Noname manuscript No.
(will be inserted by the editor)

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. Taking advantage of the acceleration in the computation of Katz index,

Mustafa Coşkun
Department of Computer Engineering
Abdullah Gül University
Kayseri,Turkey
Tel.: +90-505-0082739
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 Computer and Data Sciences
Case Western Reserve University
Cleveland, USA
Tel.: +1-216-3682963
E-mail: mehmet.koyuturk@case.edu

we propose a new link prediction algorithm that exploits locality of networks that are encountered in practical applications. Our experiments show that the resulting link prediction algorithm drastically outperforms state-of-the-art link prediction methods based on the vanilla and truncated Katz.

Keywords Fast Katz Method · Link Prediction · Network Proximity

1 Introduction

Proximity computation 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 (Liben-Nowell and Kleinberg 2007; Sarkar and Moore 2012*a*). In information retrieval, anomalous links are ranked based on the nodes' proximity to other nodes (Rattigan and Jensen 2005). In unsupervised learning, network proximity is used to quantify cluster quality (Saerens et al. 2004).

The general setting for network proximity queries is as follows: given a query node, we aim to compute a score for all other nodes in the network, based on their proximity to the query node. These measures of network distance include shortest path (minimum number of edges between two nodes), random walk with restarts, effective resistance, commute distance, and Katz-index. All measures except shortest path aim to quantify the multiplicity of relatively short paths connecting the two nodes. Katz-index accomplishes this directly, where the proximity between two nodes is defined as a weighted sum of the number of paths connecting the two nodes. The weighting is applied to assign more importance to shorter paths (Bonchi et al. 2012; Katz 1953). Katz-based proximity measures have been used in a number of applications, including link prediction (Liben-Nowell and Kleinberg 2007), clustering (Saerens et al. 2004), and ranking (Rattigan and Jensen 2005).

Motivated by problems such as link prediction and collaborative filtering, significant efforts have been devoted to reducing the computational costs associated with the computation of 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 (Bonchi et al. 2012) 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 (Bonchi et al. 2012). 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 performed efficiently (Saad 2003). 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 (Bonchi et al. 2012; Saad 2003) require a large number of iterations to converge. More concretely, Fast-Katz (Bonchi et al. 2012) offers a tight convergence upper bound, however, Conjugate Gradient (CG) performs better than Fast-Katz (Bonchi et al. 2012) for Katz-based proximity. On the other hand, direct inversion techniques (Saad 2003) 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 subnetworks and inverts the small matrices corresponding to these subnetworks. This partitioning idea can be viewed as non-overlapping domain decomposition (Smith et al. 2004). By inverting the small block diagonal matrices that correspond to small sub-networks, our hybrid procedure overcomes the memory constraint of direct inversion techniques. The denser matrix, composed of the nodes connecting these subnetworks is handled through a low rank corrected iterative CG method. By performing an iterative low rank corrected procedure on this smaller matrix (as compared to the original matrix that corresponds to the entire network), our method overcomes the computational cost considerations of classical CG. Motivated by the skewed distribution of Katz-based proximity scores(Bonchi et al. 2012), we also propose a new link prediction algorithm, called Sparse-Katz that exploits the modularity of networks encountered in practical applications.

We provide detailed theoretical justifications for our results and experimentally show the superior performance of our method on a number of real-world networks. Our experiments on these networks show that the resulting hybrid approach converges much faster than classical CG, and significantly accelerates the computation of Katz-based proximity queries for very large graphs. 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. Our experimental results on the link prediction problem show that our modularity based algorithm significantly outperforms state-of-the-art link prediction Vanilla and Truncated Katz methods.

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 a link prediction algorithm that renders Katz-based link prediction 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. In Section 5, we discuss avenues for future research. We conclude our discussion in Section 6.

2 Related Work

Node proximity queries have received significant research attention in recent years in various areas of data mining, such as searching, ranking, clustering and analyzing network structured object similarity (Coşkun et al. 2018). In particular, Katz-based node proximity queries in large graphs have been well studied (Bonchi et al. 2012).

Efficient Computation of Katz-based Proximity. One of the commonly used approaches for computing Katz-based proximity is the power method through the Neumann series expansion of the underlying linear systems of equations (Saad 2003). An alternate approach to power iterations is to use of offline computation, which directly inverts the underlying linear system of equations, typically using Cholesky factorization or eigen-decomposition (Acar et al. 2009; Sarkar and Moore 2012*b*; Wang et al. 2007). 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.

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 by exploiting a relationship between the Lanczos process and a quadrature rule in the iterative computation (Bonchi et al. 2012). However, these local search based methods for Katz proximity computation are not as efficient as CG method (Bonchi et al. 2012).

Relation to Domain Decomposition. In this paper, we focus on exact computation of Katz proximity in very large networks using non-overlapping domain decomposition (DD) (Smith et al. 2004) accelerated by low-rank correction. Domain Decomposition techniques efficiently solve (non)-linear systems of equations derived from Partial Differential Equations (PDE) using a divide-and-conquer approach (Saad 2003; Skogent 1992; Smith et al. 2004; Van der Vorst and Chan 1997). However, the rich literature on DD has not been fully exploited by research efforts in data mining and machine learning, where many graph learning related problems, including link prediction, semi-

supervised learning (Zhou et al. 2004), and graph convolutional networks (Kipf and Welling 2016) require solution to large linear systems of equations that can efficiently be solved via DD approaches. In the context of computing Katz-based proximity, our approach uses DD for partitioning the underlying graph and further accelerates the DD preconditioners by solving the dense part of the system via a low rank correction. Hence, our method is fundamentally different from existing approaches in that it simultaneously targets scalability and efficiency.

Application to Link Prediction. Link prediction can be defined as the problem of predicting the links that are likely to emerge/disappear in the future, given the current snapshot of the network. Various topological measures were extensively examined for the link prediction problem (Liben-Nowell and Kleinberg 2007). These measures can be classified into two categories: neighborhood-based measures (local) and path-based measures (global). Clearly, methods that are based on local measures, such as Common Neighbor and Adamic-Adar (Liben-Nowell and Kleinberg 2007) are more efficient than those that are based on global measures, such as PageRank (Page et al. 1999) and Katz-index (Katz 1953). However, the global measures are more effective than local measures for the link prediction problem since they account for the flow of the information through the indirect paths (Bonchi et al. 2012; Coskun and Koyutürk 2015) whereas the local methods focus only on the local neighborhood of the nodes in the network.

In the context of candidate disease gene prioritization, a common application of link prediction in computational biology, global measures are also shown to be significantly more effective than local measures (Navlakha and Kingsford 2010). However, these global measures were shown to favor high-degree genes over the genes that are relatively less connected or less studied (Erten, Bebek, Ewing and Koyutürk 2011). To alleviate this problem, Erten et. al., (Erten, Bebek and Koyutürk 2011) proposed a topological similarity-based global method that assesses the similarity of two nodes in a network using the correlation of their random-walk based proximities to all other nodes in the network (Erten, Bebek and Koyutürk 2011). Observing that the computation of topological similarity can be adversely affected by high-dimensionality in link prediction applications on social networks, we proposed a simple dimensionality reduction technique (Coskun and Koyutürk 2015). Here, we develop a link prediction algorithm, Sparse-Katz that uses the similarity between the Katz-based proximity profiles of nodes to assess the topological similarity between the nodes. In comparison to existing algorithms, the key contribution of Sparse-Katz is that it computes the proximity vectors used in the assessment of topological similarity at query time. This feature enables Sparse-Katz to personalize the dimensions of the proximity vectors based on the query node, which is facilitated by the improvement in the efficiency of computing Katz-based proximity scores provided by LRC-Katz.

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 a domain decomposition technique, 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 (Bonchi et al. 2012; Katz 1953). Namely, for a pair of nodes i and $j \in \mathcal{V}$, the Katz index is defined as:

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

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 (Bonchi et al. 2012).

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:

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

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 symmetric positive definite, and to guarantee the convergence of the Neumann series to the inverse of $(\mathbf{I} - \alpha\mathbf{G})$. Here we highlight that, in this paper, we aim to solve the Katz proximity with the hardest $\alpha = \dfrac{1}{\|\mathbf{G}\|_2 + 1}$ which makes $(\mathbf{I} - \alpha\mathbf{G})$ very close to indefinite matrix (Bonchi et al. 2012).

For a given query node q, the Katz index of every other node in $\mathcal{G}$ with respect to q is given by the qth 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:

$$\mathbf{M}\mathbf{k}_q = \alpha\mathbf{g}_q. \tag{3}$$

Here, $\mathbf{M} = (\mathbf{I} - \alpha\mathbf{G})$ and $\mathbf{g}_q$ is the qth 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 (Saad 2003, p. 196), which is applicable in this case since $\mathbf{M}$ is symmetric (Bonchi et al. 2012). 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 (Coşkun et al. 2018). 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](Saad 2003).

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*

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

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:*

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

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

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 (Coşkun et al. 2018). In the computation of Katz-index, $\mathbf{S}$ is symmetric [p.271](Saad 2003), thus the elliptic plane degrades to the real axis and the solution can be found by using Chebyshev polynomials on the real axes (Coskun et al. 2016). 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](Saad 2003). 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 (Coşkun et al. 2018), 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 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" (Karypis and Kumar 1998*a*,*b*). Given such a partitioning, we reorder the matrix $\mathbf{M}$ such that the rows/columns that correspond to nodes in the partitions $\Pi_1, \Pi_2, ..., \Pi_p$ 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 partioning 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 (Karypis and Kumar 1998*a*) 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 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 (Coşkun et al. 2018). In our implementation, we use the Incomplete Cholesky factorization along with approximate minimum algorithms (Amestoy et al. 1996*a*,*b*) 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.

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

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

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

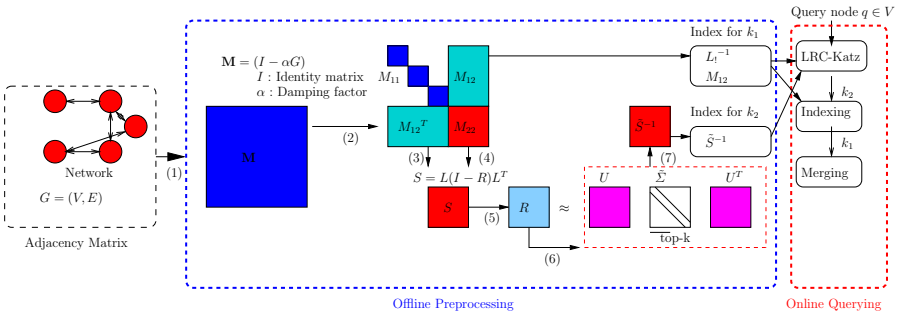


Fig. 1: **Flowchart illustrating the proposed framework for indexed processing of Katz-based proximity queries on large networks.**

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

$$\mathbf{S} = \mathbf{L}\mathbf{L}^T - \mathbf{M}_{12}^T\mathbf{M}_{11}^{-1}\mathbf{M}_{12} \tag{8}$$
$$= \mathbf{L}(\mathbf{I} - \mathbf{L}^{-1}\mathbf{M}_{12}^T\mathbf{M}_{11}^{-1}\mathbf{M}_{12}\mathbf{L}^{-T})\mathbf{L}^T \tag{9}$$
$$= \mathbf{L}(\mathbf{I} - \mathbf{R})\mathbf{L}^T \tag{10}$$

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

$$\mathbf{R} = \mathbf{L}^{-1}\mathbf{M}_{12}^T\mathbf{M}_{11}^{-1}\mathbf{M}_{12}\mathbf{L}^{-T} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T, \tag{11}$$

where the diagonal entries of $\boldsymbol{\Sigma}$ 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:

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

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

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

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

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

$$= \underbrace{\begin{pmatrix} u_1 & \cdots & u_l & u_{l+1} & \cdots & u_{n_2} \\ | & & | & | & & | \\ | & & | & | & & | \end{pmatrix}}_{\ell \quad 0} \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_l & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix} \begin{pmatrix} - & u_1^T & - \\ & & \\ - & u_l^T & - \\ - & u_{l+1}^T & - \\ & & \\ - & u_{n_2}^T & - \end{pmatrix} \begin{matrix} \} \ell \\ \\ \\ \} 0 \\ \\ \end{matrix}$$

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

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

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 $\mathbf{S}\tilde{\mathbf{S}}^{-1}$ and the eigenvalues of $\mathbf{R}$.

Theorem 1 *Assume that the eigenvalues of* $\mathbf{R}$ *are ordered as* $\sigma_1 \geq \sigma_2 \geq .... \geq \sigma_{n_2}$*, 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* $\mathbf{S}\tilde{\mathbf{S}}^{-1}$ *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

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

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

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

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

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

Finally, we have,

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

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 preproccessing 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($\mathbf{G}, \alpha, k$)
2: Construct $\mathbf{M} \leftarrow (\mathbf{I} - \alpha\mathbf{G})$
3: Use minimum-vertex-seperator graph partitioning on $\mathbf{M}$ to partition it into $\mathbf{M}_{11}, \mathbf{M}_{12}^T, \mathbf{M}_{12}, \mathbf{M}_{22}$ (Karypis and Kumar 1998*a*,*b*)
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 $\left[\tilde{\mathbf{U}}, \tilde{\mathbf{\Sigma}}\right] = Lanczos(\mathbf{R}\mathbf{l}^{(0)}, k, \mathbf{l}^{(0)})$ (Demmel 1997) ▷ 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 (Karypis and Kumar 1998*a*,*b*). Next, we reorder the entries of partitioned matrix $\mathbf{M}$

based on an approximate minimum degree ordering (AMD) (Amestoy et al. 1996*a*,*b*). 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 $\tilde{\mathbf{\Sigma}}$ via *Lanczos* procedure (Demmel 1997) 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, $b2$, 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 $i = 0$, $r^{(i)} = f - \mathbf{S}{k_2}^{(k)}$, $s = \mathbf{S}r^{(i)}$, $p = \tilde{\mathbf{S}} \backslash s^{(i)}$, $y^{(i)} = \tilde{\mathbf{S}} \backslash r^{(i)}$ ▷ $\mathbf{S}$ and $\tilde{\mathbf{S}}$ are used as matrix-vector product
6: $\gamma^{(i)} = {y^{(i)}}^{\mathrm{T}} s^{(i)}$
7: **if** $\gamma^{(k)} \leq \epsilon$ **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: $s^{(i+1)} = \mathbf{S}^{(i+1)}$
14: $y^{(i+1)} = \tilde{\mathbf{S}} \backslash r^{(i+1)}$
15: $\gamma^{(i+1)} = {y^{(i+1)}}^{\mathrm{T}} s^{(i+1)}$
16: $p^{(i+1)} = \tilde{\mathbf{S}} \backslash s^{(i+1)} + \frac{\gamma^{(i+1)}}{\gamma^{(i)}} p^{(i)}$
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, $e_q = \begin{bmatrix} e_{q_1} \\ e_{q_2} \end{bmatrix}$, based on the partition of $\mathbf{M}$ and set $b_1 = e_{q_1} - (\mathbf{I} - \alpha\mathcal{G})e_{q_1}$ and $b_2 = e_{q_2} - (\mathbf{I} - \alpha\mathcal{G})e_{q_2}$. 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 preconfitioner 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 in k_1 and k_2 and return the resulting merged vectoras Katz-based network proximity vector.

3.3 Efficient Processing of Link Prediction Queries via Katz Proximity

Observing that our hybrid algorithm enables efficient computation of Katz-based proximity at query time, we develop an algorithm that uses Katz-based proximity to efficiently and effectively process link prediction queries. Instead of using Katz-based proximity directly for link prediction, our algorithm, Sparse-Katz uses the similarity of Katz-based proximity vectors to assess the likelihood that two nodes will gain a edge. While doing so, Sparse-Katz takes the sparsity of the network into account and reduces dimensionality directly while assessing the topological similarity of the nodes.

The setting for Sparse-Katz is as follows: A query is formulated by specifying a query node q and integer s, indicating that the user aims to identify the s nodes that are most likely to gain an edge with node q. A query is also associated with two integer parameters, C and T. The parameter C, where $C > s$ and $C << n$, is used to generate a set of candidate nodes using Katz-based proximity to q. The parameter $T << n$, on the other hand, specifies the dimensionality of the vectors used by Sparse-Katz compute the topological similarity between the nodes.

Given a query, we first compute the Katz-based proximity vector for the query node, $k_q \in \mathbb{R}^{n \times 1}$ using LRC-Katz. We then identify the top T nodes with highest scores in k_q, as the anchor nodes representing the neighborhood of q. Subsequently, as shown in Figure 2, using LRC-Katz again, we compute Katz-based proximity vectors for these T nodes and represent these as an $n \times T$ matrix $\mathbf{K_T}$. Observe that the ith row of $\mathbf{K_T}$ represents the proximity of the ith node to the neighborhood of q. Then, we identify the top C nodes with highest scores in k_q as the candidate nodes that are considered for the link prediction query. We denote the set of these nodes as $\{C\} \subset \mathcal{V}$. The idea behind using Katz-based proximity directly to identify a list of candidate nodes is as follows: While topological similarity is potentially a better indicator of the likelihood of gaining an edge as compared to proximity, nodes that are too far from q are not likely to be topologically similar to q.

Once we have a set C of candidate nodes that are closest to q according to Katz-based proximity, we use Katz-based proximity profiles to assess the topological similarity between these candidate nodes and q. For this purpose, for each node $c \in C$, we compute $\beta_q(c)$ as the correlation between the cth row of $\mathbf{K_T}$ and the qth row of $\mathbf{K_T}$. The resulting vector $\beta \in \mathbb{R}^{|C| \times 1}$ contains the topological similarity scores of the candidate nodes with respect to the query node q. Finally, we identify the top s nodes with highest scores in β_q, and return these nodes as the query result. The pseudo-code for Sparse-Katz is shown in Algorithm 3.

We note that in Sparse-Katz, we need to solve $|T| + 1$ linear systems of equations for **each unique query node** for the link prediction task. These linear systems of equations can be solved using either Richardson iterations and CG or LRC-Katz. In the next section, we empirically show that when we use LRC-Katz to solve these $|T| + 1$ linear system of equations for all

Algorithm 3 SPARSE-KATZ

procedure SPARSE-KATZ
  Given *Query* q, *positive integer* s, C *and* T, where $C > s$
  Compute k_q with LRC-KATZ
  Sort k_q in descending and take $Top - C$ nodes
  Take $Top - T$ nodes that are the closest to the query node, q, in k_q vector
  for $t = 1 : T$ **do**
    `Compute` k_t `with` LRC-KATZ
    `Store` k_t`s as matrix,` K_T
  Take row-wise correlations of C nodes in K_T with respect to the row corresponding to the query node
  Sort the correlation scores as a vector, $\beta_q \in \mathbb{R}^{C \times 1}$
  Return to $Top - s$ nodes in vector β_q

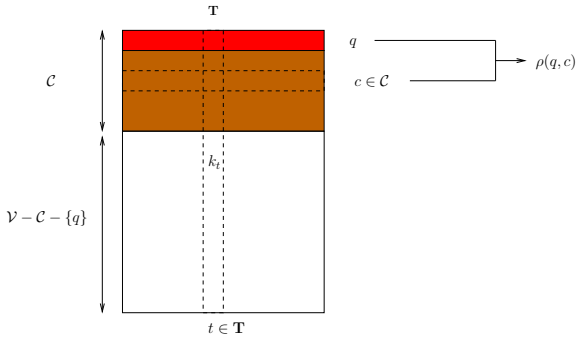


Fig. 2: **Illustration of the** SPARSE-KATZ **algorithm for link prediction. T** denotes the set of nodes that are to assess topological similarity to query node q. **C** denotes the set of candidates for link prediction. The large matrix (including red, brown, and white portions) is the matrix of Katz-based proximity vectors for the nodes in T. The red row shows the topological profile of query node q in the space of defined by the nodes in T. The rows of the brown matrix show the topological profiles of the candidate nodes in T. The topological similarity between a candidate node c and the query node q is computed using the topological profiles of q and c.

query nodes, it drastically improves the computational efficiency of SPARSE-KATZover CG or Richardson iteration.

4 Experimental Results

In this section, we first systematically evaluate the runtime performance the proposed algorithm, LRC-KATZ in processing Katz-based proximity queries. As stated in the previous section, LRC-KATZ is an "exact" algorithm in the sense that it is guaranteed to correctly identify Katz scores of all nodes in the graph for a given query node. For this reason, we focus on computational cost (measured in terms of number of iterations and runtime) in our experiments

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_Data	10,704	49,750	4.65	19.6986

for Katz-based proximity and compare LRC-Katz against another exact algorithm instead of top-k based algorithms (Bonchi et al. 2012). We do not report the pre-processing time since it takes less than a few minutes even for the largest dataset. This pre-processing time is negligible, since all datasets we consider contain more than $10K$ nodes, thus the total runtime of query processing would take much longer if Katz-based proximity queries were processed for all nodes in the network.

We then evaluate the link prediction performance of Sparse-Katz and compare it against vanilla Katz, where computation is performed using CG and Truncated Katz, which is based on Richardson iterations (Van der Vorst and Chan 1997). It is important to note that LRC-Katz is the only algorithm that permits topological similarity based link prediction (Sparse-Katz). This is because, Sparse-Katz requires repeated computation of Katz-based proximity while processing a link prediction query for a single node. Such computation is not feasible using vanilla Katz or truncated Katz. For this reason, while scoring the likelihood of an edge, the link prediction algorithms we implement using vanilla Katz and truncated Katz directly use Katz-based proximity to the query node. Consequently, although both vanilla Katz and LRC-Katz are exact algorithms for computing Katz-based proximity, their accuracy in link prediction can be different. Thus, in addition to efficiency, our comparative studies in the context of link prediction also provide an assessment of the contribution of topological similarity based link prediction in improving the accuracy of link prediction.

4.1 Datasets and Experimental Setup for Katz-based Proximity

We use six publicly available real-world network datasets commonly used in benchmarking proximity computation algorithms. For link prediction, we use two networks, one representing the human protein interaction network and the other representing the citation network obtained from DBLP. The descriptive statistics of these eight networks are shown in Table 1.

The first six real-world networks are used in assessing the runtime performance of algorithms in processing Katz-based proximity queries. `DBLP_lcc` and `Arxiv_lcc` are citation networks based on publications databases, and `Flickr` is a social network, all of which are provided by (Bonchi et al. 2012).

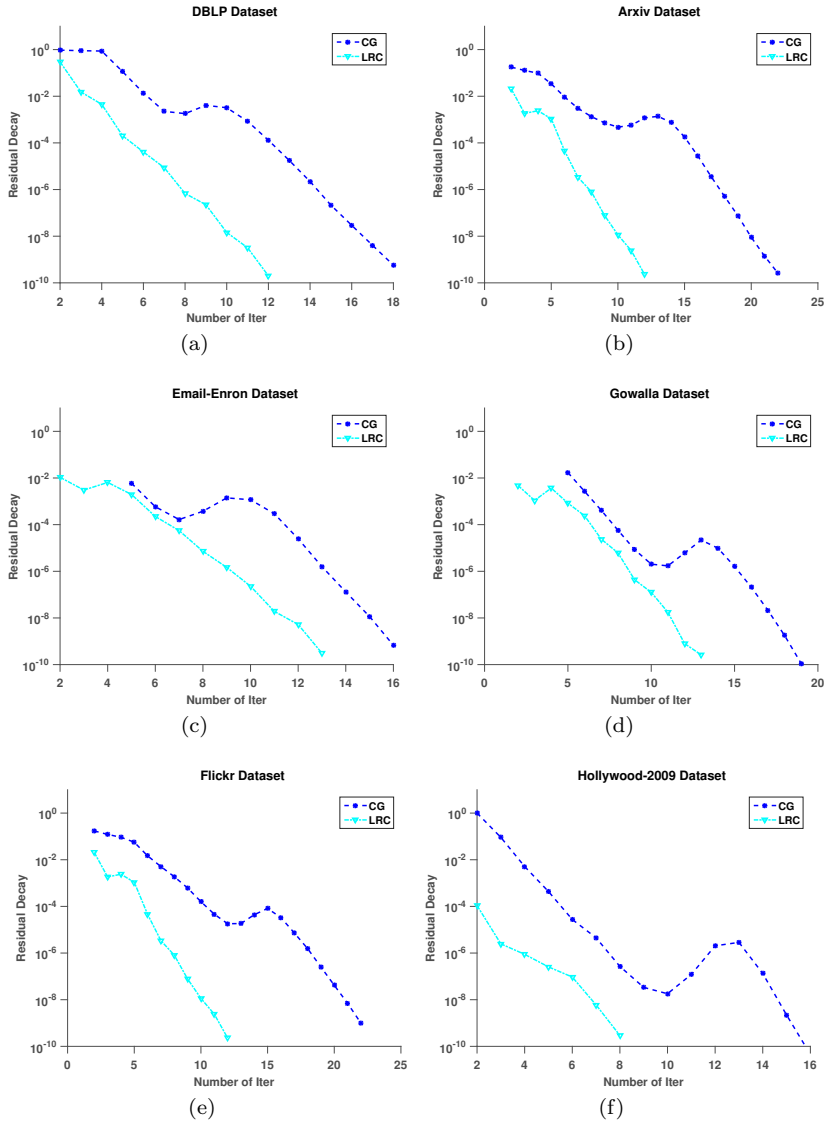


Fig. 3: **The number of iterations required for** LRC-KATZ **and CG in computing Katz proximity scores.** In these experiments, the reported numbers are the averages across 1000 randomly chosen query nodes.

`Email-Enron` is the e-mail commmunication network at Enron and `Gowalla` is a local social communication network, both of which are obtained from the SNAP collection (Leskovec et al. 2010). The last dataset we use is the publicly

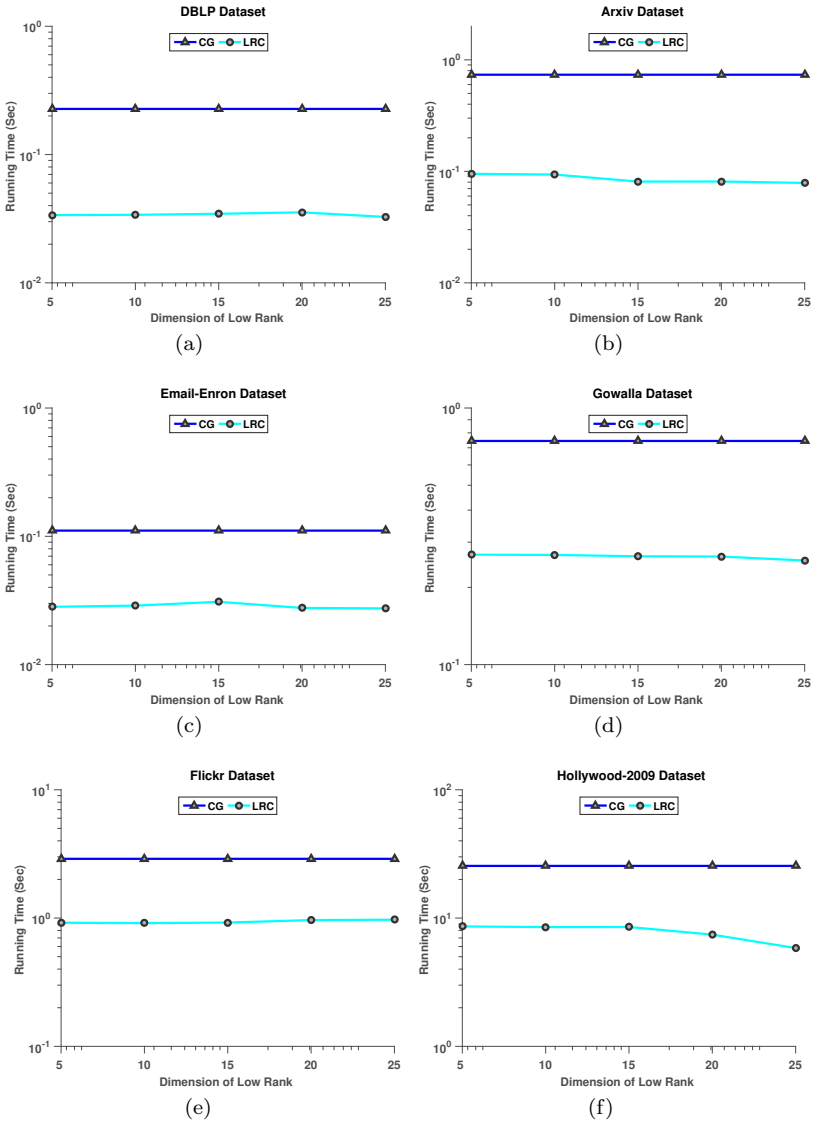


Fig. 4: **Runtime of** LRC-Katz **and CG to process a Katz-based proximity query for a single node as a function of** k **ranging from** 5 **to** 25**.** In these experiments the reported numbers are the averages across 1000 randomly chosen query nodes.

available `Hollywood-2009` (Boldi et al. 2011) Hollywood movie actor network, in which an edge represents acting together in a movie.

For the CG algorithm, we use the Matlab implementation downloaded from (Bonchi et al. 2012). We also 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 αs that is recognized as the hardest $\alpha = \frac{1}{\|\mathbf{G}\|_2 + 1}$ for computing Katz-based proximity (Bonchi et al. 2012). In practice, using such "supremum" α is recommended to fully utilize the information provided by the network (Coşkun et al. 2018). In all experiments, we randomly select 1000 query nodes and report the average of the performance figures for these 1000 queries, in which e_q is set to the identity vector for node q. For the Richardson iterations in truncated Katz, we perform 15 iterations for low degree nodes and 5 iterations for high degree nodes. All of the experiments are performed on an Intel(R) Xeon(R) CPU E5-46200 2.20 GHz server with 500 GB memory.

4.2 Runtime Performance for Katz-based Proximity

The rate of convergence of LRC-Katz in comparison to the Conjugate Gradient algorithm for all six networks is shown in Figure 3. In these experiments, k, the number of dimensions of low rank eigenvectors, is set to 5. As it can be seen LRC-Katz converges significantly faster than CG across all datasets.

We then assess the runtime performance of LRC-Katz as a function of k ranging from 5 to 25. The results of this analysis for all datasets are shown in Figure 4. As seen in the figure, faster convergence of LRC-Katz translates into savings in time, and LRC-Katz achieves more than 3-fold speed-up over CG for all networks. The performance of LRC-Katz improves as we increase the number of dimensions, however, due to the memory requirements of computing eigenvectors, we do not go beyond the first 25 eigenvectors corresponding the top eigenvalues.

4.3 Datasets and Experimental Setup for Link Prediction

We test and compare Sparse-Katz, our topological similarity based link prediction algorithm, on two comprehensive datasets: 1) a real-world collaboration network extracted from DBLP Computer Science Bibliography [1], which consists of 15 conferences in Computer Science, 2) human protein-to-protein interaction (PPI) data obtained from the IntAct database (Orchard et al. 2013).

In the DBLP dataset, 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 an undirected link if two authors published at least one paper together from 2006 to 2008. As test data, we use new co-author links that emerge between 2009 and 2010. In the PPI dataset `PPI_Data`, nodes

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

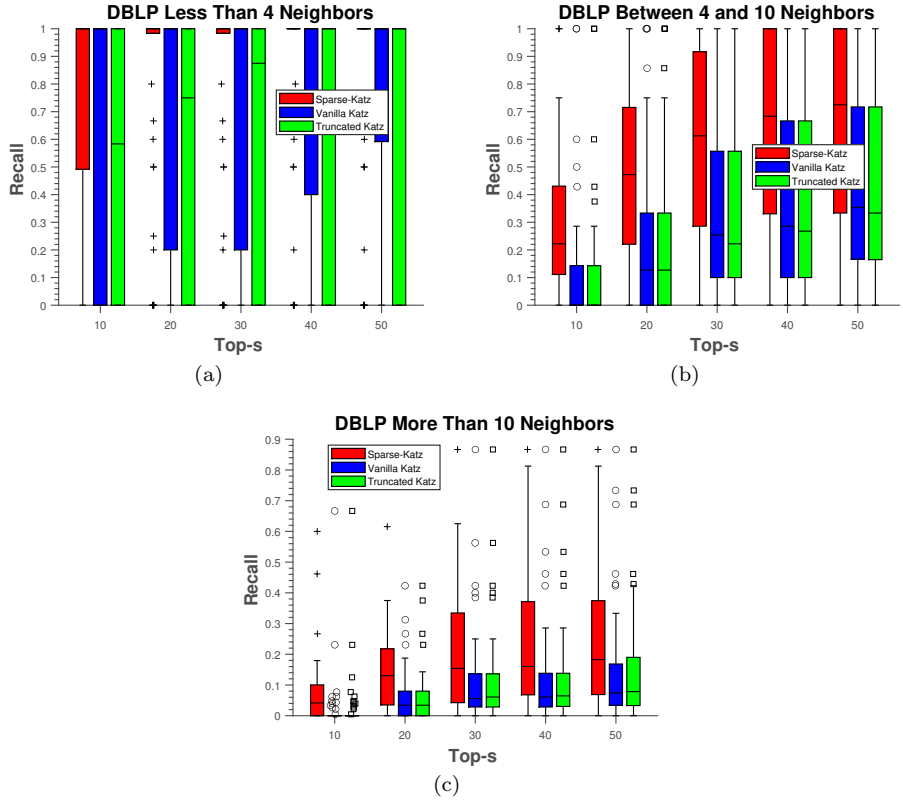


Fig. 5: **Performance evaluation of** SPARSE-KATZ **for** `DBLP_Data` The performance of SPARSE-KATZ with fixed $T = 200$ for link prediction as compared to Vanilla and Truncated Katz measure based link prediction on the `DBLP_Data`

Table 2: **The division of the nodes in the datasets used for link prediction according to their degrees in the training data**. The number of positive links in the test data is shown for each group of nodes.

Network	# of Edges 1 to 3	# of Edges 4 to 10	# of Edges > 10
`DBLP_Data`	10,389	2,877	336
`PPI_Data`	28,490	7,355	3,649

represent and edges represent interactions between. For training data, we use the interactions that are included in the 2014 version of the database. As test data, we use the interactions that are included in the 2016 version of database (which were not included in the 2014 version). These datasets are chosen as realistic cases of network evolution, where the DBLP networks evolve naturally as authors publish new papers, whereas the PPI network evolves

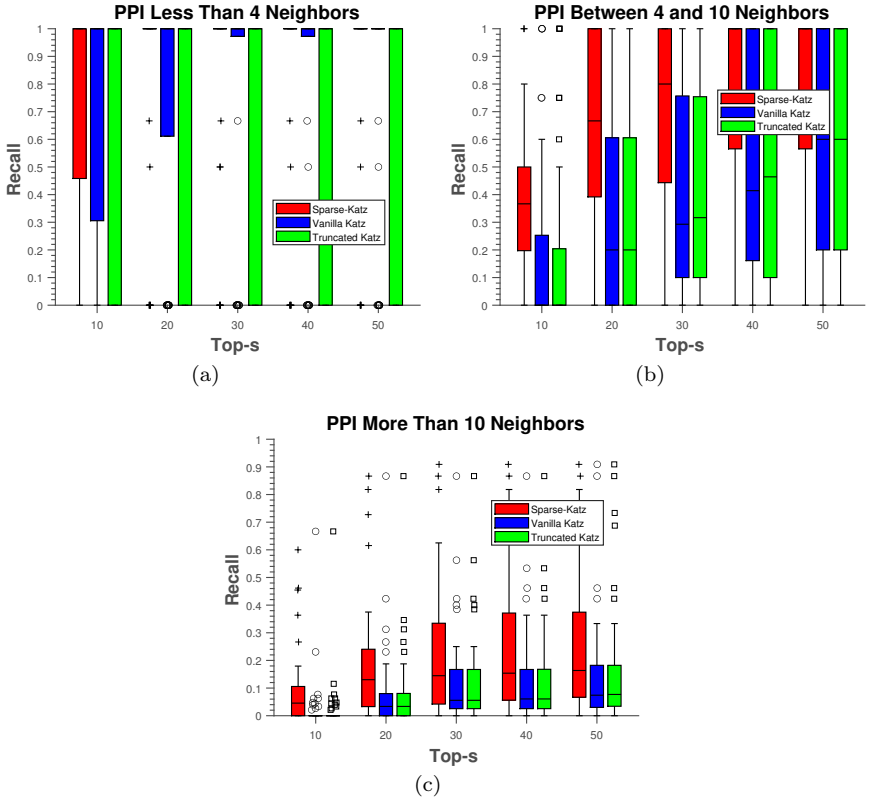


Fig. 6: **Performance evaluation of** SPARSE-KATZ **for** `PPI_Data` The performance of SPARSE-KATZ with fixed $T = 200$ for link prediction as compared to Vanilla and Truncated Katz measure based link prediction on the `PPI_Data`

with the advance of human knowledge on biological systems. These datasets' descriptive statics 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 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. For this reason, to evaluate the accuracy of link prediction methods, we use recall as the evaluation criterion and assess the recall of each method as a function of s (the number of potential new edges that are returned by the query). To investigate the effect of node degree to prediction performance, we stratify the evaluation of recall according to node degree. The distribution of the positive labels into three degree categories are shown in Table 2.

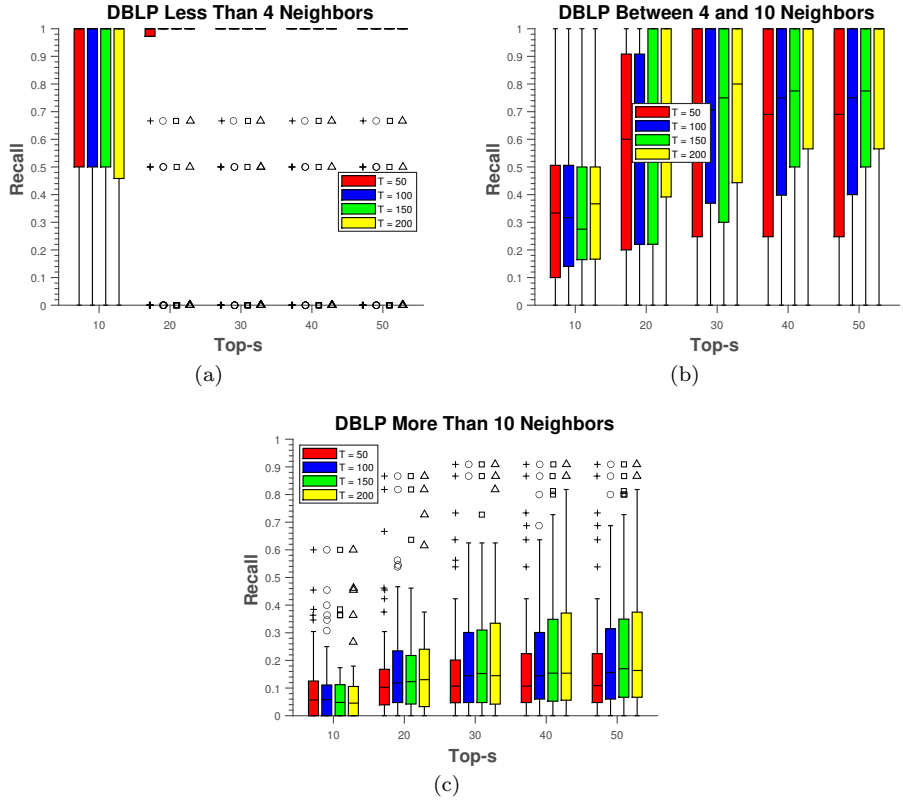


Fig. 7: **The effect of the number of dimensions used to assesss topological similarity (T) on the link prediction performance of** Sparse-Katz **on** `DBLP_Data`**.** The recall (fraction of true positives in the predicted links among all positives) provided by Sparse-Katz for four different values of T are shown as a function of the number of predicted links (s).

For the 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 13602 nodes which gained at least one edge in between 2009 and 2010, i.e, $|\mathcal{P}| = 13602$.
- We divide $\mathcal{P}$ into three categories as summarized in Table 2.
- For all experiments for `DBLP_Data` dataset, we report the mean and the standard deviation of the performance figures.

Similarly, we obtain a positive set $\mathcal{P}$ of 39494 edges for the PPI network.

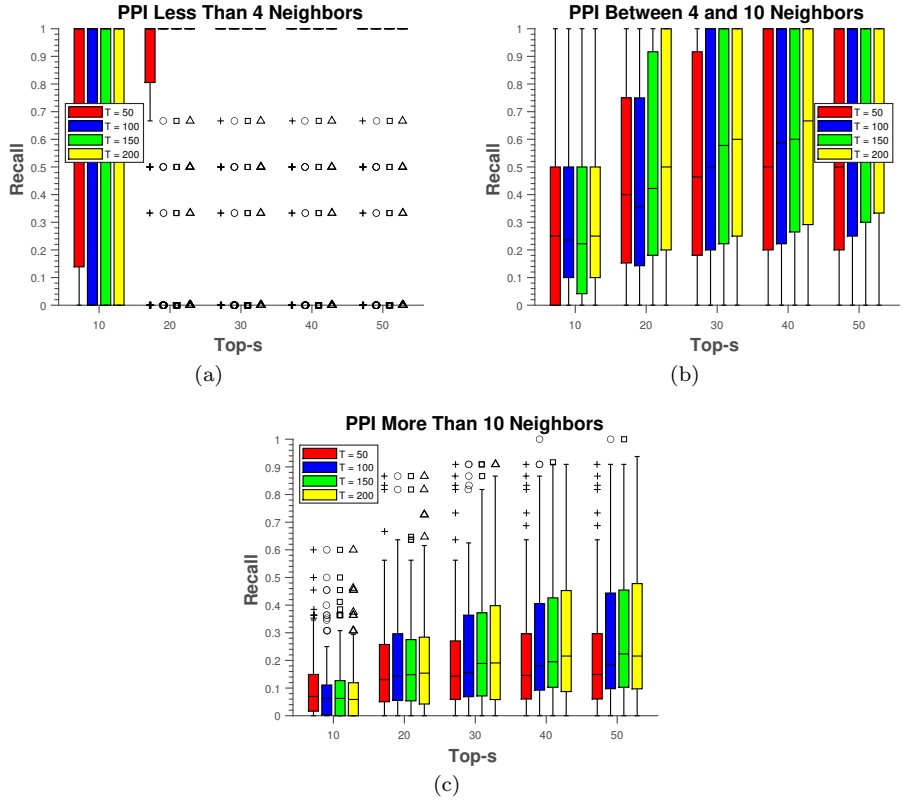


Fig. 8: **The effect of the number of dimensions used to assesss topological similarity (T) on the link prediction performance of** Sparse-Katz **on** `PPI_Data`**.** The recall (fraction of true positives in the predicted links among all positives) provided by Sparse-Katz for four different values of T are shown as a function of the number of predicted links (s).

4.4 Link Prediction Performance for Sparse-Katz

We compare the link prediction performance of Sparse-Katz against that of vanilla Katz and truncated Katz (Lu et al. 2010) with parameters $T = 200$ (i.e., 200 most proximate nodes to the query node are used to assess topological similarity) and $C = 2s$ (i.e., twice as many nodes as the number of potential edges requested by the user are considered as candidates). The results of this analysis for the DBLP data are shown in Figure 5. As seen in the figure, Sparse-Katz significantly outperforms both Vanilla and Truncated Katz across all experiments. The comparison of link prediction accuracy for the `PPI_Data` dataset is shown in the Figure 6. We observe that Sparse-Katz outperforms both of the competing methods on this dataset as well.

These results show that topological similarity based link prediction using Katz-based proximity, which is enabled by LRC-Katz, improves the accuracy of link prediction over methods that directly utilize (exact or approximate) Katz-based proximity to predict links.

Interestingly, truncated Katz does not deliver reasonable *recall* for up to 15 Richardson iterations for low degree nodes (which are the majority of test nodes). However, we confirm the link prediction performance of truncated Katz as in (Lu et al. 2010) for nodes with relatively higher degree. For the high degree nodes, truncated Katz slightly outperforms vanilla Katz with only 5 Richardson iterations.

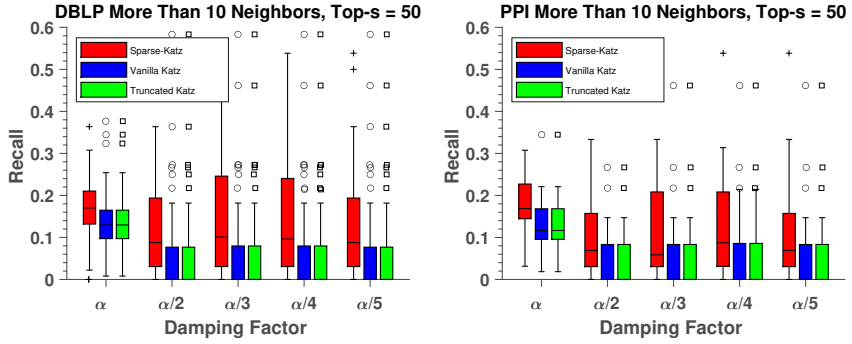
4.5 Effect of Parameter T on Link Prediction

We further evaluate the effect of parameter T, which specifies the number of dimensions used in assessing topological similarity. For this purpose, we randomly sample 100 query nodes from each of the degree groups shown in Table 2. We then plot the recall provided by Sparse-Katz for different values of T ($T = 50$, $T = 100$, $T = 150$, $T = 200$) and as a function of s (the number of predicted links). The results of this analysis are shown in Figure 7 and 8. As seen in both figures, for lower degree nodes, using a low number of dimensions to assess topological similarity is sufficient. However, for medium and high degree nodes, addition of more dimensions largely improves prediction performance, with the performance improvement saturating at around $T = 200$ for both datasets.

4.6 Effect of The Damping Factor (α)

The damping factor, α, is used to adjust the importance of the length of the paths between two nodes in computing Katz-based proximity (with larger *alpha* corresponding to more importance given to shorter paths). For the results reported in this section so far, we use $\alpha = \frac{1}{\|\mathbf{G}\|_2 + 1}$ (Bonchi et al. 2012), since this value is suggested as the hardest case for computing Katz-based proximity (from an efficiency perspective). However, the value of α can also influence the accuracy of link prediction. For this reason, we also systematically examine the effect of the damping factor on the accuracy of link prediction. The results of this analysis are shown in Figure 9. For these experiments, we restrict our analysis to a random sample of 100 high-degree nodes. As it can be seen in the figure, $\alpha = \frac{1}{\|\mathbf{G}\|_2 + 1}$ yields better link prediction accuracy as consistent with the literature (Bonchi et al. 2012). These results suggest that link prediction with Katz-based proximity is more accurate when the search is localized. However, it is important to note that $\alpha = \frac{1}{\|\mathbf{G}\|_2 + 1}$ is the largest possible value of *alpha* that renders the resulting system numerically solvable

and represents the hardest case from the perspective of computational complexity. Therefore, this result also demonstrates the importance of improving the runtime performance of the computation of Katz-based proximity.



(a) (b)

Fig. 9: **The effect of the damping factor (α) on the link prediction accuracy of** Sparse-Katz**.** We randomly select 100 high degree nodes as query nodes for each of the DBLP and PPI datasets, and plot the behavior of recall as a function of α.

4.7 Computational Advantage of Using LRC-Katz in Sparse-Katz

Recall that in Sparse-Katz(Algorithm 3), we need to solve $T + 1$ linear systems of equations for each query node. For this reason, the runtime improvement provided by LRC-Katz is essential in enabling the application of this algorithm. To investigate the effect of the algorithm used to compute Katz-based proximity on the runtime performance of Sparse-Katz, we run the algorithm by changing line 3 and 7 with each of vanilla and truncated Katz. In this experiment, we set $T = 200$ and truncated Katz iteration numbers to 15 and 5 for low and high degree nodes, respectively. We then report runtime for all query nodes the two datasets. These results are shown in detail in Table 2 and visualized in Figure 10 as a function of low rank dimension. As seen in the figure, the advantage of using LRC-Katz in Sparse-Katz is quite pronounced for all types of nodes.

5 Discussion

The application of the proposed algorithms is not limited to link prediction. In this section, we briefly discuss some of our anticipated usage of the developed algorithms in this paper for various data mining/machine learning problems.

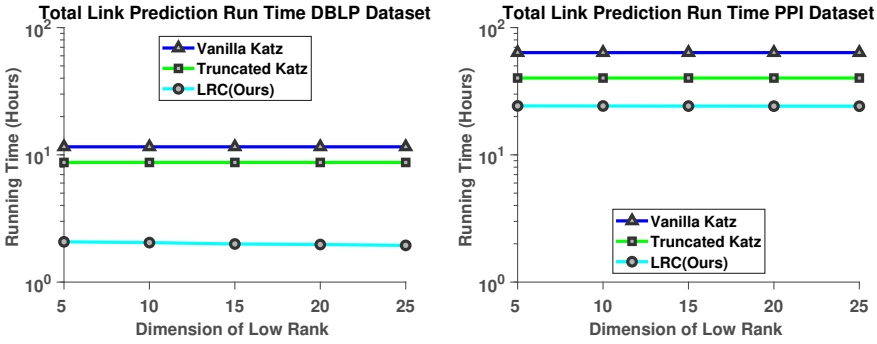


Fig. 10: **Runtime performance evaluation of** Sparse-Katz **for all positive labels in** `DBLP_Data` **and** `PPI_Data` The total runtime performance of Sparse-Katz when we use LRC-Katz , CG and Richardson iterations in Sparse-Katz line 3 and 7.

Semi-Supervised Learning(SSL) is commonly utilized in classification settings when the labeled samples are limited (Chapelle et al. 2006) and there is an underlying graph that represents the potential similarity between all samples. This class of approaches in machine learning have received significant research attention in recent years. In essence, SSL involves the incorporation of a regularization factor that represents the consistency between the labels of the nodes of the graph, which can be formulated using a linear system of equations (Zhou et al. 2004). For instance, Zhou et.al., (Zhou et al. 2004) aim to expand the set of labeled nodes by solving follow equation:

$$F^* = \beta(I - \alpha S)^{-1}Y \tag{15}$$

where S is symmetrically normalized Laplacian matrix of the graph and Y contains very few known labeled nodes. To solve the linear system represented by the above equation, earlier research focuses on computing approximate solutions (Liu et al. 2010). However, it is clear that $(I - \alpha S)$ is a symmetric positive definite matrix for any $\alpha \in (0, 1)$ and thus can efficiently be processed by LRC-Katz. Furthermore, Sparse-Katz can be used for label expansion in the SSL framework.

Graph Convolutional Networks (GCNs) are a variant of traditional Convolutional Neural Networks(CNNs) on graphs (Kipf and Welling 2016). Although the first GCN uses a two-step propagation in its feature propagation phase, later it has been shown that an infinitely many feature propagation steps can be carried out by solving a linear system of equations. Therefore, the feature propagation phase of a GCN can be separated from neural networks for better node classification (Coskun 2019; Klicpera et al. 2019). The task of feature propagation can therefore be formulated as a linear system of equations

in the *softmax* classifier (Klicpera et al. 2019):

$$Z_{PPNP} = softmax(\Pi_{ppr} H) \tag{16}$$

where $\Pi_{ppr} = \alpha(I - (1-\alpha)\hat{\tilde{\mathbf{A}}})^{-1}$ and H is the feature matrix. Here, $\tilde{\mathbf{A}}$ is the self-loop added symmetrically normalized Laplacian matrix and for this reason Π_{ppr} is the inverse of a symmetric positive definite matrix. Thus, LRC-Katz can be used efficiently solve equation 16.

Network Proximity Querying In addition to Katz measure, LRC-Katz can be used for efficiently computing network proximity using other measures, including symmetrically normalized Personalized PageRank (Page et al. 1999) and ParWalk (Wu et al. 2012).

6 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 drastically outperforms the fastest known method, Conjugate Gradient, for a wide ranges of parameter values.

We also develop an effective link prediction algorithm, Sparse-Katz that improves the accuracy of link prediction by assessing topological similarity between nodes as opposed to directly using Katz-based proximity to predict links. Since this algorithm requires repeated computation of Katz-based proximity for a single query, it poses significant challenges in terms of computational complexity. Our results show that, the runtime improvement provided by LRC-Katz in the computation of Katz-based proximity over vanilla and truncated Katz renders application of this link prediction algorithm feasible in a real-time query setting.

Future efforts in this direction would include incorporation of other proximity measures into our framework and their applications, such as semi-supervised learning and graph convolutional networks. 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

Acar, E., Dunlavy, D. M. and Kolda, T. G. (2009), Link prediction on evolving data using matrix and tensor factorizations, *in* 'Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on', IEEE, pp. 262–269.

Amestoy, P. R., Davis, T. A. and Duff, I. S. (1996*a*), 'An approximate minimum degree ordering algorithm', *SIAM Journal on Matrix Analysis and Applications* **17**(4), 886–905.

Amestoy, P. R., Davis, T. A. and Duff, I. S. (1996*b*), 'An approximate minimum degree ordering algorithm', *SIAM Journal on Matrix Analysis and Applications* **17**(4), 886–905.

Boldi, P., Rosa, M., Santini, M. and Vigna, S. (2011), Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks, *in* 'Proceedings of the 20th international conference on World wide web', ACM, pp. 587–596.

Bonchi, F., Esfandiar, P., Gleich, D. F., Greif, C. and Lakshmanan, L. V. (2012), 'Fast matrix computations for pairwise and columnwise commute times and katz scores', *Internet Mathematics* **8**(1-2), 73–112.

Chapelle, O., Schölkopf, B., Zien, A. et al. (2006), 'Semi-supervised learning, vol. 2', *Cambridge: MIT Press. Cortes, C., & Mohri, M.(2014). Domain adaptation and sample bias correction theory and algorithm for regression. Theoretical Computer Science* **519**, 103126.

Coşkun, M., Grama, A. and Koyutürk, M. (2018), 'Indexed fast network proximity querying', *Proc. VLDB Endow.* **11**(8), 840–852.
URL: *https://doi.org/10.14778/3204028.3204029*

Coskun, M. (2019), 'Graph convolutional networks meet with high dimensionality reduction', *arXiv preprint arXiv:1911.02928* .

Coskun, M., Grama, A. and Koyuturk, M. (2016), Efficient processing of network proximity queries via chebyshev acceleration, *in* 'Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, pp. 1515–1524.

Coskun, M. and Koyutürk, M. (2015), 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', IEEE, pp. 485–492.

Demmel, J. W. (1997), *Applied numerical linear algebra*, Vol. 56, Siam.

Erten, S., Bebek, G., Ewing, R. M. and Koyutürk, M. (2011), 'Dada: degree-aware algorithms for network-based disease gene prioritization', *BioData mining* **4**(1), 19.

Erten, S., Bebek, G. and Koyutürk, M. (2011), '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.

Karypis, G. and Kumar, V. (1998*a*), 'A fast and high quality multilevel scheme for partitioning irregular graphs', *SIAM Journal on scientific Computing* **20**(1), 359–392.

Karypis, G. and Kumar, V. (1998*b*), 'A parallel algorithm for multilevel graph partitioning and sparse matrix ordering', *Journal of Parallel and Distributed Computing* **48**(1), 71–95.

Katz, L. (1953), 'A new status index derived from sociometric analysis', *Psychometrika* **18**(1), 39–43.

Kipf, T. N. and Welling, M. (2016), 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907* .

Klicpera, J., Bojchevski, A. and Gunnemann, S. (2019), Combining neural networks with personalized pagerank for classification on graphs, *in* 'International Conference on Learning Representations'.
URL: *https://openreview.net/forum?id=H1gL-2A9Ym*

Leskovec, J., Huttenlocher, D. and Kleinberg, J. (2010), Signed networks in social media, *in* 'Proceedings of the SIGCHI conference on human factors in computing systems', ACM, pp. 1361–1370.

Liben-Nowell, D. and Kleinberg, J. (2007), 'The link-prediction problem for social networks', *Journal of the American society for information science and technology* **58**(7), 1019–1031.

Liu, W., He, J. and Chang, S.-F. (2010), 'Large graph construction for scalable semi-supervised learning'.

Lu, Z., Savas, B., Tang, W. and Dhillon, I. S. (2010), Supervised link prediction using multiple sources, *in* '2010 IEEE international conference on data mining', IEEE, pp. 923–928.

Navlakha, S. and Kingsford, C. (2010), 'The power of protein interaction networks for associating genes with diseases', *Bioinformatics* **26**(8), 1057–1063.

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. (2013), 'The mintact projectintact as a common curation platform for 11 molecular interaction databases', *Nucleic acids research* p. gkt1115.

Page, L., Brin, S., Motwani, R. and Winograd, T. (1999), The pagerank citation ranking: Bringing order to the web., Technical report, Stanford InfoLab.

Rattigan, M. J. and Jensen, D. (2005), 'The case for anomalous link discovery', *Acm Sigkdd Explorations Newsletter* **7**(2), 41–47.

Saad, Y. (2003), *Iterative methods for sparse linear systems*, Vol. 82, siam.

Saerens, M., Fouss, F., Yen, L. and Dupont, P. (2004), The principal components analysis of a graph, and its relationships to spectral clustering, *in* 'European Conference on Machine Learning', Springer, pp. 371–383.

Sarkar, P. and Moore, A. (2012*a*), 'A tractable approach to finding closest truncated-commute-time neighbors in large graphs', *arXiv preprint arXiv:1206.5259* .

Sarkar, P. and Moore, A. (2012*b*), 'A tractable approach to finding closest truncated-commute-time neighbors in large graphs', *arXiv preprint arXiv:1206.5259* .

Skogent, M. (1992), Domain decomposition algorithms of schwarz type, designed for massively parallel computers, *in* 'Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations', number 55, SIAM, p. 362.

Smith, B., Bjorstad, P. and Gropp, W. (2004), *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*, Cambridge university press.

Van der Vorst, H. A. and Chan, T. F. (1997), Linear system solvers: sparse iterative methods, *in* 'Parallel numerical algorithms', Springer, pp. 91–118.

Wang, C., Satuluri, V. and Parthasarathy, S. (2007), Local probabilistic models for link prediction, *in* 'icdm', IEEE, pp. 322–331.

Wu, X.-M., Li, Z., So, A. M., Wright, J. and Chang, S.-F. (2012), Learning with partially absorbing random walks, *in* 'Advances in Neural Information Processing Systems', pp. 3077–3085.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J. and Schölkopf, B. (2004), Learning with local and global consistency, *in* 'Advances in neural information processing systems', pp. 321–328.