

Fast Computation of Katz Index for Efficient Processing of Link Prediction Queries

Mustafa Coşkun · Abdelkader Baggag ·
Mehmet Koyutürk

Received: date / Accepted: date

Abstract Network proximity computations are among the most common operations in various data mining applications, including link prediction and collaborative filtering. A common measure of network proximity is Katz index, which has been shown to be among the best-performing path-based link prediction algorithms. With the emergence of very large network databases, such proximity computations become an important part of query processing in these databases. Consequently, significant effort has been devoted to developing algorithms for efficient computation of Katz index between a given pair of nodes or between a query node and every other node in the network. Here, we present LRC-KATZ, an algorithm based on indexing and low rank correction to accelerate Katz index based network proximity queries. Using a variety of very large real-world networks, we show that LRC-KATZ outperforms the fastest existing method, Conjugate Gradient, for a wide range of parameter values. We also show that, this acceleration in the computation of Katz index

Mustafa Coşkun
Department of Computer Engineering
Abdullah Gül University
Kayseri, Turkey
Tel.: +905-05-0082739
E-mail: mustafa.coskun@agu.edu.tr

Abdelkader Baggag
Qatar Computing Research Institute
Hamad Bin Khalifa University
Doha, Qatar
Tel.: +971-4454-7250
E-mail: abaggag@hbku.edu.qa

Mehmet Koyutürk
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, USA
Tel.: +121-63-682963
E-mail: mehmet.koyuturk@case.edu

can be used to drastically improve the efficiency of processing link prediction queries in very large networks. Motivated by this observation, we propose a new link prediction algorithm that exploits modularity of networks that are encountered in practical applications. Our experimental results on the link prediction problem shows that our modularity based algorithm significantly outperforms state-of-the-art link prediction Katz method.

Keywords Fast Katz Method · Link Prediction · Network Proximity

1 Introduction

Proximity measure computation for nodes in networks is a well-adapted operation in many data analytic applications. In link prediction or recommender systems, network proximity measures the node similarity in social networks [16, 22]. In information retrieval, anomalous link are ranked based on nodes proximity to other nodes [19]. In unsupervised learning, the network proximity quantify a cluster quality [21].

The general setting for network proximity queries is as follows: Given a query node and some network distance measurement, we aim at computing a score for all other nodes in the network, based on their proximity with respect to the query node. For instance, these measures of network distance include shortest path, which computes the minimum number of edges between two nodes, and Katz-based proximity, which can be defined as a measure in between any pair of nodes in a network that capture their relationship due to the various paths connecting these two pair of nodes. In many applications, Katz-based proximity are preferable to shortest path distance, since it captures the global structure of a network and multi-paths relationships of the nodes [5, 14]. Katz-based proximity measures have been used in a number of applications, including link prediction [16], clustering [21], and ranking [19].

Motivated by social network data mining problems for instance link prediction and collaborative filtering, significant efforts have been devoted to reducing computational costs associated Katz-based proximity. These efforts typically speed-up computations by taking one of the following approaches: (i) exploiting numerical properties of iterative methods, along with structural characteristics of the underlying networks to speed up query processing; (ii) avoiding iterative computations during query processing by inverting the underlying system of equations using Cholesky factorization and storing the resulting factorization as an index. For instance, in the context of top- k Katz-based proximity queries, the state-of-the-art methods [5] use breadth-first ordering of the nodes in the network to bound element-wise increments of proximity scores by exploiting a relationship between the Lanczos process and a quadrature rule in the iterative computation [5]. This bounding process eliminates nodes whose proximity values cannot exceed those of the nodes that are already among the top- k most proximate to the query node by only entering some part of the underlying network. Likewise, using Cholesky factorization of the underlying linear system, top- k proximity computations can be computed

efficiently [20]. These approaches have been demonstrated to yield significant improvement in computation time, however, their application to larger networks is limited. Specifically, for very large networks, iterative methods [5, 20] require a large number of iterations to converge. More concretely, Fast-Katz [5] offers a tight convergence upper bound, however, Conjugate Gradient (CG) performs better than Fast-Katz [5] for Katz-based proximity. On the other hand, direct inversion techniques [20] are not scalable to large matrices, since the inverse of a sparse matrix is usually dense.

In this paper, we propose a hybrid approach that partitions the network into disjoint sub-networks and inverts the small matrices corresponding to these subnetworks. The denser matrix, composed of the nodes connecting these subnetworks is handled through a low rank corrected iterative CG method. By inverting small block diagonal matrices, the hybrid procedure overcomes the memory constraint of direct inversion techniques. By performing an iterative low rank corrected procedure on a smaller matrix, it overcomes the computational cost considerations of classical CG method. We first use graph partitioning [13] to partition the matrix into a sparse block-diagonal matrix and a dense, but much smaller matrix. And, we index the inverse of the sparse block diagonal matrix through a computationally efficient procedure. We then propose an efficient low rank correction technique along with CG, to speed up the iterative process for the dense matrix. At query time of Katz-based proximity, we first solve the dense part of the system using the low rank corrected iterative solver, and use the stored index (of the block diagonal matrix) to solve the rest of the system.

We describe these processes in detail, and show that the resulting hybrid approach converges much faster than classical CG. Our hybrid approach significantly accelerates the computation of Katz-based proximity queries for very large graphs. We provide a detailed theoretical justifications for our results and experimentally show the superior performance of our method on a number of real-world networks. Specifically, we show that our method yields over at least 3-fold improvement in the runtime of online query processing over the best state-of-the-art method, CG across all our experiments.

Moreover, we also show that, low rank corrected Katz-based proximity can be used to drastically improve the efficiency of processing link prediction queries in very large networks. Motivated by skewed distribution of Katz scores [5], we propose a new link prediction algorithm, called SPARSE-KATZ that exploits modularity of networks that are encountered in practical applications. Our experimental results on the link prediction problem shows that our modularity based algorithm significantly outperforms state-of-the-art link prediction Katz method.

In summary, the two main contributions of the proposed framework are the following:

- We introduce a hybrid approach to indexing-based acceleration of Katz-based network proximity queries, in which the network is divided into two components, where the larger and sparser part of the resulting system

is solved by indexing the inverse of the corresponding matrix, and the smaller and denser part of the system is solved during query processing using proposed low rank corrected CG method.

- We introduce another link prediction algorithm that renders Katz-based link prediction approach more effective by exploiting the skewed distribution of Katz scores.

Taken together, these two contributions bring the field closer to real-time processing of proximity queries as well as link prediction task on very large networks.

The rest of the paper is organized as follows: in the next section, we provide a review of the literature on efficient processing of Katz-based network proximity and link prediction. In Section 3, we define Katz-based proximity and link prediction problem, and describe our method, along with its theoretical justifications. In Section 4, we provide detailed experimental assessments of our method on very large networks for both Katz based proximity and its link prediction task. We conclude our discussion and summarize avenues for future research in Section 5.

2 Related Work

Node proximity queries have been soaring significant research attention in recent years in the context of searching, ranking, clustering and analyzing network structured object similarity [6]. In particular, Katz-based node proximity queries in large graph have been well studied [5]. One of the commonly used approaches to computing Katz-base based proximity is the power method through the Neumann series expansion of the underlying linear systems of equations [20].

An alternate approach to power iterations is the use of offline computation, which directly inverts the underlying linear system of equations, typically using Cholesky factorization or eigen-decomposition [1, 23, 24]. These methods tend compute network proximity rapidly, using a single matrix vector multiplication, however, they involve in some expensive preprocessing, and their memory requirements constrain their use to smaller networks. In contrast, we here propose to compute an approximate solution to linear system of equations by approximating the inverse of Schur Matrix with low rank correction approach, and use this low rank approximation as a preconditioner in CG to compute the exact solution at query time. In doing so, we significantly improve on both memory and storage requirements for indexing as well as runtime of proximity computation.

There have also been extensive efforts aimed at scaling top- k proximity queries to large sparse networks for Katz-based proximity. These methods utilize the topology of the network to perform a local search around the query node, based on the premise that nodes with high random-walk based proximity to the query node are also close to the query node in terms of the number of

hops by exploiting a relationship between the Lanczos process and a quadrature rule in the iterative computation [5]. However, these local search based methods for Katz proximity computation are not as efficient as CG method.

In this paper, we focus on exact computation of Katz proximity in very large networks using a novel, hybrid approach accelerated with low rank correction. Our method is fundamentally different from existing approaches in that it simultaneously targets scalability and efficiency. Our method can be used for efficiently computing Katz-based proximity of all nodes in a undirected network. Moreover, we also develop a new link prediction algorithm that takes advantages of sparse distribution Katz scores and drastically improves the effectiveness of Katz-based link prediction approach.

3 Methods

In this section, we first define Katz-index and formulate node proximity queries based on Katz index. We then describe our approach to indexing, which is based on graph-partitioning indexing, i.e., to partition the resulting linear system and to index the sparser part of the system. Subsequently, we discuss how the iterative computation can be accelerated using low rank correction to refine the solution, and solve the remaining part of the linear system. Finally, we discuss how these two approaches can be used in combination, to efficiently process Katz-based network proximity queries. The workflow of the proposed framework is shown in Figure 1.

3.1 Katz Index

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected and connected network, where \mathcal{V} denotes the set of $|\mathcal{V}|$ nodes and \mathcal{E} denotes the set of edges, with sizes indicated as $|\mathcal{V}|$ and $|\mathcal{E}|$, respectively. Katz index quantifies the proximity between a pair of nodes in this network as the weighted sum of all paths connecting the two nodes, where the weights of the paths decay exponentially with path length [5, 14]. Namely, for a pair of nodes i and $j \in \mathcal{V}$, the Katz index is defined as:

$$K_{i,j} = \sum_{l=1}^{\infty} \alpha^l paths_l(i, j), \quad (1)$$

where $paths_l(i, j)$ denotes the number of paths of length l connecting i and j in \mathcal{G} . The parameter α is a damping factor that is used to tune the relative importance of longer paths, where $0 < \alpha < 1$, thus smaller α assigns more importance to shorter paths [5].

By observing the relationship between the number of paths in \mathcal{G} and the powers of the adjacency matrix \mathbf{G} of \mathcal{G} , of size $|\mathcal{V}| \times |\mathcal{V}|$, the computation of Katz index can be formulated as an algebraic problem. To see this, let \mathbf{K} denote the *Katz matrix*, i.e., the matrix of Katz indices between all pairs of nodes in \mathcal{G} .

Since $(\mathbf{G}^l)_{i,j}$ is equal to the number of paths of length l between i and j , \mathbf{K} can be written as:

$$\mathbf{K} = \sum_{l=1}^{\infty} \alpha^l \mathbf{G}^l = (\mathbf{I} - \alpha \mathbf{G})^{-1} - \mathbf{I}, \quad (2)$$

where \mathbf{I} is the identity matrix. In the rest of our discussion, we assume that $\alpha < 1/\|\mathbf{G}\|_2$ to ensure that $(\mathbf{I} - \alpha \mathbf{G})$ is positive definite, and to guarantee the convergence of the Neumann series to the inverse of $(\mathbf{I} - \alpha \mathbf{G})$.

For a given query node q , the Katz index of every other node in \mathcal{G} with respect to q is given by the q th column of \mathbf{K} , which we denote \mathbf{k}_q . Observe that the computation of \mathbf{k}_q corresponds to solving the following linear system:

$$\mathbf{M} \mathbf{k}_q = \alpha \mathbf{g}_q. \quad (3)$$

Here, $\mathbf{M} = (\mathbf{I} - \alpha \mathbf{G})$ and \mathbf{g}_q is the q th column of the adjacency matrix \mathbf{G} . Since \mathbf{M} does not depend on the query node q , it can be inverted offline, and the inverse can be used as an index to compute $\mathbf{k}_q = \alpha \mathbf{M}^{-1} \mathbf{g}_q$ by performing a single matrix vector multiplication during query processing. However, inverting \mathbf{M} and storing \mathbf{M}^{-1} is not feasible for very large graphs, since the inverse of a general sparse matrix is dense. Therefore, existing algorithms solve this linear system of equations using an iterative solver such as the (preconditioned) Conjugate Gradient method [20, p. 196], which is applicable in this case since \mathbf{M} is symmetric [5]. While these iterative methods greatly accelerate the computation of Katz index, they are not fast enough to enable real-time query processing in very large graphs.

Recently, to enable efficient processing of random-walk based queries on billion-scale networks, we have developed I-CHOPPER, a hybrid method that uses a combination of indexing and accelerated iterative solvers [6]. In the following subsection, we show how a similar idea can be applied to the processing of Katz index based queries by indexing the inverse of the matrix that corresponds to sparser parts of the network and performing low-rank correction at query time to obtain an exact solution for the rest of the network. We then describe how the resulting algorithm, LRC-KATZ, can be used to efficiently perform Katz index based link prediction.

3.2 Graph Partitioning Based Indexing

As in I-CHOPPER, LRC-KATZ exploits the sparsity of real-world networks to efficiently compute and index the inverse of a large part of \mathbf{M} . The key insight behind this approach is that, although the inverse of a general sparse matrix is dense, the inverse of a block-diagonal matrix with a low bandwidth is sparse [p.87] [20].

Since most real-world networks are scale-free, most of the nodes in the network have low degree. Consequently, observing that the non-zero structure of the matrix \mathbf{M} is identical to that of the adjacency matrix of the network,

we can reorder the rows of \mathbf{M} such that \mathbf{M} can be partitioned into a very large block-diagonal matrix with a low bandwidth (corresponding to small connected subgraphs consisting of low-degree nodes) and relatively dense but much smaller matrices (corresponding to hubs and their connections to other nodes). The following lemma shows how such partitioning of \mathbf{M} can be used to separate the solution of the linear system of Equation 3 into two parts:

Lemma 1 *Suppose a linear system $\mathbf{M}\mathbf{k}_q = \tilde{\mathbf{g}}_q$, can be partitioned as*

$$\begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{12}^T & \mathbf{M}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{k}_{q1} \\ \mathbf{k}_{q2} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{g}}_{q1} \\ \tilde{\mathbf{g}}_{q2} \end{bmatrix}, \quad (4)$$

such that \mathbf{M}_{11} is invertible. Letting $\mathbf{S} = \mathbf{M}_{22} - \mathbf{M}_{12}^T \mathbf{M}_{11}^{-1} \mathbf{M}_{12}$ denote the Schur complement, the linear system can be solved as:

$$\mathbf{k}_{q2} = \mathbf{S}^{-1} (\tilde{\mathbf{g}}_{q2} - \mathbf{M}_{12}^T \mathbf{M}_{11}^{-1} \tilde{\mathbf{g}}_{q1}), \quad (5)$$

$$\mathbf{k}_{q1} = \mathbf{M}_{11}^{-1} (\tilde{\mathbf{g}}_{q1} - \mathbf{M}_{12} \mathbf{k}_{q2}). \quad (6)$$

Proof The proof of this lemma is straightforward, and hence it is not included.

In our application, $\tilde{\mathbf{g}}_q = \alpha \mathbf{g}_q$. This lemma applies to the computation of Katz indices as long as $\alpha < 1/\|\mathbf{G}\|_2$, since \mathbf{M} is diagonally dominant and invertible in that case. In the light of this lemma, the computation of Katz indices can be performed as follows:

1. **Indexing:** Construct \mathbf{M} .
2. **Indexing:** Partition \mathcal{G} using multi-way minimum-vertex-separator partitioning.
3. **Indexing:** Reorder \mathbf{M} so that \mathbf{M}_{11} contains the internal edges of all parts resulting from the partitioning with nodes within each partition corresponding to successive rows (hence columns), $\mathbf{M}_{12} = \mathbf{M}_{12}^T$ contains the edges between nodes in partitions and nodes in the separator, and \mathbf{M}_{22} contains the edges between nodes in the separator.
4. **Indexing:** Compute and store \mathbf{M}_{11}^{-1} , \mathbf{M}_{12} , and \mathbf{S} .
5. **Query Processing:** For a query q , compute \mathbf{k}_{q2} as given in Equation (5), but without inverting \mathbf{S} , as described below.
6. **Query Processing:** Compute \mathbf{k}_{q1} by performing two matrix-vector multiplications as given in Equation (6).

This procedure is identical to the procedure implemented in I-CHOPPER, with one important difference in the computation of \mathbf{k}_{q2} during query processing (Step 5). In I-CHOPPER, this computation is accelerated using Chebyshev polynomials over the elliptic plane [6]. In the computation of Katz-index, \mathbf{S} is symmetric [p.271] [20], thus the elliptic plane degrades to the real axis and the solution can be found by using Chebyshev polynomials on the real axes [7]. However, this requires knowledge of the largest and smallest eigenvalues of \mathbf{S} , and it is costly to compute these eigenvalues. Since the matrix in question

is not symmetric in queries involving random walks, I-CHOPPER addresses this problem by pre-computing these eigenvalues using Arnoldi's method [p. 160] [20]. In the computation of Katz-index, however, \mathbf{S} is symmetric, and therefore this computation can be avoided by utilizing methods that do not require an eigen-bound. Specifically, we use the Conjugate Gradient (CG) method to solve the linear system involving \mathbf{S} , since CG does not require the knowledge of the largest and smallest eigenvalues of \mathbf{S} . Still, this computation can be accelerated using eigenvectors of a low-rank approximation of \mathbf{S}^{-1} .

Since the details of all other steps are described in [6], here we briefly describe the key idea in each step. We then focus on the description of low-rank approximation and describe this approach in detail.

Steps 2 and 3 – Partitioning of \mathcal{G} and Reordering of \mathbf{M} : The idea behind the partitioning of the network is to reorder the rows and columns of \mathbf{M} in such a way that we can obtain a block diagonal \mathbf{M}_{11} with a small bandwidth, i.e., the non-zero entries in matrix \mathbf{M}_{11} are condensed around its diagonal, ensuring that \mathbf{M}_{11}^{-1} is sparse. To accomplish this, we use multi-way minimum-vertex-cover partitioning to partition the nodes of \mathcal{G} into p partitions such that each node in partition Π_i are connected only to nodes in Π_i or to a set Π_s of nodes that are classified as the “vertex-separator” [12, 13]. Given such a partitioning, we reorder the matrix \mathbf{M} such that the rows/columns that correspond to nodes in the partitions $\Pi_1, \Pi_2, \dots, \Pi_p$ are ordered next to each other, and rows/columns that correspond to the nodes in Π_s are at the bottom/right of the matrix. As a result, the reordered matrix \mathbf{M} can be divided into the following sub-matrices: (i) \mathbf{M}_{11} contains the non-zeros that correspond to the edges within the partitions, (ii) \mathbf{M}_{12} and \mathbf{M}_{12}^T contain the non-zeros that correspond to the edges between nodes in a partition and nodes in Π_s , (iii) \mathbf{M}_{22} contains the non-zeros that correspond to the edges between nodes Π_s . Since minimum-vertex-separator graph partitioning is a NP-hard problem, we use a heuristic that is well-suited to our application. Namely, the PART-GRAPHRECURSIVE package implemented in the MeTiS graph partitioning tool [21] allows the user to put a threshold on the size of the vertex separator, as opposed to minimizing it, and recursively bipartitions the network until this threshold is reached. Therefore, we can directly control the size of \mathbf{S} (number of rows/columns of \mathbf{S} is equal to the number of nodes in the vertex separator) and the recursive partitioning generates many small partitions with roughly equal sizes, thereby keeping the bandwidth of \mathbf{M}_{11} small.

Step 4 – Computation of \mathbf{M}_{11}^{-1} and \mathbf{S} . Once \mathbf{M}_{11} is constructed, we invert \mathbf{M}_{11}^{-1} , which is also relatively sparse and can be stored as an index. Here, we remark that the inversion of \mathbf{M}_{11} is feasible even for *graphs with hundred millions of nodes* since it is block diagonal with a small bandwidth and there exists many efficient algorithms for inverting banded matrices [6]. In our implementation, we use the Incomplete Cholesky factorization along with approximate minimum algorithms [2, 3] before we invert the sparse block diagonal matrix \mathbf{M}_{11} . Once \mathbf{M}_{11}^{-1} is available, we compute \mathbf{S} as defined in Lemma 1, and store \mathbf{M}_{11}^{-1} , \mathbf{S} , and \mathbf{M}_{12} .

Step 5 – Computation of \mathbf{k}_{q2} During Query Processing.

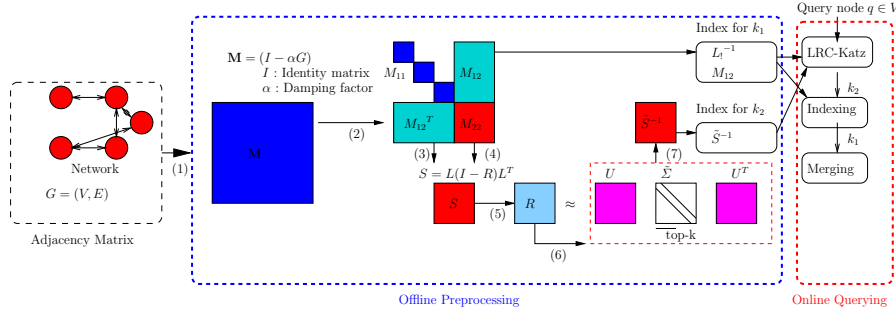


Fig. 1: **Flowchart illustrating the proposed framework for indexed processing of Katz-based proximity queries on large networks.**

Recall that processing of a Katz index query involves the computation of \mathbf{k}_q for a given query node q . As described in Lemma 1, we divide the computation of \mathbf{k}_q into the computation of \mathbf{k}_{q1} and the computation of \mathbf{k}_{q2} . Since the computation of \mathbf{k}_{q1} required knowledge of \mathbf{k}_{q2} , we first compute \mathbf{k}_{q2} during query processing. This computation requires solution of the system

$$\mathbf{S}\mathbf{k}_{q2} = (\mathbf{g}_2 - \mathbf{M}_{12}^T \mathbf{M}_{11}^{-1} \mathbf{g}_1) = \mathbf{f}, \quad (7)$$

where \mathbf{f} can be computed efficiently (by performing a single matrix-vector multiplication) during query processing, since we form and index \mathbf{M}_{12}^T and \mathbf{M}_{11}^{-1} in Steps 3 and 4. However, solving the linear system $\mathbf{S}\mathbf{k}_{q2} = \mathbf{f}$, during query processing or pre-computing and storing the inverse of \mathbf{S} is not feasible since \mathbf{S} is a relatively dense matrix. For this reason, we compute a low-rank approximation for \mathbf{S} offline and store this approximation as an index that can be used to efficiently compute \mathbf{k}_{q2} during query processing. We now explain this process. To avoid cluttered notation, we drop the subscripts (q) in the following sections.

3.2.1 Low Rank Correction

The idea behind Low Rank Correct Katz Algorithm (LRC-KATZ) is as follows: To solve $\mathbf{S}\mathbf{k}_2 = \mathbf{f}$, we approximate the Schur complement $\mathbf{S} \in \mathbb{R}^{n_2 \times n_2}$ via \mathbf{M}_{22} plus some low rank vectors so that we use sparser matrices instead of dense matrix \mathbf{S} .

Let $\mathbf{M}_{22} = \mathbf{L}\mathbf{L}^T$ be the Cholesky factorization of \mathbf{M}_{22} and recall that the Schur complement matrix can be rewritten as

$$\mathbf{S} = \mathbf{L}\mathbf{L}^T - \mathbf{M}_{12}^T \mathbf{M}_{11}^{-1} \mathbf{M}_{12} \quad (8)$$

$$= \mathbf{L}(\mathbf{I} - \mathbf{L}^{-1} \mathbf{M}_{12}^T \mathbf{M}_{11}^{-1} \mathbf{M}_{12} \mathbf{L}^{-T}) \mathbf{L}^T \quad (9)$$

$$= \mathbf{L}(\mathbf{I} - \mathbf{R}) \mathbf{L}^T \quad (10)$$

Now define the eigen-decomposition of the symmetric matrix \mathbf{R} as follows:

$$\mathbf{R} = \mathbf{L}^{-1} \mathbf{M}_{12}^T \mathbf{M}_{11}^{-1} \mathbf{M}_{12} \mathbf{L}^{-T} = \mathbf{U} \mathbf{\Sigma} \mathbf{U}^T, \quad (11)$$

where the diagonal entries of Σ are the eigenvalues of \mathbf{R} and \mathbf{U} is the column matrix that contains the corresponding eigenvectors, which are orthogonal to each other. Then \mathbf{S} can be rewritten as:

$$\mathbf{S} = \mathbf{L}(\mathbf{I} - \mathbf{R})\mathbf{L}^T = \mathbf{L}(\mathbf{I} - \mathbf{U}\Sigma\mathbf{U}^T)\mathbf{L}^T = \mathbf{L}\mathbf{U}(\mathbf{I} - \Sigma)(\mathbf{L}\mathbf{U})^T. \quad (12)$$

Thus, the inverse of the Schur complement matrix \mathbf{S} becomes:

$$\begin{aligned} \mathbf{S}^{-1} &= \left(\mathbf{L}\mathbf{U}(\mathbf{I} - \Sigma)(\mathbf{L}\mathbf{U})^T \right)^{-1} \\ &= \mathbf{L}^{-T}\mathbf{U}(\mathbf{I} - \Sigma)^{-1}\mathbf{U}^T\mathbf{L}^{-1} \\ &= \mathbf{L}^{-T} \left[\mathbf{I} + \mathbf{U}(\mathbf{I} - \Sigma)^{-1}\mathbf{U}^T - \mathbf{I} \right] \mathbf{L}^{-1} \\ &= \mathbf{M}_{22}^{-1} + \mathbf{L}^{-T}\mathbf{U} \left[(\mathbf{I} - \Sigma)^{-1} - \mathbf{I} \right] \mathbf{U}^T\mathbf{L}^{-1}. \end{aligned} \quad (13)$$

Now consider approximating \mathbf{R} using its most dominant ℓ eigenvectors. That is, define $\tilde{\mathbf{R}} \approx \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{U}}^T$, where $\tilde{\mathbf{U}} \in \mathbb{R}^{n_2 \times n_2}$, $\text{diag}(\tilde{\Sigma}) = (\sigma_1, \sigma_2, \dots, \sigma_\ell, 0, 0, \dots, 0)$ and $\tilde{\mathbf{U}}$ consists of first ℓ eigenvectors of \mathbf{U} padded with zeros, i.e.,:

$$\mathbf{R} \approx \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{U}}^T$$

$$= \begin{pmatrix} \begin{matrix} u_1 & u_l & u_{l+1} & u_{n_2} \\ \begin{matrix} \vdots \\ \vdots \end{matrix} & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{matrix} \end{pmatrix} \begin{pmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_l & \\ & & & 0 \\ & & & & \ddots \\ & & & & & 0 \end{pmatrix} \begin{pmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{pmatrix} \begin{matrix} u_1^T \\ u_l^T \\ u_{l+1}^T \\ u_{n_2}^T \end{matrix} \end{pmatrix} \begin{matrix} \left. \begin{matrix} u_1^T \\ u_l^T \\ u_{l+1}^T \end{matrix} \right\} \ell \\ \left. u_{n_2}^T \right\} 0 \end{matrix}$$

Using this approximation to \mathbf{R} , we define an approximation to \mathbf{S}^{-1} as follows:

$$\tilde{\mathbf{S}}^{-1} = \mathbf{M}_{22}^{-1} + \mathbf{L}^{-T}\tilde{\mathbf{U}}[(\mathbf{I} - \tilde{\Sigma})^{-1} - \mathbf{I}]\tilde{\mathbf{U}}^T\mathbf{L}^{-1} \quad (14)$$

Note that we never compute $\tilde{\mathbf{S}}^{-1}$ in practice, we define it here solely for theoretical justification.

The following theorem establishes the relationship between the eigenvalues of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^{-1}$ and the eigenvalues of \mathbf{R} .

Theorem 1 Assume that the eigenvalues of \mathbf{R} are ordered as $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{n_2}$, where n_2 is size of \mathbf{S} . For a given integer ℓ , define $\tilde{\mathbf{S}}^{-1}$ as in Equation (14). Then, the eigenvalues of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^{-1}$ are in the form of

$$\lambda_i = \begin{cases} 1 & \text{if } i \leq \ell. \\ 1 - \sigma_i & \text{otherwise} \end{cases}$$

Proof From Equations (13) and (14), we can write $\mathbf{S}^{-1} - \tilde{\mathbf{S}}^{-1} = \mathbf{L}^{-T} \tilde{\mathbf{U}}[(\mathbf{I} - \boldsymbol{\Sigma})^{-1} - (\mathbf{I} - \tilde{\boldsymbol{\Sigma}})^{-1}] \tilde{\mathbf{U}}^T \mathbf{L}^{-1}$. Then, we have,

$$\mathbf{L}^T \mathbf{S}^{-1} \mathbf{L} - \mathbf{L}^T \tilde{\mathbf{S}}^{-1} \mathbf{L} = \tilde{\mathbf{U}}[(\mathbf{I} - \boldsymbol{\Sigma})^{-1} - (\mathbf{I} - \tilde{\boldsymbol{\Sigma}})^{-1}] \mathbf{U}^T.$$

Multiplying both sides of the above equality with $(\mathbf{L}^T \mathbf{S}^{-1} \mathbf{L}) \tilde{\mathbf{U}}^{-1} = \mathbf{L}^{-1} \mathbf{S} \mathbf{L}^{-T}$, we have

$$\mathbf{I} - \mathbf{L}^{-1} \mathbf{S} \tilde{\mathbf{S}}^{-1} \mathbf{L} = \mathbf{L}^{-1} \mathbf{S} \mathbf{L}^{-T} \tilde{\mathbf{U}}[(\mathbf{I} - \boldsymbol{\Sigma})^{-1} - (\mathbf{I} - \tilde{\boldsymbol{\Sigma}})^{-1}] \tilde{\mathbf{U}}^T.$$

From the definition, we know that $\mathbf{L}^{-1} \mathbf{S} \mathbf{L}^{-T} = (\mathbf{I} - \mathbf{R}) = \tilde{\mathbf{U}}(\mathbf{I} - \boldsymbol{\Sigma}) \tilde{\mathbf{U}}^T$. Then, by using orthogonality of \mathbf{U} , we have

$$\mathbf{I} - \mathbf{L}^{-1} \mathbf{S} \tilde{\mathbf{S}}^{-1} \mathbf{L} =$$

$$\tilde{\mathbf{U}}(\mathbf{I} - \boldsymbol{\Sigma}) \tilde{\mathbf{U}}^T \tilde{\mathbf{U}}[(\mathbf{I} - \boldsymbol{\Sigma})^{-1} - (\mathbf{I} - \tilde{\boldsymbol{\Sigma}})^{-1}] \tilde{\mathbf{U}}^T$$

$$\tilde{\mathbf{U}}[\mathbf{I} - (\mathbf{I} - \boldsymbol{\Sigma})(\mathbf{I} - \tilde{\boldsymbol{\Sigma}})^{-1}] \tilde{\mathbf{U}}^T.$$

Finally, we have,

$$\mathbf{S} \tilde{\mathbf{S}}^{-1} = (\mathbf{L} \tilde{\mathbf{U}})[(\mathbf{I} - \boldsymbol{\Sigma})(\mathbf{I} - \tilde{\boldsymbol{\Sigma}})^{-1}](\mathbf{L} \tilde{\mathbf{U}})^{-1}.$$

Q.E.D.

It follows from this theorem that if we compute the top ℓ eigenvectors of \mathbf{R} matrix, we can use these eigenvectors and \mathbf{M}_{22} to efficiently solve the system in Equation (7). This is because, in the iterative solution of (7), we multiply \mathbf{k}_2 vector by a matrix that contains $\ell \times \ell$ identity matrix on top instead of \mathbf{S} matrix at each iteration. From the theorem, we can approximate the first ℓ part of inverse of \mathbf{S} via low rank and since this $\ell \times \ell$ upper part of inverse of \mathbf{S} is already computed in the preprocessing phase, we automatically use precomputed part in the iterative computation of (7). Setting the iterative process this way, we eliminate $\ell \times \ell$ computation from equation (7).

Algorithm 1 The Preprocessing Phase

- 1: **procedure** PREPROCESS(\mathbf{G}, α, k)
 - 2: Construct $\mathbf{M} \leftarrow (\mathbf{I} - \alpha \mathbf{G})$
 - 3: Use minimum-vertex-seperator graph partitioning on \mathbf{M} to partition it into $\mathbf{M}_{11}, \mathbf{M}_{12}^T, \mathbf{M}_{12}, \mathbf{M}_{22}$ [12, 13]
 - 4: Decompose \mathbf{M}_{11} into \mathbf{L}_1 and \mathbf{L}_1^T using Cholesky factorization and invert \mathbf{L}_1 and \mathbf{L}_1^T
 - 5: Decompose \mathbf{M}_{22} into \mathbf{L} and \mathbf{L}^T using Cholesky factorization
 - 6: Create $\mathbf{l}^{(0)}$ as normalized random vector
 - 7: Compute $[\tilde{\mathbf{U}}, \tilde{\boldsymbol{\Sigma}}] = \text{Lanczos}(\mathbf{R} \mathbf{l}^{(0)}, k, \mathbf{l}^{(0)})$ [9] ▷ As matrix-vector product
-

3.2.2 The LRC-KATZ Algorithm

In this section, we outline our algorithm for Katz-based network proximity computation. In “offline” preprocessing Algorithm 1, we first construct \mathbf{M} and use *PartGraphRecursive* in METIS to partition \mathbf{M} in such a way that \mathbf{M}_{11} is a sparse block diagonal matrix, and \mathbf{M}_{22} is dense but smaller [12, 13]. Next, we reorder the entries of partitioned matrix \mathbf{M} based on an approximate minimum degree ordering (AMD) [2, 3]. After reordering entries of \mathbf{M} , we invert the Cholesky factorization of sparse block-diagonal matrix \mathbf{M}_{11} and obtain \mathbf{L}_1^{-1} and \mathbf{L}_1^{-T} . Then, we construct $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{S}}$ via *Lanczos* procedure [9] without forming \mathbf{R} matrix. Subsequently, we form matrices for $\tilde{\mathbf{S}}$ and store the resulting values and matrices into an index to use them in query processing phase of our algorithm, LRC-KATZ.

Algorithm 2 The LRC-KATZ Algorithm

```

1: procedure LRC-KATZ
2:   Partition vector  $e_q$  into  $e_1$  and  $e_2$  for query,  $q$ 
3:   Construct  $b_1$ ,  $b_2$ , and  $f = (g_2 - \mathbf{M}_{12}^T \mathbf{M}_{11}^{-1} g_1)$ 
4:   Create  $k_2^{(0)}$  as normalized random vector
5:   Set  $i = 0$ ,  $r^{(i)} = f - \mathbf{S}k_2^{(i)}$ ,  $s = \mathbf{S}r^{(i)}$ ,  $p = \tilde{\mathbf{S}}s^{(i)}$ ,  $y^{(i)} = \tilde{\mathbf{S}}r^{(i)}$   $\triangleright \mathbf{S}$  and  $\tilde{\mathbf{S}}$  are
   used as matrix-vector product
6:    $\gamma^{(i)} = y^{(i)T} s^{(i)}$ 
7:   if  $\gamma^{(i)} \leq \epsilon$  then
8:      $k_2 = k_2^{(i)}$  and terminate
9:    $q^{(i)} = \mathbf{S}p^{(i)}$ 
10:   $\alpha^{(i)} = \frac{\gamma^{(i)}}{\|q^{(i)}\|^2}$ 
11:   $k_2^{(i+1)} = k_2^{(i)} + \alpha^{(i)}p^{(i)}$ 
12:   $r^{(i+1)} = r^{(i)} - \alpha^{(i)}q^{(i)}$ 
13:   $s^{(i+1)} = \mathbf{S}r^{(i+1)}$ 
14:   $y^{(i+1)} = \tilde{\mathbf{S}}r^{(i+1)}$ 
15:   $\gamma^{(i+1)} = y^{(i+1)T} s^{(i+1)}$ 
16:   $p^{(i+1)} = \tilde{\mathbf{S}}s^{(i+1)} + \frac{\gamma^{(i+1)}}{\gamma^{(i)}}p^{(i)}$ 
17:  if  $i < i_{\max}$  then
18:     $i \leftarrow i + 1$  and go to line 7
19:  Compute  $k_1 \leftarrow L_1^{-1}(L_1^{-T}(g_1 - \mathbf{M}_{12}k_2))$  and merge  $k_1$  and  $k_2$  as Katz-vector

```

In the query phase of Katz-based proximity, for a given query node, q . We first construct the identity vector e_q and reorder the entries of e_q using the same ordering of \mathbf{M} . Subsequently, we divide e_q into two parts, $e_q = \begin{bmatrix} e_{q1} \\ e_{q2} \end{bmatrix}$, based on the partition of \mathbf{M} and set $b_1 = e_{q1} - (\mathbf{I} - \alpha\mathcal{G})e_{q1}$ and $b_2 = e_{q2} - (\mathbf{I} - \alpha\mathcal{G})e_{q2}$. Next, we use the indexed matrices and $\tilde{\mathbf{S}}$ to compute k_2 in equation (7), the lower part of solution of the linear system, with Conjugate Gradient method. Here, the $\tilde{\mathbf{S}}$ serves as preconditioner of Conjugate Gradient to refine norms of eigenvectors of \mathbf{S} . Finally, using k_2 and the indexed matrices, we compute k_1 , the upper part of solution of the linear system. We then merge the entries

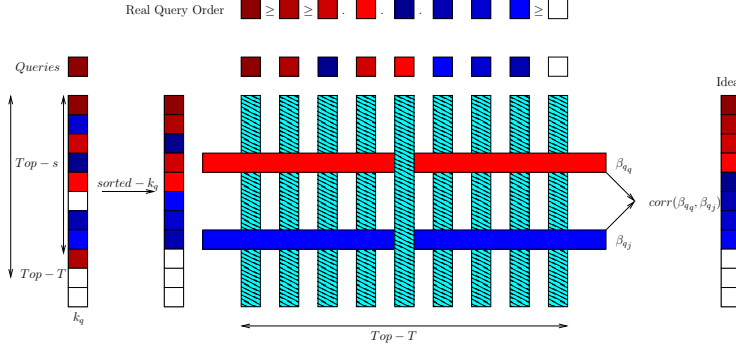


Fig. 2: **Flowchart illustrating the proposed framework for SPARSE-KATZ.** In the uppermost part of the Figure shows the assumed color strength, the darkest red is the strongest. From left to right, we start a query node q and compute its Katz-based scores as k_q and sort them. Assume that $s = 8$ and $T = 9$, For T nodes, we compute Katz-based scores of them and store as cyan matrix. Then, for row q and j , we have β_{qq} and β_{qj} 9×1 vectors. To measure the global closeness of j to q , we take correlation of β_{qq} and β_{qj} and approximate to the rightmost vector.

in k_1 and k_2 and return the resulting merged vector as Katz-based network proximity vector.

3.3 Efficient Processing of Link Prediction Queries via Katz Proximity

Link prediction can be defined as the problem of predicting the links that are likely to emerge/dissappear in the future, given the current state of the network [16]. Various topological measures were broadly examined by Liben et al. [16], as unsupervised link prediction features. One can classify these measures into two categories: Neighborhood-based measures (local) and path-based measures (global). Clearly, The former are cheaper to compute, however, such local measures treat the network as a “bag of interactions rather than a true network, since they do not take into account the potential flow of information across the network through indirect paths, yet the latter (global) are more effective at link prediction since they account for the flow of the information through the indirect paths [5, 8].

In the context of disease gene prioritization, as an application of link prediction problem, global measures are also shown to be significantly more effective than local measures [17]. However, these global measures are favor high-degree nodes over nodes with lower degree [10]. To alleviate this problem, Erten et al. [11] proposed Pearson Correlation based global method that assess the closeness of nodes in the network by comparing the views of the network from the perspective of nodes. More precisely, the authors’ approach was to compute all proximity vectors and store them as a matrix for all nodes, as *Katz*

matrix \mathbf{K} in equation (2), and take the row-wise Pearson Correlation as a new global proximity measure [11]. Later, it has been shown that Pearson Correlation based approach's performance [11] for the link prediction problem can adversely affected by high dimensionality, sparsity of the proximity vector [8]. To alleviate this high dimensionality problem, Coskun et al. [8] proposed two effective dimensionality reduction techniques. However, these techniques requires to compute all proximity matrix and store it in the memory. Clearly, we neither want to compute the *Katz matrix*, \mathbf{K} nor hold it in the memory.

Here in this paper, we propose an alternative algorithm, called SPARSE-KATZ for Katz-based proximity measure that takes sparsity into account without forming the *Katz matrix*. The idea behind SPARSE-KATZ is as follows: For a given query node q and an integer s , we compute Katz-based proximity, k_q via LRC-KATZ and sort the scores of k_q . Afterward, we take Top- s nodes corresponding to the highest s scores in k_q . Then, we compute Katz-based proximity of top- T nodes corresponding top- T the highest scores in k_q . This way, we aim at treating those top- T nodes as modular nodes that see the query node as an important nodes in whole network. This approach enables us to approximate the dimensionality reduction approaches introduced by [8] without forming \mathbf{K} matrix. The workflow of the proposed framework is shown in Figure 2.

Algorithm 3 SPARSE-KATZ

```

1: procedure SPARSE-KATZ
2:   Given Query  $q$ , positive integer  $s$  and  $T$ 
3:   Compute  $k_q$  with LRC-KATZ
4:   Sort  $k_q$  in descending and take Top –  $s$  nodes
5:   for  $t = 1 : T$  do
6:     Compute  $k_t$  with LRC-KATZ
7:     Store  $k_{ts}$  as matrix
8:   Take row-wise Pearson Correlation of  $K_T$  matrix and form Top –  $T$  list
9:   Change Top –  $s$  list according the order of Top –  $T$  list

```

4 Experimental Results

In this section, we systematically first evaluate the runtime performance the proposed algorithm, LRC-KATZ in processing Katz-based proximity queries. As stated in the preceding section, LRC-KATZ is “exact” in the sense that it is guaranteed to correctly identify Katz scores of a given query node. For this reason, we focus on computational cost (measured in terms of number of iterations and runtime) in our experiments for Katz-based proximity and compare LRC-KATZ against another exact algorithm instead of top- k based algorithms [5]. We do not report the preprocessing time since it takes less than a few minutes even for the largest dataset.

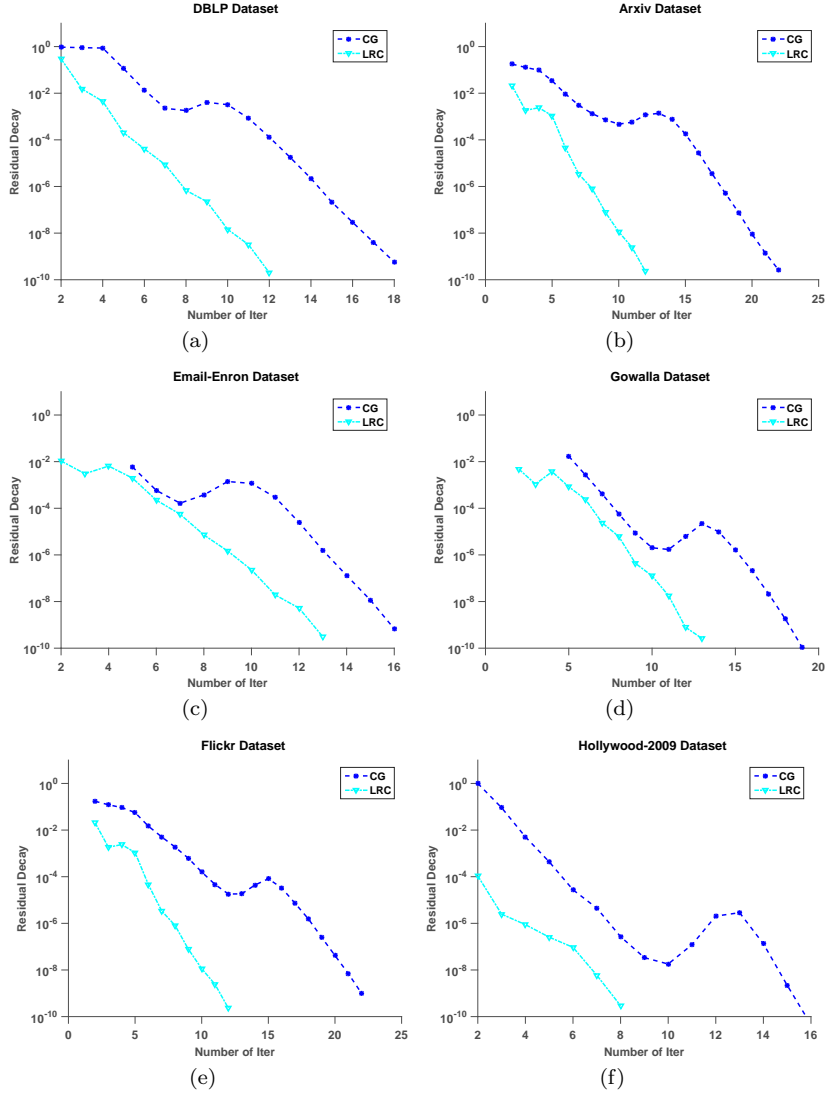


Fig. 3: **The number of iterations required for LRC-KATZ and CG in computing Katz proximity scores** In these experiments, the reported numbers are the averages across 1000 randomly chosen query nodes.

We then evaluate the link prediction performance of SPARSE-KATZ and compare it against Katz measure performance for link prediction.

We start our discussion by describing the datasets and the experimental setup for Katz-based proximity. Subsequently, we compare the performance of LRC-KATZ in processing Katz-based network proximity queries with the

Table 1: Network data sets used in the experiments

| Network | Number of Nodes | Number of Edges | Average Node Degree | $\ G\ _2$ |
|----------------|-----------------|-----------------|---------------------|-----------|
| DBLP_lcc | 93,156 | 178,145 | 3.82 | 39.5753 |
| Arxiv_lcc | 86,376 | 517,563 | 11.98 | 99.3319 |
| Email-Enron | 36,692 | 183,831 | 10.02 | 111.2871 |
| Gowalla | 196,591 | 950,327 | 9.67 | 169.3612 |
| Flickr | 513,969 | 3,190,452 | 12.41 | 663.3587 |
| Hollywood-2009 | 1,139,905 | 113,891,327 | 99.13 | 2247.5591 |
| PPI_Data | 12,976 | 99,814 | 7.6916 | 94.4121 |
| DBLP_2006-2008 | 179,266 | 765,346 | 4.2693 | 61.7765 |

fastest state-of-the-art algorithm, Conjugate Gradient method [20]. Finally, we end our discussion by describing datasets for link prediction application of Katz-based proximity and comparing performance of our link prediction algorithm SPARSE-KATZ against Katz score based link prediction algorithm.

4.1 Datasets and Experimental Setup for Katz-based Proximity

We use six publicly available real-world network datasets and two graphs we collected ourselves. We provide descriptive statistics of these eight networks in Table 1. The first six real-world networks, in our Katz-based proximity experiments. **DBLP_lcc** and **Arxiv_lcc** citation-based networks based on publications databases, and **Flickr** is social network, are publicly provided by [5]. **Email-Enron** email connection network at Enron and **Gowalla** local social communication network come from SNAP collection [15]. Beside these datasets, we lastly use publicly available **Hollywood-2009** [4] Hollywood movie actor network for Katz-based network proximity experiments. The last two real-world network in Table 1 refers the link prediction experiment datasets which collected by ourselves. We give detailed description of these two link prediction datasets in the subsequent sections.

In our implementation phase, for CG algorithm, we use the Matlab implementation downloaded from [5]. We implement LRC-KATZ and SPARSE-KATZ in Matlab. We assess the performance of the algorithms for a fixed damping factor, i.e for each dataset we use the α that defined as the hardest case $\alpha = \frac{1}{\|G\|_2 + 1}$ proximity queries [5]. In practice, using such small α is recommended to fully utilize the information provided by the network [6]. For all experiments to assess the Katz proximity for the first six datasets, we randomly select 1000 query nodes and report the average of the performance figures for these 1000 queries. In all Katz-based proximity experiments, e_q is set to the identity vector for node q . All of the experiments are performed on an Intel(R) Xeon(R) CPU E5-46200 2.20 GHz server with 500 GB memory.

Table 2: Positive Test Distribution of Link Prediction Datasets

| Network | # of Edges [1, 3] | # of Edges [4, 10] | # of Edges > 10 |
|----------------|-------------------|--------------------|-----------------|
| DBLP_2006-2008 | 194,474 | 97,237 | 16,873 |
| PPI_Data | 28,490 | 7,355 | 3,649 |

4.2 Runtime Performance for Katz-based Proximity

The performance of LRC-KATZ in comparison to Conjugate Gradient (CG) algorithm as a function of k , which is dimension of low rank eigenvectors, i.e number of top eigenvectors corresponding the top eigenvalues of R matrix. We provide our analysis to six publicly available networks given in [4, 5, 15]. We first evaluate the performance of LRC-KATZ against CG in terms of number of iteration for a fixed dimension $k = 5$ for randomly chosen 1000 query nodes. Resulting analysis is reported in Figure 3 as an average iteration number. As it can be seen LRC-KATZ significantly outperforms CG across all datasets. We then assess the runtime performance of LRC-KATZ in second with varying k , low rank dimensions and compare it against CG. As seen in the Figure 4, LRC-KATZ outperforms CG, achieving more than 3-fold speed-up for all networks. The performance of LRC-KATZ improves as we increase the dimension, however, due to memory requirements of eigenvectors, we do not go beyond very first eigenvectors corresponding the top eigenvalues.

4.3 Datasets and Experimental Setup for Link Prediction

We test and compare the proposed algorithm SPARSE-KATZ on two comprehensive datasets: one of which is real-world collaboration networks extracted from DBLP Computer Science Bibliography¹, and the other dataset consists of human protein-to-protein interaction (PPI) data obtained from the IntAct database [18].

In the DBLP dataset DBLP_2006-2008, for training data, we consider authors who have published papers between 2006 and 2008. In this network, the authors are represented by nodes and there is a undirected link if two authors published at least one paper from 2006 to 2008 and, as test data, we use new co-author links that emerge between 2009 and 2010. In PPI dataset PPI_Data, for training data, we use proteins whose interactions are known in 2014, as test data, we use the proteins whose interactions are discovered in 2016, however, they have not been known in 2014. Here, proteins are used as nodes and interactions among them represent the edges. These datasets' descriptive static are provided in the last two rows of Table 1.

The objective of link prediction is to predict links that will emerge in the network in the future. For this reason, a positive label in this setup refers to a new link that emerges in the future version of a network, whereas a negative

¹ <http://www.informatik.uni-trier.de/ley/db/>

label refers to two nodes that remain unconnected in the future version. Since the real-world networks are highly sparse, the number of negative pairs is much larger than the number of positive pairs. Hence, predicting negative labels are relatively straightforward. Instead, the remaining challenge is to predict positive labels. Therefore, in our experiment, we only focus on predicting positive labels correctly and report a *recall* evaluation rather than considering positive and negative labels together and reporting a *recall-precision* evaluation. For this reason, we divide the positive labels into three categories to evaluate them. Namely, for DBLP data set, let \mathbf{W} denote the set of authors who published at least one paper in the testing interval [2009, 2010], but have not published together in the training interval ([2006, 2008]). We construct our positive node pairs from this set as follows:

- The positive test set \mathcal{P} is composed of $u, v \in \mathbf{W}$ such that u and v published a paper between 2009 and 2010.
- For DBLP network, \mathcal{P} consists of 308584 added edges in between 2009 and 2010, i.e., $|\mathcal{P}| = 308584$.
- We divide \mathcal{P} into three categories as in summarized Table 2.
- For all experiments for DBLP_2006–2008 dataset, we report the mean and the standard deviation of the performance figures

With similar logic above, we subsampled the $\mathcal{P} = 39494$ sets in PPI network, PPI_Data, from 2016. In our all experiments, we use default hardest α values depending on $\|G\|_2$ values of DBLP_2006–2008 and PPI_Data.

4.4 Link Prediction Performance for SPARSE-KATZ

The performance of SPARSE-KATZ in comparison to Katz-based algorithm’s link prediction application as a function of s , which is the top- s most probable edges that are computed by both algorithms, Katz and SPARSE-KATZ. We provide our analysis to two networks given in the last two rows of Table 1. We first evaluate the performance of SPARSE-KATZ against Katz in terms of number of *recall* for all the positive set added to DBLP_2006–2008 in between 2009 and 2010. Resulting analysis is reported in Figure 5 as mean and standard deviation of the *recall* values. As it can be seen SPARSE-KATZ significantly outperforms Katz proximity based link prediction across all sub-sampled links. We then assess the link prediction performance of SPARSE-KATZ for PPI_Data dataset and compare it against Katz measure. As seen in the Figure 6, SPARSE-KATZ outperforms Katz measure drastically.

5 Conclusion

In this paper, we propose an alternate approach to accelerating Katz- based network proximity queries. The proposed approach is based on low rank correction of underlying partitioned linear systems of equation derived from Katz

matrix. We show that our approach, LRC-KATZ, significantly decreases convergence times in practice on real-world problems. Using a number of large real-world networks, we show that LRC-KATZ outperforms existing the fastest known method, Conjugate Gradient, significantly for wide ranges of parameter values. When integrated with the fact that Katz scores distributes sparsely in the Katz vector, Chopper yields further improvement in performance of link prediction task over state-of-the-art Katz measure. Future efforts in this direction would include incorporation of other proximity measures into our framework and their applications. Furthermore, while LRC-KATZ is an “exact algorithm and our experiments focus on runtime performance for this reason, there also exist approximate methods that compromise accuracy for improved runtime. Constructing an approximate version of LRC-KATZ can provide further insights into the trade-off between runtime and accuracy in the context of network proximity problems.

References

1. Acar, E., Dunlavy, D.M., Kolda, T.G.: Link prediction on evolving data using matrix and tensor factorizations. In: Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on, pp. 262–269. IEEE (2009)
2. Amestoy, P.R., Davis, T.A., Duff, I.S.: An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications* **17**(4), 886–905 (1996)
3. Amestoy, P.R., Davis, T.A., Duff, I.S.: An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications* **17**(4), 886–905 (1996)
4. Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In: Proceedings of the 20th international conference on World wide web, pp. 587–596. ACM (2011)
5. Bonchi, F., Esfandiari, P., Gleich, D.F., Greif, C., Lakshmanan, L.V.: Fast matrix computations for pairwise and columnwise commute times and katz scores. *Internet Mathematics* **8**(1-2), 73–112 (2012)
6. Coşkun, M., Grama, A., Koyutürk, M.: Indexed fast network proximity querying. *Proc. VLDB Endow.* **11**(8), 840–852 (2018). DOI 10.14778/3204028.3204029. URL <https://doi.org/10.14778/3204028.3204029>
7. Coskun, M., Grama, A., Koyuturk, M.: Efficient processing of network proximity queries via chebyshev acceleration. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1515–1524. ACM (2016)
8. Coskun, M., Koyutürk, M.: Link prediction in large networks by comparing the global view of nodes in the network. In: Data Mining Workshop (ICDMW), 2015 IEEE International Conference on, pp. 485–492. IEEE (2015)
9. Demmel, J.W.: Applied numerical linear algebra, vol. 56. Siam (1997)
10. Erten, S., Bebek, G., Ewing, R.M., Koyutürk, M.: Dada: degree-aware algorithms for network-based disease gene prioritization. *BioData mining* **4**(1), 19 (2011)
11. Erten, S., Bebek, G., Koyutürk, M.: Vavien: an algorithm for prioritizing candidate disease genes based on topological similarity of proteins in interaction networks. *Journal of computational biology* **18**(11), 1561–1574 (2011)
12. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* **20**(1), 359–392 (1998)
13. Karypis, G., Kumar, V.: A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing* **48**(1), 71–95 (1998)
14. Katz, L.: A new status index derived from sociometric analysis. *Psychometrika* **18**(1), 39–43 (1953)

15. Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. In: Proceedings of the SIGCHI conference on human factors in computing systems, pp. 1361–1370. ACM (2010)
16. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. *Journal of the American society for information science and technology* **58**(7), 1019–1031 (2007)
17. Navlakha, S., Kingsford, C.: The power of protein interaction networks for associating genes with diseases. *Bioinformatics* **26**(8), 1057–1063 (2010)
18. Orchard, S., Ammari, M., Aranda, B., Breuza, L., Briganti, L., Broackes-Carter, F., Campbell, N.H., Chavali, G., Chen, C., Del-Toro, N., et al.: The mintact project: intact as a common curation platform for 11 molecular interaction databases. *Nucleic acids research* p. gkt1115 (2013)
19. Rattigan, M.J., Jensen, D.: The case for anomalous link discovery. *Acm Sigkdd Explorations Newsletter* **7**(2), 41–47 (2005)
20. Saad, Y.: Iterative methods for sparse linear systems, vol. 82. *siam* (2003)
21. Saerens, M., Fous, F., Yen, L., Dupont, P.: The principal components analysis of a graph, and its relationships to spectral clustering. In: *European Conference on Machine Learning*, pp. 371–383. Springer (2004)
22. Sarkar, P., Moore, A.: A tractable approach to finding closest truncated-commute-time neighbors in large graphs. *arXiv preprint arXiv:1206.5259* (2012)
23. Sarkar, P., Moore, A.: A tractable approach to finding closest truncated-commute-time neighbors in large graphs. *arXiv preprint arXiv:1206.5259* (2012)
24. Wang, C., Satuluri, V., Parthasarathy, S.: Local probabilistic models for link prediction. In: *icdm*, pp. 322–331. IEEE (2007)

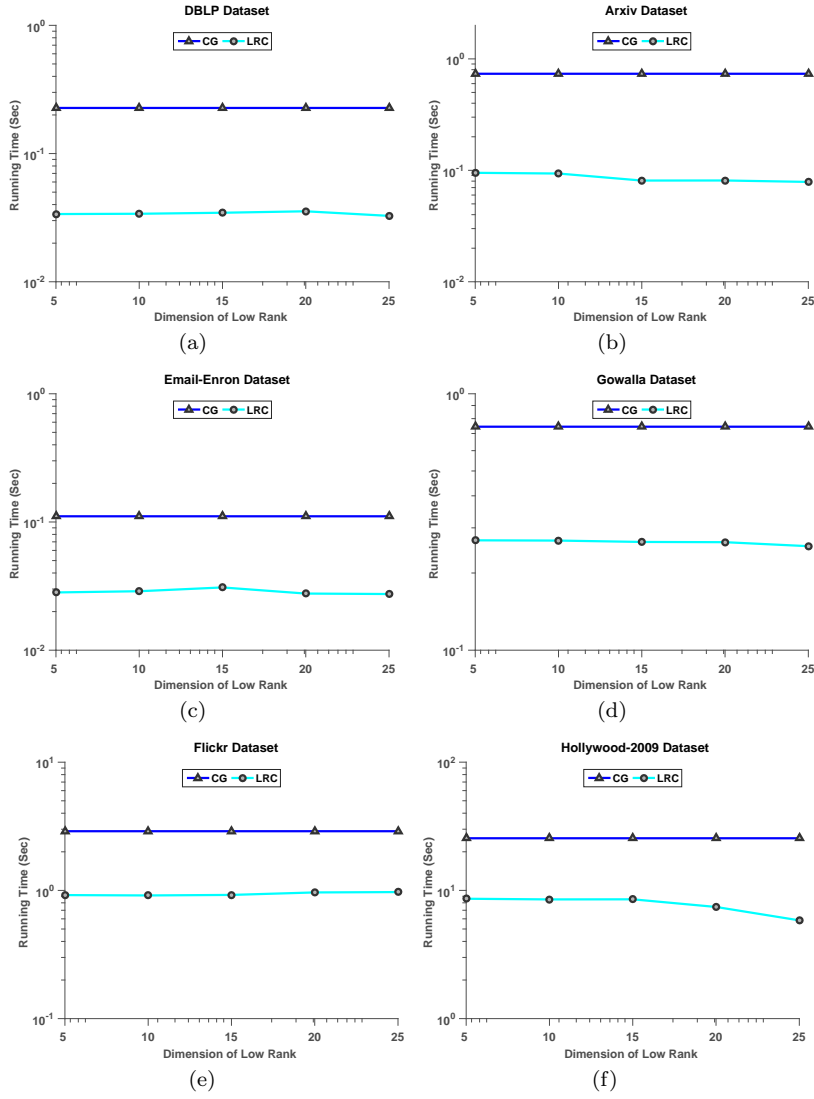


Fig. 4: Runtime in seconds of LRC-KATZ and CG to process queries for Katz proximity as a function of k ranging from 5 to 25. In these experiments the reported numbers are the averages across 1000 randomly chosen query nodes.

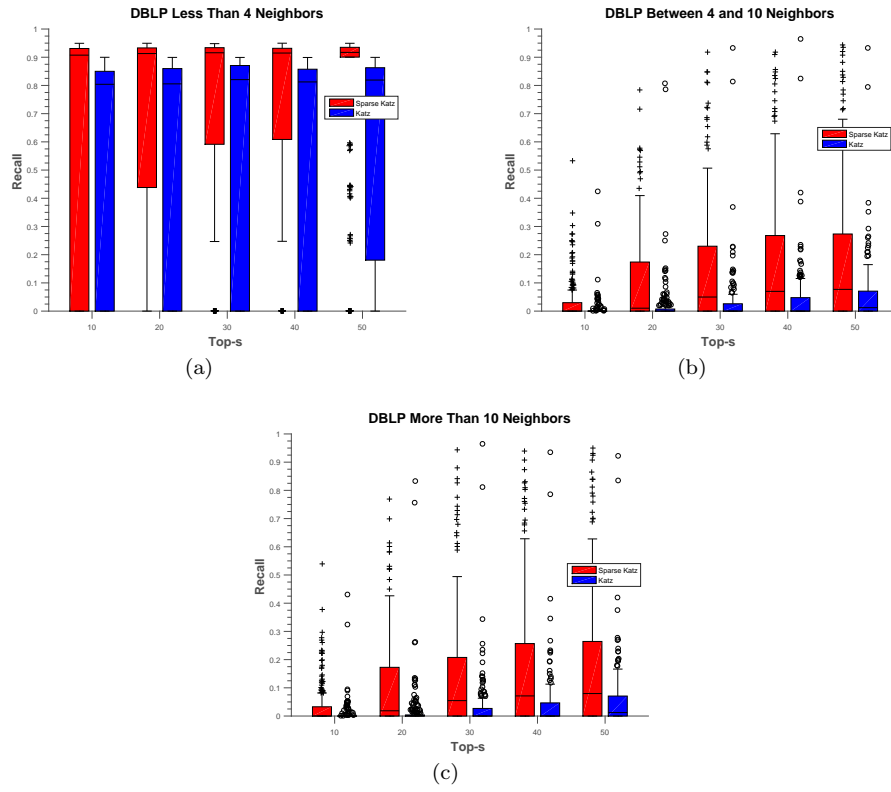


Fig. 5: **Performance evaluation of SPARSE-KATZ for DBLP_2006-2008** The performance of SPARSE-KATZ link prediction as compared to Katz measure based link prediction on the DBLP_2006-2008

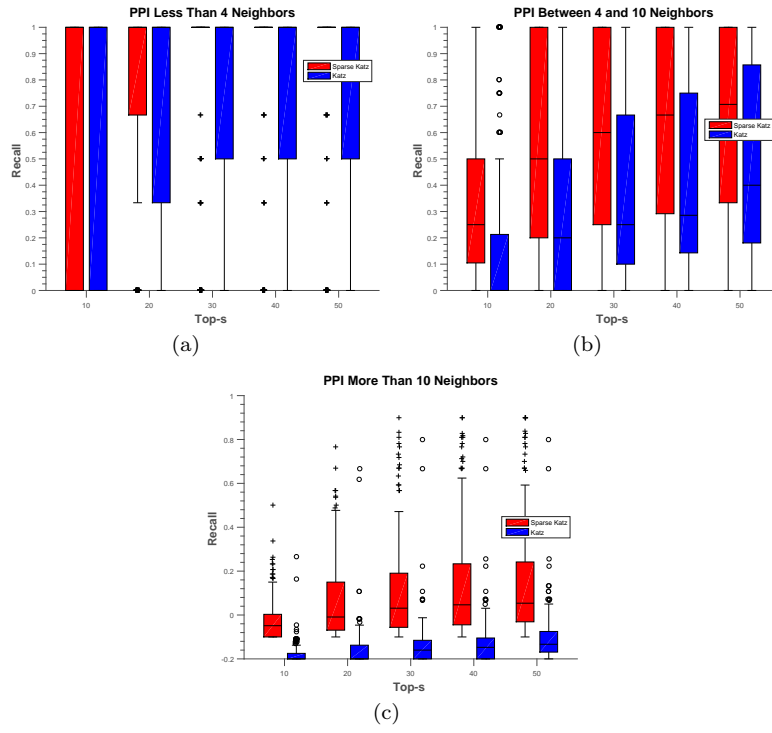


Fig. 6: **Performance evaluation of SPARSE-KATZ for PPI_Data** The performance of SPARSE-KATZ link prediction as compared to Katz measure based link prediction on the PPI_Data