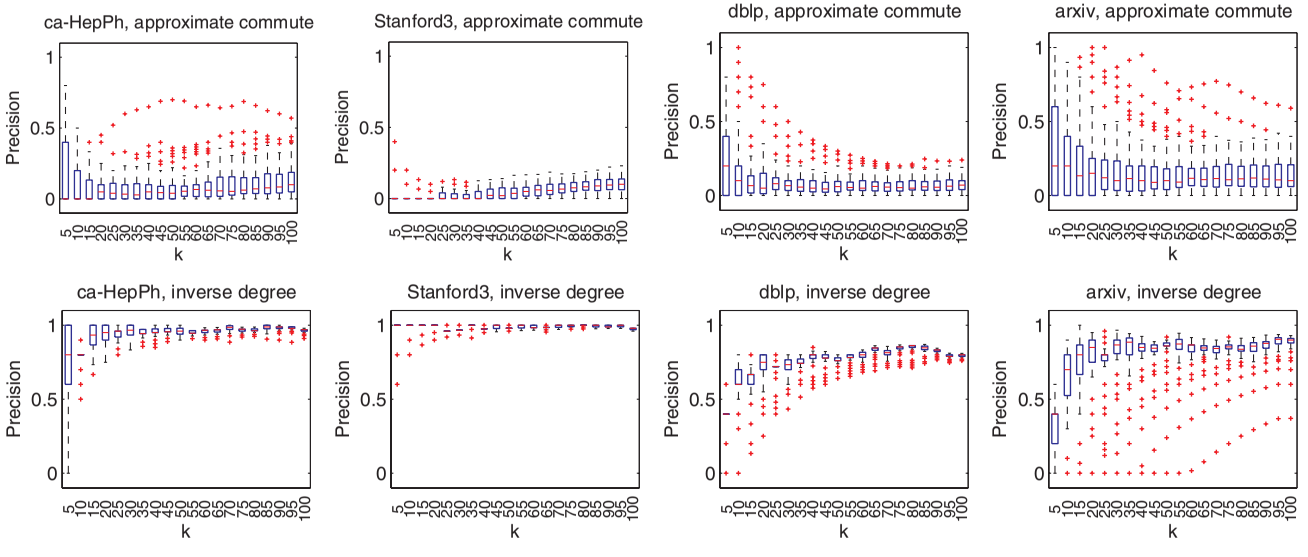

Figure 4. Precision at k for the columnwise commute-time approximations (top) over a few hundred trial columns. Precision at k for the inverse degree heuristic (bottom) over the same columns. These figures show standard box plots of the result for each column (color figure available online).

exploit sparsity. The latter—including approaches such as truncated commute time [Sarkar and Moore 07]—are beyond the scope of this work, since they require a different computational treatment in terms of caching and parallelization. Thus, we again use conjugate gradient (CG) as our point of comparison. The exact solution is computed by solving $(\boldsymbol{I} - \alpha \boldsymbol{A})\mathbf{k}_i = \mathbf{e}_i$, again using the MINRES method, to a tolerance of 10^{-12}.

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 the precision at k and the Kendall-τ measures should tend to 1 as we increase the work in our algorithm. Indeed, this is what we observe in Figure 5. For these figures, we pick a vertex with a fairly large degree and run Algorithm 6 with the "hard alpha" value mentioned in previous sections. As the algorithm runs, we track work with respect to the number of effective matrix–vector products. An effective matrix–vector product corresponds to our algorithm examining the same number of edges as a matrix–vector product. For example, 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. The idea is that the amount of work in one effective matrix–vector product is about the same as the amount of work in one iteration of CG. Hence, we can compare algorithms on this ground. As evident from the legend in each figure, we look at precision at k for four values of k, namely $10, 25, 100, 1000$, and also the Kendall-τ for these same values. While all of the measures should tend to 1 as we increase work, 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.

For comparison, we show results from the conjugate gradient method for the top-25 set after 2, 5, 10, 15, 25, and 50 matrix–vector products. In these results, the top-25 set has nearly converged after the equivalent of a single matrix–vector product, which is equivalent to just one iteration of the CG algorithm. The CG algorithm does not provide any useful information until it converges. Our top-k algorithm produces useful partial information with much less work.

6.7. Running Time

Finally, we show the empirical running time of our implementations in Tables 4 and 5. Table 4 describes the running time of the two pairwise algorithms. We show the 25th, 50th, and 75th percentiles of the time taken to compute the

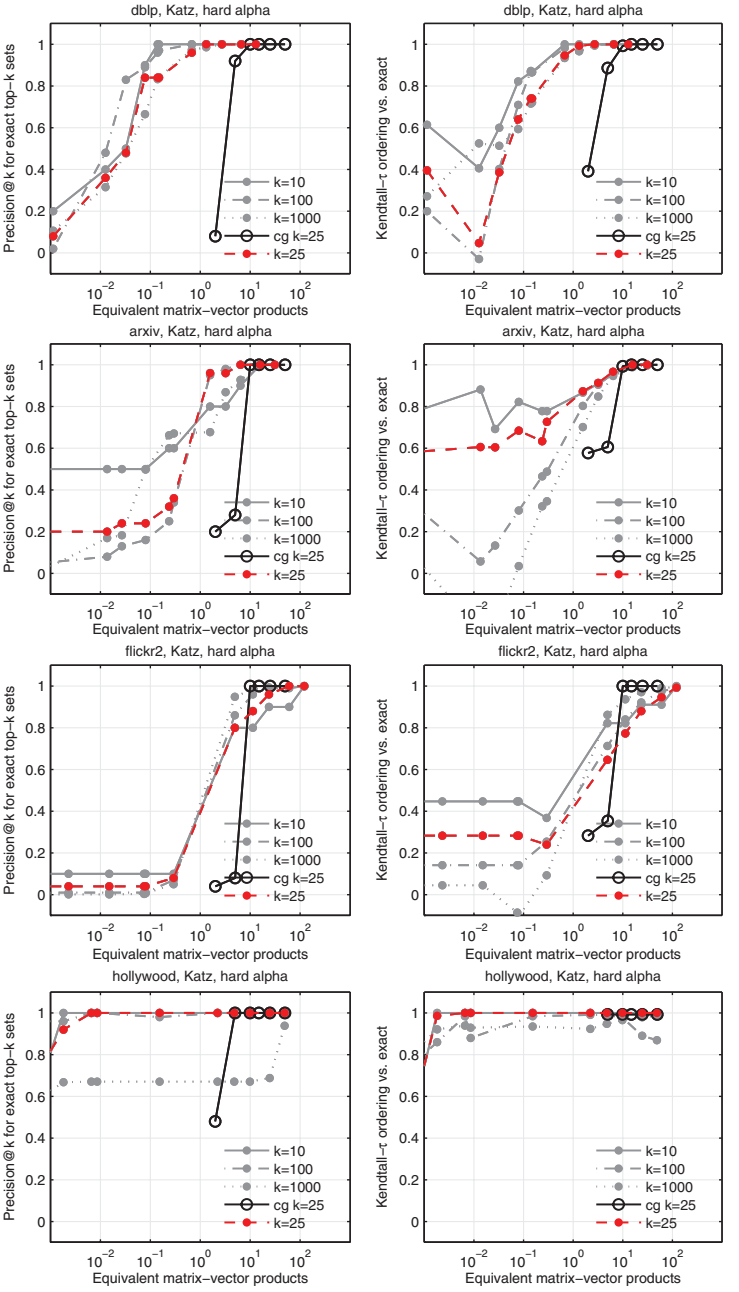

Figure 5. Convergence results for our columnwise Katz algorithm in terms of the precision of the top-k set (left) and the ordering of the true top-k set (right). See Section 6.6 for the discussion (color figure available online).

			Pairwise Katz running time (sec.)			Pairwise commute running time (sec.)		
Graph	Avg. Verts.	Deg.	25%	Median	75%	25%	Median	75%
tapir	1024	5.6	0.0	0.0	0.0	0.0	0.0	0.1
stanford-cs	2759	7.4	0.0	0.0	0.0	0.1	0.2	0.2
ca-GrQc	4158	6.5	0.0	0.0	0.0	0.1	0.1	0.1
wiki-Vote	7066	28.5	0.0	0.0	0.0	0.2	0.2	0.2
ca-HepTh	8638	5.7	0.0	0.0	0.0	0.1	0.2	0.2
ca-HepPh	11204	21.0	0.0	0.0	0.0	0.4	0.4	0.4
Stanford3	11586	98.1	0.2	0.2	0.2	0.6	0.7	0.7
ca-AstroPh	17903	22.0	0.1	0.1	0.1	0.5	0.5	0.7
ca-CondMat	21363	8.5	0.0	0.0	0.1	0.4	0.5	0.5
email-Enron	33696	10.7	0.1	0.1	0.1	1.1	1.2	1.3
soc-Epinions1	75877	10.7	0.2	0.2	0.2	2.8	3.2	3.7
soc-Slashdot0811	77360	12.1	0.2	0.2	0.2	2.6	2.8	3.4
arxiv	86376	12.0	0.2	0.3	0.3	4.8	6.0	6.5
dblp	93156	3.8	0.1	0.1	0.2	3.0	3.2	3.4
email-EuAll	224832	3.0	0.3	0.4	0.4	11.2	14.2	17.2
flickr2	513969	12.4	1.3	1.7	1.8	54.8	60.0	69.8
hollywood-2009	1069126	105.3	16.5	17.0	17.4	199.2	246.0	272.5

Table 4. Running time of the pairwise algorithms. The "0.0" second entries are rounded down for display. These are really just less than 0.1 seconds. The three columns for each type of problem show the 25th, 50th, and 75th percentiles of the wall-clock time to compute the results in Figure 3.

results from Figure 3. Our implementation is not optimized, and so these results indicate the current real-world performance of the algorithms.

Table 5 describes the running time of the columnwise Katz algorithm. Here, we picked columns of the matrix to approximate in two ways: (i) randomly, and (ii) to sample the entire degree distribution. As in previous experiments, we took the 1st, 2nd, 3rd, 4th, 5th, 10th, 20th ... vertices from the set of vertices sorted in order of decreasing degree. For each column picked in this manner, we ran Algorithm 6 and recorded the wall clock time. The 25th, 50th, and 75th percentiles of these times are shown in the table for each of the two sets of vertices.

Graph	Avg. Verts.	Deg.	Random columns running time (sec.)			Degree columns running time (sec.)		
			25%	Median	75%	25%	Median	75%
tapir	1024	5.6	0.0	0.0	0.0	0.0	0.0	0.0
stanford-cs-sym	2759	7.4	0.0	0.0	0.0	0.0	0.0	0.0
ca-GrQc	4158	6.5	0.0	0.0	0.0	0.0	0.0	0.0
wiki-Vote	7066	28.5	0.0	0.0	0.4	0.4	0.4	0.4
ca-HepTh	8638	5.7	0.0	0.0	0.0	0.0	0.0	0.0
ca-HepPh	11204	21.0	0.0	0.0	0.0	1.1	1.1	1.1
Stanford3	11586	98.1	0.0	0.0	1.7	1.8	1.9	1.9
ca-AstroPh	17903	22.0	0.0	0.0	0.0	0.6	0.7	0.9
ca-CondMat	21363	8.5	0.0	0.0	0.0	0.1	0.1	0.1
email-Enron	33696	10.7	0.0	0.0	0.0	0.9	1.0	1.1
soc-Epinions1	75877	10.7	0.0	0.0	0.0	3.7	4.1	4.5
soc-Slashdot0811	77360	12.1	0.0	0.0	0.0	2.4	2.8	3.7
arxiv	86376	12.0	0.0	0.0	0.0	0.0	0.6	0.7
dblp	93156	3.8	0.0	0.0	0.0	0.0	0.0	0.0
email-EuAll	224832	3.0	0.0	0.0	0.0	1.1	1.7	2.5
flickr2	513969	12.4	0.0	0.0	0.0	11.5	52.6	55.5
hollywood-2009	1069126	105.3	0.0	0.0	0.0	0.3	0.4	0.4

Table 5. Running time of the columnwise Katz algorithm. The "0.0" second entries are rounded down for display. These are really just less than 0.1 seconds. The three columns show the 25th, 50th, and 75th percentiles of the wall-clock time of the experiments described in Section 6.7.

For this algorithm, the degree of the target node has a considerable impact on the algorithm running time. This effect is particularly evident in the flickr2 data. The randomly chosen columns are found almost instantly, whereas the degree sampled columns take considerably longer. A potential explanation for this behavior is that starting at a vertex with a large degree will dramatically increase the residual at the first step. If these new vertices *also* have a large degree, then this effect will multiply, and the residual will rise for a long time before converging. Even in the cases in which the algorithm took a long time to converge, it explored only a small fraction of the graph (usually about 10% of the vertices), and so it retained its locality property. This property suggests that optimizing our implementation could reduce these running times.

7. Conclusion and Discussion

The goal of this manuscript is to estimate commute times and Katz scores in a rapid fashion. Let us summarize our contributions and experimental findings.

For the pairwise commute-time problem, we have implemented Algorithm 4, based on the relationship between the Lanczos process and a quadrature rule (Section 4.1). This algorithm uses a similar mechanism to that of conjugate gradient (CG). It outperforms the latter in terms of total matrix–vector products, because it provides upper and lower bounds that allow for early termination, whereas CG does not provide an easy way of detecting convergence for a specific pairwise score.

For the pairwise Katz problem, we have proposed Algorithm 5, also based on the same quadrature theory. This algorithm involves two simultaneous Lanczos iterations. In practice, this means more work per iteration than a simple approach based on CG. A careful implementation of Algorithm 5 would merge the two Lanczos processes into a "joint process" and perform the matrix–vector products simultaneously. In our tests of this idea, we have found that the combined matrix–vector product took only 1.5 times as long as a single matrix–vector product.

For the columnwise commute-time problem, we have investigated a variation of the conjugate gradient method that also provides an estimate of the diagonals of the matrix inverse. We have found that these estimates were fairly crude approximations of the commute-time scores. We have also investigated whether the degree-based heuristic from [von Luxburg et al. 10] provides better information. It indeed seems to perform better, which suggests that the smallest elements of a column of the commute-time matrix may not be a useful set of related nodes.

For the columnwise Katz algorithm, we have proposed Algorithm 6 based on the techniques used for personalized PageRank computing. The idea with these techniques is to exploit sparsity in the solution vector itself to derive faster algorithms. We have shown that this algorithm converges in two cases: Result 5.1, where we established a precise convergence result, and Theorem 5.3, where we established only asymptotic convergence.

We believe that these results paint a useful picture of the strengths and limitations of our algorithms. In the spirit of reproducible research, we make our data, computational codes, and figure-plotting codes available for others.[1]

Here are a few possible directions for future work:

[1] Available at http://cs.purdue.edu/homes/dgleich/codes/fast-katz-2011.

Alternatives for pairwise Katz. First, there are alternatives to using the identity

$$\mathbf{u}^T f(\boldsymbol{Z})\mathbf{v} = \frac{1}{4}(\mathbf{u}+\mathbf{v})^T f(\boldsymbol{Z})(\mathbf{u}+\mathbf{v}) - \frac{1}{4}(\mathbf{u}-\mathbf{v})^T f(\boldsymbol{Z})(\mathbf{u}-\mathbf{v})$$

in the $\mathbf{u} \neq \mathbf{v}$ case. The first alternative is based on the nonsymmetric Lanczos process [Golub and Meurant 10]. This approach still requires two matrix–vector products per iteration, but it directly estimates the bilinear form and also provides upper and lower bounds. A concern with the nonsymmetric Lanczos process is that it is possible to encounter degeneracies in the recurrence relationships that stop the process short of convergence. Another alternative is based on the block Lanczos process [Golub and Meurant 10]. However, this process does not yet offer upper and lower bounds.

A theoretical basis for the localization of Katz scores. The inspiration for the columnwise Katz algorithm was the highly successful personalized PageRank algorithms. The localization of these personalized PageRank vectors was made precise in a theorem from [Andersen et al. 06] that related the personalized PageRank vector to cuts in the graph. In brief, if there is a good cut close to a vertex, then the personalized PageRank vector will be localized on a few vertices. An interesting question is whether Katz matrices enjoy a similar property. We hope to investigate this question in the future.

References

[Acar et al. 09] Evrim Acar, Daniel M. Dunlavy, and Tamara G. Kolda. "Link Prediction on Evolving Data Using Matrix and Tensor Factorizations." In *ICDMW '09: Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, pp. 262–269. IEEE Computer Society, 2009.

[Andersen et al. 06] Reid Andersen, Fan Chung, and Kevin Lang. "Local Graph Partitioning Using PageRank Vectors." In *Proc. of the 47th Annual IEEE Sym. on Found. of Comp. Sci.*, 2006.

[Benzi and Boito 10] Michele Benzi and Paola Boito. "Quadrature Rule-Based Bounds for Functions of Adjacency Matrices." *Linear Algebra and Its Applications* 433:3 (2010), 637–652.

[Berkhin 07] Pavel Berkhin. "Bookmark-Coloring Algorithm for Personalized PageRank Computing." *Internet Math.* 3:1 (2007), 41–62.

[Blackford et al. 96] Laura Susan Blackford, J. Choi, A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, K. Stanley, J. Dongarra, S. Hammarling, G. Henry, and D. Walker. "Scalapack: A Portable Linear Algebra Library for Distributed Memory Computers—Design Issues and Performance." In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing (CDROM)*. Washington, DC: IEEE Computer Society, 1996.

[Boldi and Vigna 04] Paolo Boldi and Sebastiano Vigna. "The Webgraph Framework I: Compression Techniques." In *Proceedings of the 13th International Conference on the World Wide Web*, pp. 595–602. New York: ACM Press, 2004.

[Boldi et al. 11] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. "Layered Label Propagation: A Multiresolution Coordinate-Free Ordering for Compressing Social Networks." In *Proceedings of the 20th WWW2011*, pp. 587–596, March 2011.

[Chantas et al. 08] G. Chantas, N. Galatsanos, A. Likas, and M. Saunders. "Variational Bayesian Image Restoration Based on a Product of t-Distributions Image Prior." *IEEE Transactions on Image Processing* 17:10 (2008), 1795–1805.

[Chung et al. 03] Fan Chung, Linyuan Lu, and Van Vu. "Eigenvalues of Random Power Law Graphs." *Annals of Combinatorics* 7:1 (2003), 21–33.

[Davis and Rabinowitz 84] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration*, 2nd edition. New York: Academic Press, 1984.

[Davis and Hu 10] Timothy A. Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection." To appear in *ACM Transactions on Mathematical Software, 2010*. Available online (http://www.cise.ufl.edu/research/sparse/matrices/), 2010.

[Farkas et al 01] Illés J. Farkas, Imre Derényi, Albert-László Barabási, and Tamás Vicsek. "Spectra of 'Real-World' Graphs: Beyond the Semicircle Law." *Phys. Rev. E*, 64:2 (2001), 026704.

[Foster et al. 01] Kurt C. Foster, Stephen Q. Muth, John J. Potterat, and Richard B. Rothenberg. "A Faster Katz Status Score Algorithm." *Comput. & Math. Organ. Theo.* 7:4 (2001), 275–285.

[Fouss et al. 07] François Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. "Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation." *IEEE Trans. Knowl. Data Eng.* 19:3 (2007), 355–369.

[Gilbert and Teng 02] John R. Gilbert and Shang-Hua Teng. "MESHPART: Matlab Mesh Partitioning and Graph Separator Toolbox." Available online (http://www.cerfacs.fr/algor/Softs/MESHPART/), 2002.

[Göbel and Jagers 74] F. Göbel and A. A. Jagers. "Random Walks on Graphs." *Stochastic Processes and Their Applications*, 2:4 (1974), 311–336.

[Golub and Meurant 94] G. H. Golub and Gérard Meurant. "Matrices, Moments and Quadrature." In *Numerical analysis 1993 (Dundee, 1993)*, *Pitman Res. Notes Math. Ser.* 303, pp. 105–156. Harlow, Essex, UK: Longman Sci. Tech., 1994.

[Golub and Meurant 97] Gene H. Golub and Gérard Meurant. "Matrices, Moments and Quadrature II: How to Compute the Norm of the Error in Iterative Methods." *BIT Num. Math.* 37:3 (1997) 687–705.

[Golub and Meurant 10] Gene H. Golub and Gérard Meurant. *Matrices, Moments, and Quadrature with Applications*, Princeton Series in Applied Mathematics. Princeton: Princeton University Press, 2010.

[Hirai et al. 00] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. "Webbase: A Repository of Web Pages." *Computer Networks* 33:1–6 (2000), 277–293.

[Jeh and Widom 02] Glen Jeh and Jennifer Widom. "SimRank: A Measure of Structural-Context Similarity." In *Proc. of the 8th ACM Intl. Conf. on Know. Discov. and Data Mining (KDD'02)*, 2002.

[Katz 53] Leo Katz. "A New Status Index Derived from Sociometric Analysis." *Psychometrika* 18 (1953), 39–43.

[Lanczos 50] Cornelius Lanczos. "An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators." *Journal of Research of the National Bureau of Standards* 45:4 (1950), 255–282.

[Lanczos 53] Cornelius Lanczos. "Solution of Systems of Linear Equations by Minimized Iterations." *Journal of Research of the National Bureau of Standards* 49:1 (1953), 33–53.

[Lehoucq et al. 97] R. B. Lehoucq, D. C. Sorensen, and C. Yang. "ARPACK User's Guide: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods." Available online (http://www.caam.rice.edu/software/ARPACK/SRC/ug.ps.gz), 1997.

[Leskovec 10] Jure Leskovec. "The Stanford Large Network Dataset Collection." Available online (http://snap.stanford.edu/data/index.html), 2010.

[Leskovec et al. 07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. "Graph Evolution: Densification and Shrinking Diameters." *ACM Trans. Knowl. Discov. Data* 1 (2007), 1–41.

[Leskovec et al. 09] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. "Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters." *Internet Mathematics* 6:1 (2009), 29–123.

[Leskovec et al. 10] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. "Signed Networks in Social Media." In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, CHI '10, pp. 1361–1370. New York: ACM, 2010.

[Li et al. 10] Pei Li, Hongyan Liu, Jeffrey Xu Yu, Jun He, and Xiaoyong Du. "Fast Single-Pair SimRank Computation." In *Proc. of the SIAM Intl. Conf. on Data Mining (SDM2010)*, Columbus, OH, 2010.

[Liben-Nowell and Kleinberg 03] David Liben-Nowell and Jon M. Kleinberg. "The Link Prediction Problem for Social Networks." In *Proc. of the ACM Intl. Conf. on Inform. and Knowlg. Management (CIKM'03)*, 2003.

[Luo and Tseng 92] Z. Q. Luo and P. Tseng. "On the Convergence of the Coordinate Descent Method for Convex Differentiable Minimization." *Journal of Optimization Theory and Applications* 72:1 (1992), 7–35.

[McSherry 05] Frank McSherry. "A Uniform Approach to Accelerated PageRank Computation." In *Proc. of the 14th Intl. Conf. on the WWW*, pp. 575–582. New York: ACM, 2005.

[Mihail and Papadimitriou 02] Milena Mihail and Christos H. Papadimitriou. "On the Eigenvalue Power Law." In *RANDOM '02: Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, pp. 254–262. London: Springer, 2002.

[O'Leary 80] Dianne P. O'Leary. "The Block Conjugate Gradient Algorithm and Related Methods." *Linear Algebra and Its Applications* 29 (1980), 293–322.

[Page et al. 99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. "The PageRank Citation Ranking: Bringing Order to the Web." Technical Report 1999-66, Stanford University, November 1999. Available online (http://dbpubs.stanford.edu:8090/pub/1999-66).

[Paige and Saunders 75] C. C. Paige and M. A. Saunders. "Solution of Sparse Indefinite Systems of Linear Equations." *SIAM Journal on Numerical Analysis* 12:4 (1975), 617–629.

[Rattigan and Jensen 05] Matthew J. Rattigan and David Jensen. "The Case for Anomalous Link Discovery." *SIGKDD Explor. Newsl.* 7:2 (2005), 41–47.

[Richardson et al. 03] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. "Trust Management for the Semantic Web." In *The SemanticWeb—ISWC 2003*, Lecture Notes in Computer Science 2870, pp. 351–368. New York: Springer, 2003.

[Saerens et al. 04] Marco Saerens, François Fouss, Luh Yen, and Pierre Dupont. "The Principal Components Analysis of a Graph, and Its Relationships to Spectral Clustering." In *Proc. of the 15th Euro. Conf. on Mach. Learn.*, 2004.

[Sarkar and Moore 07] Purnamrita Sarkar and Andrew W. Moore. "A Tractable Approach to Finding Closest Truncated-Commute-Time Neighbors in Large Graphs." In *Proc. of the 23rd Conf. on Uncert. in Art. Intell. (UAI'07)*, 2007.

[Sarkar et al. 08] Purnamrita Sarkar, Andrew W. Moore, and Amit Prakash. "Fast Incremental Proximity Search in Large Graphs." In *Proc. of the 25th Intl. Conf. on Mach. Learn. (ICML'08)*, 2008.

[Saunders 07] Michael Saunders. "cgLanczos: CG Method for Positive Definite $ax = b$." Available online (http://www.stanford.edu/group/SOL/software/cgLanczos.html), accessed on 2011-03-25, 2007.

[Spielman and Srivastava 08] Daniel A. Spielman and Nikhil Srivastava. "Graph Sparsification by Effective Resistances." In *Proc. of the 40th Ann. ACM Symp. on Theo. of Comput. (STOC'08)*, pp. 563–568, 2008.

[Traud et al. 11] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. "Social Structure of Facebook Networks." arXiv, cs.SI: 1102.2166, February 2011.

[Varga 62] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, 1962.

[von Luxburg et al. 10] Ulrike von Luxburg, Agnes Radl, and Matthias Hein. "Getting Lost in Space: Large Sample Analysis of the Commute Distance." In *Advances in Neural Information Processing Systems 24 (NIPS2010)*, 2010.

[Wang et al. 07] Chao Wang, Venu Satuluri, and Srinivasan Parthasarathy. "Local Probabilistic Models for Link Prediction." In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pp. 322–331. Washington, DC: IEEE Computer Society, 2007.

[Yen et al. 07] 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.

Francesco Bonchi, Yahoo! Research, Avinguda Diagonal 177, 8th floor, 08018 Barcelona, Spain (bonchi@yahoo-inc.com)

Pooya Esfandiar, Ayogo Games, 210-314 W Cordova St., Vancouver, BC V6B 1E8, Canada (pooya@ayogo.com). Work completed at the University of British Columbia.

David F. Gleich, Purdue University, Computer Science Department, 305 N. University Ave. Rm. 1207, West Lafayette, IN 47907, USA (dgleich@purdue.edu). Work completed at Sandia National Laboratories.

Chen Greif, University of British Columbia, 2366 Main Mall, Department of Computer Science, Vancouver, BC V6T 1Z4, Canada (greif@cs.ubc.ca)

Laks V. S. Lakshmanan, University of British Columbia, 2366 Main Mall, Department of Computer Science, Vancouver, BC V6T 1Z4, Canada (laks@cs.ubc.ca)

Received April 4, 2011; accepted June 3, 2011.