Indexed Fast Network Proximity Querying

Mustafa Coşkun
(1) Electrical Engineering & Computer Science
Case Western Reserve University,
Cleveland, OH 44106, USA.
(2) Qatar Computing Research Institute, HBKU
Doha, Qatar.
mustafa.coskun@case.edu

Ananth Grama
Department of Computer Science
Purdue University
West Lafayette, IN 47906, USA.
ayg@cs.purdue.edu

Mehmet Koyutürk
(1) Electrical Engineering & Computer Science
(2) Center for Proteomics & Bioinformatics
Case Western Reserve University
Cleveland, OH 44106, USA.
mehmet.koyuturk@case.edu

ABSTRACT

Node proximity queries are among the most common operations on network databases. A common measure of node proximity is random walk based proximity, which has been shown to be less susceptible to noise and missing data. Real-time processing of random-walk based proximity queries poses significant computational challenges for larger graphs with over billions of nodes and edges, since it involves solution of large linear systems of equations. Due to the importance of this operation, significant effort has been devoted to developing efficient methods for random-walk based node proximity computations. These methods either aim to speed up iterative computations by exploiting numerical properties of random walks, or rely on computation and storage of matrix inverses to avoid computation during query processing. Although both approaches have been well studied, the speed-up achieved by iterative approaches does not translate to real-time query processing, and the storage requirements of inversion-based approaches prohibit their use on very large graph databases.

We present a novel approach to significantly reducing the computational cost of random walk based node proximity queries with scalable indexing. Our approach combines domain graph-partitioning based indexing with fast iterative computations during query processing using Chebyshev polynomials over the complex elliptic plane. This approach combines the query processing benefits of inversion techniques with the memory and storage benefits of iterative approache. Using real-world networks with billions of nodes and edges, and top-k proximity queries as the benchmark problem, we show that our algorithm, I-Chopper, significantly outperforms existing methods. Specifically, it drastically reduces convergence time of the iterative procedure, while also reducing storage requirements for indexing.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 44th International Conference on Very Large Data Bases, August 2018, Rio de Janeiro, Brazil.
Proceedings of the VLDB Endowment, Vol. 11, No. 8
Copyright 2018 VLDB Endowment 2150-8097/18/4.
DOI: https://doi.org/10.14778/3204028.3204029.

PVLDB Reference Format:
Mustafa Coşkun, Ananth Grama, Mehmet Koyutürk. Indexed Fast Network Proximity Querying. *P*VLDB, 11 (8): 840-852, 2018.
DOI: https://doi.org/10.14778/3204028.3204029.

1. INTRODUCTION

Computation of proximity measures for nodes in networks is a common operation in diverse data analytic applications. In link prediction or recommender systems, network proximity quantifies node similarity in heterogeneous networks [20, 34]. In information retrieval, nodes are ranked based on their random walk proximity to other nodes [35, 43]. In computational biology, genes are ranked according to their network proximity to known disease implicated genes [28].

The general setup for network proximity queries is as follows: Given a query node, we are interested in computing a score for all other nodes in the network, based on their proximity to the query node with respect to some notion of network distance. Common measures of network distance include shortest path, which computes the minimum number of edges between two nodes, and random-walk-based proximity, which simulates random walks starting from the query node to compute stationary probabilities for the random walker to end at each other node. In top-k proximity queries, we are interested in returning the k nodes that are closest to the query node. In many applications, random walk based proximities are preferable to shortest path distance, since they capture the global structure of a network and multi-faceted relationships of the nodes [19, 36]. Random walk based proximity measures have been used in a number of applications, including web search [30], clustering [2], and link prediction [11, 25]. Some of the well known random walk based proximity measures include SimRank [15], network propagation [38], diffusion state distance [8], and random walk with restarts (RWR) [37].

Owing to its widespread use, significant efforts have been devoted to reducing computational costs associated random walk proximity. These efforts typically speed-up computations by implementing 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 QR/LU decompositions and storing the resulting decompositions as an index. For instance, in the context of top-k proximity queries, many state-of-the-art methods [10, 14, 39, 41] use breadth-first ordering of the nodes in the network to bound element-wise increments of proximity scores in the iterative computation. 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. Likewise, using a QR or LU decomposition of the underlying linear system, top-k proximity computations can be computed efficiently [4, 14]. These techniques have been demonstrated to yield significant improvement in runtime, however, their application to larger networks (tens of millions of nodes and billions of edges) is limited. Specifically, for very large networks, iterative methods [10,14,39,41] require a large number of iterations to converge. Among these, CHOPPER offers tightest convergence upper bound and best performance, compared to other methods. However, its applicability is limited to undirected graphs. On the other hand, direct inversion techniques [4,14] are not scalable to large matrices, since the inverse of a sparse matrix is usually dense, thus posing strong memory (for the computation of the index) and storage (for the storage of the index) constraints.

In this paper, we propose a hybrid approach that partitions the network into disjoint sub-networks and inverts the matrices corresponding to these subnetworks. The denser matrix, composed of the nodes connecting these subnetworks is handled through a suitably accelerated iterative procedure. By inverting small sparse matrices, the hybrid procedure overcomes the memory constraint of direct inversion techniques. By performing the iterative procedure on a smaller matrix and suitably accelerating this procedure, it overcomes the computational cost considerations of iterative methods. We first use graph partitioning [22] to partition the matrix into a sparse block-diagonal matrix and a dense, but much smaller matrix. We then index the inverse of the sparse block diagonal matrix through a computationally efficient procedure. We then use Chebyshev polynomials, along with an incomplete LU decomposition based preconditioner [5], to speed up the iterative process for the dense matrix. At query time, we first solve the dense part of the system using the proposed 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 the hybrid approach converges much faster than power method based formulations and is much more scalable than direct inversion methods. Our hybrid approach significantly accelerates the computation of random walk based top-k proximity queries for very large graphs. We provide a detailed theoretical analysis for our results and experimentally show the superior performance of our method on a number of real-world networks. Specifically, we show that: (i) our method yields significant improvement in storage requirements for the index, about 50-fold for the tested networks, over a state-of-the-art method, which can only be applied to the smallest network in our experiments; and (ii)our method yields over 25-fold improvement in the runtime of online query processing over one of the very few state-of-the-art methods that can scale to graphs with tens of millions of nodes and billions of edges.

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

- We introduce a hybrid approach to indexing-based acceleration of 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 Chebyshev acceleration.
- We generalize the application of Chebyshev acceleration to directed graphs, enabling fast processing of proximity queries in a broader range of networks. The application of Chebyshev acceleration to directed graphs is not straightforward since the matrix underlying the linear system that is associated with a directed graph may have complex eigenvalues.

Taken together, these two contributions bring the field closer to real-time processing of proximity queries 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 network proximity queries. In Section 3, we define random walk proximity and top-k proximity queries, and describe our method, along with its analysis. In Section 4, we provide detailed experimental assessments of our method on very large networks. We conclude our discussion and summarize avenues for future research in Section 5.

2. RELATED WORK

Node proximity queries have received significant research attention in recent years in the context of searching, ranking, and analyzing network structured object (node) similarity [14]. In particular, top-k proximity queries in graph databases have been well studied, since the original PageRank results [30]. One of the commonly used approaches to computing random walk based proximity is the power method [32]. An alternate approach to power iterations is the use of offline computation, which directly inverts the underlying linear system of equations, typically using LU decomposition or eigen-decomposition [14,37,40]. These methods can compute network proximity rapidly, using a single matrix vector multiplication, however, they involve expensive preprocessing, and their memory requirements constrain their use to smaller networks. Another LU based approach proposed by Shin et al. [33] by applies Schur Lemma to build an index for RWR-based proximity queries. Namely, they partition the matrix so that low degree nodes' matrix is block-diagonal (thus easily invertible) and use LU-decomposition to reduce the space requirement for storing Schur Matrix. However, Shin et al.'s method is aimed at using an exact solution to the linear system for indexing. Therefore, this method is limited by the space requirement of the inverse of Schur Matrix. In contrast, we here propose to compute an approximate solution to linear system of equations by approximating the inverse of Schur Matrix using incomplete LU factorization, and use preconditioned and accelerated iterations to compute the exact solution at query time. In doing so, we significantly improve on both memory and storage requirements for indexing.

There have also been several efforts aimed at scaling top-k proximity queries to large sparse networks. These methods take advantage of the numerical/ structural properties of the network to bound the proximity of nodes in the iterative computation, and to retrieve top-k most proximate nodes with respect to a given query node [10, 17, 39, 41]. Other methods utilize the topology of the network to perform a local search around the query object, 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 [3, 6, 27, 42]. However, these local search based methods are approximate, in the sense that they do not guarantee the precise set of k nodes that are most proximate to the query node. Recently, two efficient methods, FLoS [39] and Chopper [10] have been proposed for exact computation of top-k proximity queries efficiently. However, these methods are only applicable to undirected graphs.

In this paper, we focus on exact computation of network proximity in very large networks using a novel, hybrid approach. Our method is fundamentally different from existing approaches in that it simultaneously targets scalability and efficiency, in terms of memory requirements of preprocessing, storage requirements of indexing, and the computational cost of query processing. Our method can be used for efficiently computing random walk based proximity of all nodes in a directed/undirected network, or to speed up processing of top-k proximity queries.

3. METHODS

In this section, we first introduce random walk with restarts (RWR) and the top-k network proximity querying problem based on RWR. We then describe our graph-partitioning based approach to partitioning the resulting linear system and indexing the sparser part of the system. Subsequently, we discuss how the iterative computation can be accelerated using Chebyshev polynomials 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 top-k network proximity queries. The workflow of the proposed framework is shown in Figure 1.

3.1 RWR-Based Proximity

Let $\mathbf{G} = (\mathbf{V},\ \mathbf{E})$ be a directed or undirected network, where $\mathbf{V}$ denotes the set of nodes and $\mathbf{E}$ denotes the set of edges. Given a node $q \in \mathbf{V}$, random walk with restarts based proximity to q is defined as follows:

$$x_q = (1-\alpha)\mathbf{P}x_q + \alpha\mathbf{s}_q. \tag{1}$$

Here, $\mathbf{P}$ denotes the column-normalized stochastic matrix derived from the adjacency matrix $\mathbf{A}$ of $\mathbf{G}$ by dividing each entry by the corresponding column sum, $\mathbf{s}_q$ denotes the restart vector that contains a 1 at its qth entry and a 0 in all other entries (or we can apply queries that involve proximity to a set of nodes by assigning probabilities to each query node; this procedure is called *personalized pagerank*), and $0 < \alpha < 1$ is the teleport probability, or damping factor that is used to adjust the localization of search (a larger α results in a search that is more concentrated around the neighborhood of the query node). Defined this way, $\mathbf{x}_q(u)$ represents the probability of being at node u at a random step of a sufficiently long random walk that starts at q and either moves to an adjacent node (with probability $1-\alpha$) or restarts at node q (with probability α) at each step.

In practice, $\mathbf{x}_q$ is computed iteratively, by setting ${\mathbf{x}_q}^{(0)} = \mathbf{s}_q$ and computing:

$${\mathbf{x}_q}^{(t+1)} = (1-\alpha)\mathbf{P}{\mathbf{x}_q}^{(t)} + \alpha\mathbf{s}_q \tag{2}$$

in the tth iteration. This iterative procedure terminates when $||{\mathbf{x}_q}^{(t+1)} - {\mathbf{x}_q}^{(t)}||_2$ is below a prescribed threshold, implying convergence. Since this power iteration method uses repeated sparse matrix-vector multiplications with virtually no extra storage, it is appealing for scalability. However, for large graphs and relatively small restart probability, α, the number of iterations can be large, rendering the computation expensive. This is particularly problematic for real-time processing of proximity queries in very large graph databases.

An alternate approach to computing RWR-based proximity is to rewrite Equation (1) as a linear system of equations:

$$\mathbf{M}\mathbf{x}_q = \mathbf{b}, \tag{3}$$

where $\mathbf{M} = (\mathbf{I} - (1-\alpha)\mathbf{P}), \mathbf{b} = \alpha\mathbf{s}_q$, and $\mathbf{I}$ is the identity matrix. Since $\mathbf{M}$ does not depend on the query, it can be inverted offline, and the inverse can be used as an index to compute $x_q = \mathbf{M}^{-1}\mathbf{b}$ by performing a single matrix vector multiplication during online query processing. Note that this solution applies to all linear systems derived from random walk computations, since $0 < \alpha < 1$, and therefore the coefficient matrix is diagonally dominant and invertible. In practice, 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.

3.2 Top-k Proximity Queries

Given a network $\mathbf{G} = (\mathbf{V},\ \mathbf{E})$, a query node $q \in \mathbf{V}$, and a positive number k, the top-k proximity query for RWR-based proximity aims to identify k nodes in $\mathbf{V}$ that have the largest values in x_q [16]. The state-of-the-art in efficient computation of top-k proximity queries is based on two approaches. The first approach uses the error bound in the iterative computation of x_q to eliminate nodes whose proximity values cannot exceed the proximity values of the nodes that are already in the top-k [10, 16, 39, 41]. This elimination process continues until all but the top-k nodes are eliminated. Among these methods, Chopper , which is based on Chebyshev Polynomials, offers the tightest error bound [10]. Chopper yields asymptotically faster convergence in theory, and significantly reduces convergence times in practice.

The second approach to efficient processing of top-k proximity queries is to use index based preprocessing [14] by performing LU decomposition of the linear system (3). As discussed before, this approach improves efficiency by requiring a single matrix vector multiplication during query processing. However, this approach is only applicable to smaller graphs due to scalability issues (both memory and computation) associated with the inversion of the complete LU decomposition.

In this paper, we propose a hybrid method that effectively combines the two approaches, to take advantage of indexing, as well as accelerated convergence. Our method efficiently approximates the inverse of the matrix, and uses this approximation to build an index with significantly less space requirement than that of the actual inverse. At query time, we use this index to compute part of the linear system in Equation (3). We then use a Chebyshev polynomial based method to further accelerate the iterative computation, to enable processing of top-k proximity queries on large graphs within a few iterations, for the remaining part of the lin-

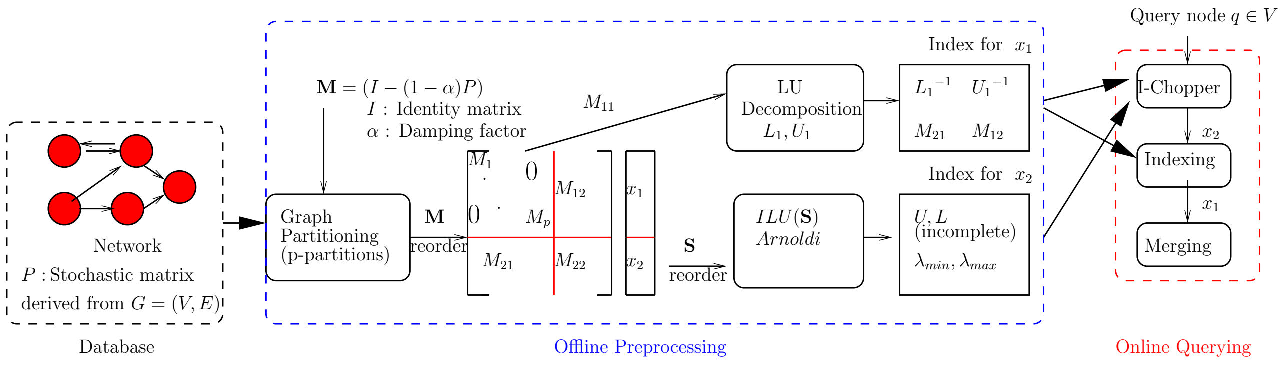
Query node $q \in V$
Index for x_1
$\mathbf{M} = (I-(1-\alpha)P)$
I : Identity matrix
α : Damping factor
M_{11}
LU Decomposition L_1, U_1
${L_1}^{-1}$ ${U_1}^{-1}$ M_{21} M_{12}
Index for x_2
I-Chopper
x_2
Indexing
x_1
Merging
Network
P : Stochastic matrix derived from $G=(V,E)$
Graph Partitioning (p-partitions)
$\mathbf{M}$ reorder
M_1 0 0 M_p M_{12} M_{21} M_{22} x_1 x_2
$\mathbf{S}$ reorder
$ILU(\mathbf{S})$ Arnoldi
U, L (incomplete) $\lambda_{min}, \lambda_{max}$
Database
Offline Preprocessing
Online Querying

Figure 1: **Flowchart illustrating the proposed framework for indexed processing of proximity queries on large networks.** For a given directed network in the database, the proposed index is constructed as follows: We first construct $\mathbf{M}$ and use *PartGraphRecursive* graph partitioning method in METIS to partition $\mathbf{M}$ in such a way that M_{11} is a sparse block diagonal matrix, and M_{22} is dense but smaller. After partitioning the linear system, we compute the LU decomposition of sparse block-diagonal matrix M_{11} and obtain ${L_1}^{-1}$ and ${U_1}^{-1}$. Then, we construct $S = M_{22} - M_{21}{M^{-1}}_{11}M_{12}$, and compute its incomplete LU decomposition as $[L, U] = ILU(S)$. Subsequently, we use Arnoldi's method to find the maximum and minimum eigenvalues of S, and store the resulting values and matrices in an index. Next, we use the indexed matrices and extreme eigenvalues of S to compute x_2, the lower part of solution of the linear system, using a preconditioned Chebyshev acceleration. Finally, using x_2 and the indexed matrices, we compute x_1, the upper part of solution of the linear system. We then merge the entries in x_1 and x_2, and return the nodes that correspond to the top-k entries in the resulting merged vector.

ear system (3). The flowchart of the proposed framework is shown in Figure 1.

3.3 Graph Partitioning Based Indexing

Inverting $\mathbf{M}$ involves significant computation (depending on the structure/ sparsity of the graph – cubic in the number of nodes in the worst case) and memory that is quadratic in number of nodes. Our key observation to addressing this problem is that effective use of indexing does not require computation of an exact solution to (3). Rather, if we can solve a large part of the linear system in Equation (3) by exploiting the sparsity of the network, we can use a fast iterative method to solve the remaining (and smaller) part of the system quickly. In other words, we can use a approximate solution to the linear system for indexing, and we can use a fast iterative solver during query processing to compute the exact solution. The following lemma lays the foundation for our method:

LEMMA 1. *Suppose a linear system $\boldsymbol{M}x_q = b$ can be partitioned as* $\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} x_{q1} \\ x_{q2} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, *such that M_{11} is invertable. Then, letting $S = M_{22} - M_{21}{M^{-1}}_{11}M_{12}$ denote the Schur complement of M_{11}, the linear system can be solved as:*

$$x_q = \begin{bmatrix} x_{q1} \\ x_{q2} \end{bmatrix} = \begin{bmatrix} {M_{11}}^{-1}(b_1 - M_{12}x_{q2}) \\ S^{-1}(b_2 - M_{21}{M_{11}}^{-1}b_1) \end{bmatrix}.$$

The proof of this lemma is provided by Boyd et al. [7]. This lemma applies to the solution of linear systems derived from random walk computations, since $\mathbf{M}$ is diagonally dominant and invertible as discussed above.

Observe that, by the above lemma, the solution to a linear system involves inversion of M_{11} and S. Now if M_{11} is large and block-diagonal with a small bandwidth, then it can be inverted easily and its inverse can be efficiently stored since it will also be sparse. In this case, the matrix S will be relatively small compared to the matrix $\mathbf{M}$, and the part of the system that involves inversion of S can be solved iteratively during query processing. This approach is well-suited to the processing of network proximity queries on very large graphs, since the networks are usually very sparse and contain many low-degree nodes. For this reason, it is possible to partition these networks to obtain a large and block-diagonal M_{11} (representing the loosely connected nodes) and a dense but relatively small S (representing the hubs in the network). To obtain a block-diagonal M_{11}, we use graph partitioning, which is commonly used in solving linear systems that arise from partial differential equations [32].

Graph partitioning involves partitioning the vertices of a given graph into two or more partitions, to optimize an objective function that can be equal to the number of edges or vertices cut by the partition, or a function of these. Here, an edge is said to be cut by the partition if its two incident vertices are assigned to different parts, whereas a vertex is said to be cut by the partition (or belongs to the vertex separator) if it is adjacent to vertices in more than one part. For example, in the paralellization of numerical methods that solve linear systems, the number of edges on the cut represents the amount of communication between processes, hence minimizing the number of edges in the cut is desired [32]. On the other hand, in the context of our problem, a multi-way minimum-vertex-separator partitioning of the network is desirable, since we are interested in minizing the number of vertices (i.e., rows/columns of $\mathbf{M}$) that are on the cut, since the vertices on the cut correspond to the rows and columns of $\mathbf{M}$ that are assigned to the denser part of the matrix.

To explain the formulation of the reordering of $\mathbf{M}$ as a graph partitioning problem, we first note that the matrices $\mathbf{M}$, $\mathbf{P}$, and $\mathbf{A}$ have identical non-zero structures (except for diagonal entries), which can be represented as a graph by $\mathbf{G}$ itself. Therefore, a multi-way minimum-vertex-separator partitioning of $\mathbf{G}$ will result in a large number of small parts with most edges within the parts. Consequently, the matrix $\mathbf{M}$ can be reordered to obtain a linear system in the form given in Lemma 1 [9,21,22] as follows: (1) Small parts with nodes heavily connected to each other with no edges between the parts, forming the diagonal blocks of M_{11}; (2)Local interfaces between these parts, i.e., the edges between the vertices assigned to the parts and the vertices in the vertex separator, that is M_{12} and M_{21}; (3) The edges between the nodes in the vertex separator, i.e., M_{22} [24]. In this formulation, there is clearly a trade-off between the bandwidth of the matrix M_{11} and the number of nodes in the vertex separator (i.e., the size of matrix M_{22}) [32]. The PartGraphRecursive package implemented in the MeTiS graph partitioning tool [21] allows the user to put a constraint on the size the vertex separator (as opposed to minimizing it), thereby enabling the user to directly balance this trade-off. This is desirable for our application since graph partitioning is performed offline in our framework, thus this parameter can be tuned for each network during preprocessing and indexing. For this reason, we use the PartGraphRecursive package in our implementation and provide a detailed analysis on the effect of this parameter on preprocessing time, index size, and query processing time in the Experimental Results.

In summary, the idea is to partition $\mathbf{G}$ in such a way that all subnetworks (corresponding to disjoint graph partitions) of $\mathbf{M}$ can be ordered as a block diagonal matrix to render ${M_{11}}^{-1}$ to be sparse. Once M_{11} is computed, we compute ${M_{11}}^{-1}$, which is also sparse and can be stored as an index. Note that inversion of M_{11} is feasible even for billion-scale graphs since it is block diagonal with a small bandwidth and many efficient algorithms exist for inverting banded matrices. In our implementation, we use the Incomplete LU decomposition along with approximate minimum algorithms [1,12]. At query time, if we can compute an exact solution to x_{q_2}, we can compute an exact solution to $x_{q_1} = {M_{11}}^{-1}(b_1 - M_{12}x_{q_2})$ by performing a single matrix vector multiplication.

In the next section, we discuss how to efficiently solve the system $Sx_{q_2} = (b_2 - M_{21}{M_{11}}^{-1}b_1)$ using Chebyshev polynomials. In order to use these polynomials, we need to know the maximum and minimum eigenvalues of matrix S. We compute these eigenvalues using the Arnoldi method [13] in the "offline" preprocessing phase. We explain how and why we compute the extreme eigenvalues of S in the following section.

3.4 Query Processing

In the query phase, for a given query node, q, we first construct $\mathbf{s}_q$, which denotes the restart vector and contains a 1 at its qth entry and a 0 in all other entries. Next, we reorder the entries of $\mathbf{s}_q$ and divide $\mathbf{s}_q$ into two parts, $\mathbf{s}_q = \begin{bmatrix} s_{q_1} \\ s_{q_2} \end{bmatrix}$, using the node ordering and graph partitioning based indexing described in the previous section. Finally, we set $b_1 = \alpha s_{q_1}$ and $b_2 = \alpha s_{q_2}$, where α is restart probability. Then, we use the b_2 solve for x_{q_2} using the matrices stored in the index, and an accelerated iterative procedure using Chebyshev polynomials and incomplete LU-decomposition based preconditioning. Once x_{q_2} is computed, we compute x_{q_1} directly using the matrices stored in the index.

To solve for x_{q_2}, we rewrite $Sx_{q_2} = (b_2 - M_{21}{M_{11}}^{-1}b_1)$ as:

$$Sx_{q_2} = f \tag{4}$$

Algorithm 1: The Preprocessing Phase

```
   Method: $PreProcess(\mathbf{G}, \alpha, t)$
   Input  : $\mathbf{G}$, $\alpha$, $t$
   Output : ${U_1}^{-1}, {L_1}^{-1}, U, L, d, c$, and $\sigma_{min}$
1  Construct $\mathbf{P}$ by column normalizing $\mathbf{G}$
2  Construct $\mathbf{M} \leftarrow (\mathbf{I} - (1-\alpha)\mathbf{P})$
3  Use minimum-vertex-seperator graph partitioning on
     $\mathbf{M}$ to partition it into $M_{11}, M_{21}, M_{12}, M_{22}$ [21,22]
4  Decompose $M_{11}$ into $L_1$ and $U_1$ using LU
     decomposition and invert $L_1$ and $U_1$
5  Construct and decompose $S$ into $L$ and $U$ using
     incomplete LU decomposition [13].
6  Create $\mathbf{l}^{(0)}$ as normalized random vector
7  Compute $[\lambda_{min}, \lambda_{max}] = Arnoldi(S\mathbf{l}^{(0)}, t, \mathbf{l}^{(0)})$
8  Compute $[\sigma_{min}, \sigma_{max}] = Lanczos(S^T S\mathbf{l}^{(0)}, t, \mathbf{l}^{(0)})$ [13]
9  Set $c = \frac{\lambda_{max} + \lambda_{min}}{2}$ and $d = \frac{\lambda_{max} - \lambda_{min}}{2}$
10 end Set $\mu = \left(\frac{a + \sqrt{a^2 - d^2}}{c + \sqrt{c^2 - d^2}}\right)$ and $\zeta \leftarrow \frac{1}{c}$
```

where $f = (b_2 - M_{21}{M_{11}}^{-1}b_1)$.

Here, f can be computed efficiently, since we precompute and index M_{21} and ${M_{11}}^{-1}$ in the indexing phase. However, since S is relatively dense, solving (4) potentially takes a large number of iterations using standard iterative methods [32]. For this reason, we solve (4) by accelerating an iterative procedure using Chebyshev polynomials. To simplify the notation, we drop the subscripts term (q) in the following sections.

3.4.1 Accelerating Iterative Procedures

The idea behind Chebyshev acceleration is as follows: To solve $Sx_2 = f$, in the general power method, we start with an initial guess $\mathbf{x}_2^{(0)}$ and define the first residual as $\mathbf{r}_2^{(0)} = f - S\mathbf{x}_2^{(0)}$. This residual is used to construct $\mathbf{x}_2^{(1)} = \mathbf{x}_2^{(0)} + \mathbf{r}_2^{(0)}$. For the tth iteration of this procedure, the tth iterate is defined as $\mathbf{x}_2^{(t)} = \mathbf{x}_2^{(t-1)} + \mathbf{r}_2^{(t-1)}$. Notice that all the precomputed residuals $\mathbf{r}_2^{(0)}, ..., \mathbf{r}_2^{(t-2)}$ are discarded in this procedure. In Chebyshev acceleration, instead, we keep all the residuals computed till the tth iteration to compute a better approximation to $\mathbf{x}_2$, a linear combination of the residuals from previous iterations. For this purpose, for a given series $\gamma_t(m)$ for $0 \le m \le t-1$, we define vector $\mathbf{y}_2^{(t)}$ as follows:

$$\mathbf{y}_2^{(t)} = \mathbf{y}_2^{(t-1)} + \sum_{m=0}^{t-1} \gamma_t(m)\mathbf{r}_2^{(m)} \tag{5}$$

Our objective is to find a sequence γ_t such that the sequence $\mathbf{y}_2^{(t)}$ converges to $\mathbf{x}_2$ faster than $\mathbf{x}_2^{(t)}$. Note that, as we explain later in this section, we do not need to store all above vectors to compute $\mathbf{y}_2^{(t)}$. We only need to consider three vectors by exploiting the three-term recurrence of Chebyshev polynomial for computing $\mathbf{y}_2^{(t)}$. Observe that, if we use $\mathbf{y}_2^{(t)}$ to approximate $\mathbf{x}_2$, the error in the tth iteration can be defined as:

$$\mathbf{e}^{(t)} = \mathbf{y}_2^{(t)} - \mathbf{x}_2. \tag{6}$$

Thus, for each t, the chosen γ_t should minimize $||\mathbf{e}^{(t)}||_2$, subject to the constraint: $\sum_{m=0}^{t} \gamma_t(m) = 1$. This constraint ensures that $\mathbf{y}_2^{(t)}$ converges to $\mathbf{x}_2$.

LEMMA 2. *The error at tth step is given by $\mathbf{e}^{(t)} = p_t(S)\mathbf{e}^{(0)}$, where p_t is a monic polynomial with $p_t(0) = 1$.*

PROOF. By induction. Omitted. $\square$

This lemma suggests that we need to find a sequence of polynomials p_t such that $\|\mathbf{e}^{(t)}\| \leq \|p_t(S)\|\|\mathbf{e}^{(0)}\|$ is as small as possible. From Jordan form for any diagonally dominant matrix S and polynomial p_t, we can write $\Lambda(p_t(S)) = p_t(\Lambda(S))$, where $\Lambda(.)$ denotes the set of eigenvalues of the matrix [31]. This suggests that we must choose a polynomial p_t that has smallest degree in the spectrum of S with $p_t(0) = 1$. This last observation implies that we need to compute the spectrum of S to choose the smallest degree polynomial in it (this can be thought of as solving a min-max problem over the spectrum of given a matrix). Computing all eigenvalues of S would defeat our purpose, since this computation is more expensive than solving (4). Instead of computing the spectrum of S, the commonly used strategy is to relax the min-max problem by considering the extreme eigenvalues of S [13]. For this reason, we compute the maximum and minimum eigenvalues of S matrix using the Arnoldi method [13] in the indexing phase.

3.4.2 *Chebyshev Polynomials*

Based on a similar idea and knowledge of real eigenvalue bound of the stochastic matrix of an undirected graph, CHOPPER [10] accelerates RWR-based proximity queries by solving the relaxed min-max problem:

$$\min_{p_t \in \mathbb{P}_t, p_t(\lambda)=1} \quad F(p_t) \tag{7}$$

where $\mathbb{P}_t$ denotes the family of all polynomials of order t and

$$F(p_t) = \max_{\lambda \in [-1+\alpha, 1-\alpha]} |p_t(\lambda)|. \tag{8}$$

CHOPPER [10] uses the eigenvalue bounds, $[-1+\alpha, 1-\alpha] \subseteq \mathbb{R}$, of the stochastic matrix of an undirected graph and the well known Chebyshev polynomial of first kind, given by the following recurrence: $T_0(z) = 1, T_1(z) = z, T_{t+1}(z) = 2zT_t(z) - T_{t-1}(z)$to solve min-max problem (8).

Notice, however, that in equation (8), eigenvalue bounds, $[-1+\alpha, 1-\alpha]$ are real. This holds for only undirected graphs. For directed graphs, we have a real valued general matrix S, which may have complex eigenvalues. Hence, we must define the eigenvalue bound in the complex elliptic plane as illustrated in Figure 2, and replace the eigenvalue bounds $[-1+\alpha, 1-\alpha] \subseteq \mathbb{R}$ with an ellipse. That is, we need to find an ellipse that encloses the spectrum of S.

To further elaborate, let $E(c, d, a)$ denote the family of all ellipses that are centered at c with foci $d + c$ and $d - c$ with semi-major axes a in a contiguous complex plane. We are looking for an ellipse that encapsulates the spectral radius of S to solve the min-max problem in complex plane, as in [10]. Finding such an ellipse requires determination of values of d and c first. Before we define these parameters, observe that S is a real valued matrix, regardless of whether the underlying graph is directed or undirected. This implies that S has either real symmetric or conjugate complex eigenvalues, i.e., if $5i$ is an eigenvalue of S, so is $-5i$. Therefore, we can fix the center of the ellipse onto the real line. This suggests

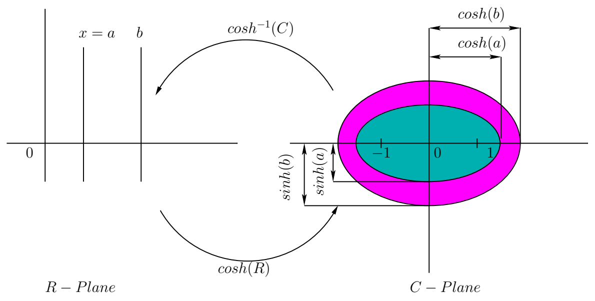

Figure 2: **Illustration of Joukowski mapping to obtain Chebyshev polynomial over complex plane.** The Chebyshev polynomial in complex plane is give by $T_t(z) = cosh(tcosh^{-1}(z))$. The line segments $x = a$ and $x = b$ are mapped onto an ellipse by $cosh$ map. Here, the ellipse has semi-major axes $|cosh(x)|$, semi-minor axes $|sinh(x)|$ at foci -1 and 1. This mapping is also called Joukowski transformation, and can be used to define Chebyshev polynomials in a complex plane.

that major axes of the ellipse must be real axes or parallel to imaginary axes as shown in Figure 3. (Note that there are cases in which S could have eigenvalues as $xi+y$ and $xi-y$, which causes major axes to be parallel to imaginary axes, we omit these possibilities in Figure 3 to avoid confusion. In such cases, all procedures presented here still apply, due to real arithmetic operations [26]).

The best ellipse enclosing the spectrum of S is obtained by the parameters $c = \dfrac{\lambda_{max} + \lambda_{min}}{2}$ and $d = \dfrac{\lambda_{max} - \lambda_{min}}{2}$ [31], where λ_{min} and λ_{max} are extreme eigenvalues of S. Moreover, the scaled and shifted asymptotically optimal Chebyshev polynomial that solves the min-max problem over the ellipse, is defined as follows [26]:

$$\hat{T}_t(z) = \frac{T_t(\frac{c-z}{d})}{T_t(\frac{c-\xi}{d})} \tag{9}$$

Here, ξ is any value that lies outside of the ellipse. In our case, for some small $\epsilon > 0$, we can safely take $\xi = 2 - \alpha + \epsilon$ or 0 since $\mathbf{P}$ is column normalized and $0 < \alpha < \|(\mathbf{I} - (1 - \alpha)\mathbf{P})\| < 2 - \alpha + \epsilon$. This holds for S matrix because its norm is always larger than 0. Since we know that the maximum is attained over the end point of an ellipse, then, for $z = c + a$, from Saad et al. [31], we have

$$\max_{z \in E(c,d,a)} \hat{T}_t(z) = \frac{T_t(\frac{a}{d})}{|T_t(\frac{c-\xi}{d})|}. \tag{10}$$

In the light of the above discussions, we can state the following theorem to describe the use of Chebyshev polynomials for solving a linear system involving S general matrix over the elliptic plane.

THEOREM 1. *Let λ_{min} and λ_{max} be the extreme eigenvalues of real-valued matrix S and $E(c, d, a)$ be an ellipse obtained by mapping $c = \dfrac{\lambda_{max} + \lambda_{min}}{2}$ and $d = \dfrac{\lambda_{max} - \lambda_{min}}{2}$ with cosh as shown above. Let ξ be 0 outside the ellipse.*

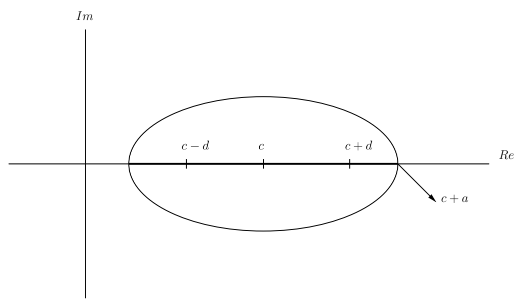

Figure 3: **Illustration of the ellipse that encloses the eigenvalues of S perfectly in the complex plane.** This ellipse is obtained by mapping changed and shifted parameters, c and d, through the Joukowski transformation.

Algorithm 2: The Arnoldi Algorithm

```
   Method: $Arnoldi(S, t, \mathbf{l}^{(0)})$
   Input  : $S$, $t$, and $\mathbf{l}^{(0)}$
   Output : $\mathbf{Q}_{t+1}$ and $\mathbf{H}_{t+1,t}$
 1 $q_1 \leftarrow \frac{\mathbf{l}^{(0)}}{\|\mathbf{l}^{(0)}\|_2}$, $j \leftarrow 1$
 2 while $t > j$ do
 3     $z \leftarrow Sq_j$
 4     for $i = 1$ to $j$ do
 5         $h_{i,j} \leftarrow {q_i}^T z$
 6         $z \leftarrow z - h_{i,j} q_i$
 7     end
 8     $h_{j+1,j} \leftarrow \|z\|_2$
 9     if $(h_{j+1,j} == 0)$ then
10         break
11     end
12     $q_{j+1} \leftarrow \frac{z}{h_{j+1,j}}$
13     $j \leftarrow j + 1$
14 end
```

Then,

$$\underset{p_t \in \mathbb{P}_t, p_t(0)=1}{argmin} \; \underset{z \in E(c,d,a)}{max} \; p_t(\Lambda(S)) \leq \frac{T_t(\frac{a}{d})}{|T_t(\frac{c}{d})|}$$

PROOF. Detailed proof can be found in Gander et al. [18] □

Notice that in order to define the perfect ellipse and Chebyshev polynomial over it, we must know the extreme eigenvalues of S, λ_{min}, and λ_{max}. In the following section, we explain how these eigenvalues can be efficiently computed using the well-known Arnoldi method [13]. Note that this computation is performed during preprocessing, and the eigenvalues are indexed in the database.

3.4.3 *Finding Extreme Eigenvalues Using the Arnoldi Method*

The Arnoldi method (Algorithm 2 [13]) approximates eigenpairs (eigenvalue and its corresponding eigenvector) from the reduced Hessenberg form by enforcing Petrov-Galerkin condition in which the eigenvectors must be in the Krylov space. Formally, the algorithm takes the matrix vector product for S, initial normalized random vector $\mathbf{l}^{(0)}$, and Krylov subspace dimension, t, and produces an $n \times (t+1)$ orthonormal column matrix $\mathbf{Q}_{t+1}$ and $(t+1) \times t$ upper Hessenberg matrix $\mathbf{H}_{t,t}$ such that ${Q_t}^T S Q_t = \mathbf{H}_{t,t}$ holds. Here the columns of Q_t form an orthonormal basis for Krylov subspace:

$$\mathcal{K}_t(S, \mathbf{l}^{(0)}) = \quad span\left\{\mathbf{l}^{(0)}, ..., \mathbf{S}^t \mathbf{l}^{(0)}\right\} \tag{11}$$

The eigenvalues of $\mathbf{H}_{t,t}$ approximate the top eigenvalues of S and corresponding top eigenvectors can be written as:

$$Q_t \tilde{V},$$

where $\mathbf{H}_{t,t} = \tilde{V}\tilde{\Lambda}\tilde{V}^T$.

The Arnoldi method uses the matrix vector product without requiring computation of the S matrix [13]. This method is particularly useful when it is used with a preconditioner, since it does not require computation of matrix-matrix multiplications, and the entire computation can be carried out using matrix-vector multiplications [13].

Algorithm 3: The I-CHOPPER Algorithm

```
   Input  : $S, {U_1}^{-1}, {L_1}^{-1}, U, L, M_{21}, M_{12}, c, d, \sigma_{min}, \sigma_{max}$,
            $q$, $k$, $\alpha$, and $\mathbf{s}_q$, $\epsilon$
   Output: a set $R \subseteq \mathbf{V}$ that contains the top-$k$ most
            proximate nodes to $q$
 1 Partition vector $q$ into $b_1$ and $b_2$
 2 Set $f \leftarrow (b_2 - M_{21}({U_1}^{-1}({L_1}^{-1} b_1)))$
 3 $R \leftarrow$ nodes in $S$, $t \leftarrow 1$,$\mathbf{y}_2^{(0)}, \Delta\mathbf{y}_2^{(0)} \leftarrow \mathbf{0}$
 4 $\mathbf{r}_2^{(0)} \leftarrow f - S\mathbf{y}_2^{(0)}$
 5 while $||\mathbf{r}_2^{(t)}||_2 \leq \epsilon$ do
 6     $z \leftarrow U \backslash L \backslash \mathbf{r}_2^{(t)}$
 7     $\beta \leftarrow (\frac{d\zeta}{2})^2$, $\zeta \leftarrow \frac{1}{c - \frac{\beta}{\zeta}}$, $\mathbf{y}_2^{(t)} \leftarrow \mathbf{y}_2^{(t-1)} + \zeta\Delta\mathbf{y}_2^{(t-1)}$
 8     $\mathbf{r}_q^{(t)} \leftarrow f - S\mathbf{y}_2^{(t)}$, $\Delta\mathbf{y}_2^{(t)} = z + \beta\Delta\mathbf{y}_2^{(t-1)}$
 9     if $\frac{\sigma_{max}}{\sigma_{min}}\mu^t < \epsilon$ then
10         break;
11     end
12 end
13 Compute $x_1 \leftarrow {U_1}^{-1}({L_1}^{-1}(b_1 - M_{12}y_2))$ and merge $x_1$
       and $y_2$ as $x_q$;
14 Sort and retrieve top-$k$ from $x_q$;
```

3.4.4 *Implementation of Chebyshev Acceleration over the Elliptic Plane*

Having computed the extreme eigenvalues of S using the Arnoldi method, we now present an efficient method for computing $\mathbf{y}_2^{(t)} = \mathbf{y}_2^{(t-1)} + \sum_{m=0}^{t-1} \gamma_t(m)\mathbf{r}_2^{(m)}$. The definition of $\mathbf{y}_2^{(t)}$ suggests that its computation requires storage of t additional residual vectors in memory at the tth iteration, throughout the power iteration. However, based on the observation that Chebyshev polynomials satisfy a three-term recurrence, we can compute $\mathbf{y}_2^{(t)}$ using only three vectors.

Let λ be an arbitrary eigenvalue of S inside or on the boundary of the ellipse. Then, tth iteration, from Lemma 2 and definition of optimal Chebyshev polynomial, we have $\hat{T}_t(\lambda) = \frac{T_t(\frac{c-\lambda}{d})}{T_t(\frac{c}{d})}$ and $\mathbf{e}^{(t)} = p_t(S)\mathbf{e}^{(0)}$. Now, we have

$\Delta\mathbf{y}_2^{(t)} = \mathbf{y}_2^{(t+1)} - \mathbf{y}_2^{(t)} = (\hat{T}_t(S) - \hat{T}_{t+1}(S))\mathbf{e}^{(0)}$ from last two equations (note that λ and S can be used interchangeably, since S is a diagonalizable matrix for $\alpha > 0$).

Finally, for $t > 0$, we can use the above equation, three-term recurrence of the Chebyshev polynomial, $\mathbf{r}_2^{(t)} = S\mathbf{e}^{(t)}$, and basic algebraic manipulations to obtain [32]:

$$\begin{aligned}\Delta\mathbf{y}_2^{(t)} &= \frac{2}{d}\left[\frac{T_t(\frac{c}{d})}{T_{t+1}(\frac{c}{d})}\right] S\hat{T}_t(S)\mathbf{e}^{(0)} \\ &+ \left[\frac{T_{t-1}(\frac{c}{d})}{T_{t+1}(\frac{c}{d})}\right](\hat{T}_{t-1}(S) - \hat{T}_t(S))\mathbf{e}^{(0)} \qquad (12) \\ &= \frac{2}{d}\left[\frac{T_t(\frac{c}{d})}{T_{t+1}(\frac{c}{d})}\right]\mathbf{r}_2^{(t)} + \left[\frac{T_{t-1}(\frac{c}{d})}{T_{t+1}(\frac{c}{d})}\right]\Delta\mathbf{y}_2^{(t-1)}.\end{aligned}$$

In other words, we can compute $\mathbf{y}_2^{(t+1)}$ by storing only $\mathbf{y}_2^{(t)}, \Delta\mathbf{y}_2^{(t)}$, and $\mathbf{r}_2^{(t)}$.

Based on the above derivations and the definition of Chebyshev polynomial over the complex elliptic plane, we can construct I-Chopper as shown in Algorithm 3, and use the following theorem to characterize the rate of convergence:

Theorem 2. *Let σ_{min} and σ_{max} be minimum and maximum singular values of S. Then, in the $t-th$ step, of* I-Chopper *we have*
$$\frac{\left\| f - S\boldsymbol{y}_2^{(t)}\right\|_2}{\|f\|_2} \le \frac{\sigma_{max}}{\sigma_{min}}\left(\frac{a+\sqrt{a^2-d^2}}{c+\sqrt{c^2-d^2}}\right)^t.$$

Proof. Let $Z\Lambda Z^{-1}$ be Jordan form of S, where Λ contains the eigenvalues of S in its diagonal, and columns of Z contain the corresponding eigenvectors. Then, from Lemma 2, we have
$$\begin{aligned}\min_{x_2\in\mathcal{K}_t(S,f)} \| f - Sy_2\| &= \min_{p_t\in\mathbb{P}_t, p_t(0)=1} \left\| Zp_t(\Lambda)Z^{-1}f\right\|_2 \\ &\le \min_{p_t(0)=1} \max_{\Lambda\in E(c,d,a)} \left\| ZZ^{-1}\right\| \| p_t(\Lambda)\| \|f\| \\ &\le \| Z\| \| Z^{-1}\| \frac{T_t(\frac{a}{d})}{|T_t(\frac{c}{d})|}\|f\|\end{aligned}$$

Now, from the definition of condition number, we know that $\| Z\| \left\| Z^{-1}\right\|_2 = \frac{\sigma_{max}}{\sigma_{min}}$. From Gander et al. [18], and the definition of shifted and scaled Chebyshev polynomial for $c > a > d$ we have:
$$\frac{T_t(\frac{a}{d})}{|T_t(\frac{c}{d})|} = \left(\frac{a+\sqrt{a^2-d^2}}{c+\sqrt{c^2-d^2}}\right)^t.$$
This completes the proof. □

In practice, note that an incomplete LU preconditioner is used at Line 6 of I-Chopper to elliminate the effect of $\| Z\|_2$ for general S matrix, i.e., eigenvector deflation. In this setting, the ILU preconditioner serves to minimize the effect of eigenvectors being non-orthogonal.

3.4.5 *Time Complexity Analysis*

The runtime of the proposed indexing and querying techniques can be highly variable, depending on the structural properties of the network besides its size. Nevertheless, for completeness, we provide worst-case runtime analyses for these algorithms here. In the next section, we also report results from comprehensive empirical studies, with a view to characterizing the indexing and query processing times of I-Chopper on a broad range of networks with variable properties.

Theorem 3. *Let $|E|$ denote the number of non-zero entries in $\boldsymbol{M}$ and p denote the user-defined parameter that tunes the ratio of the number of nodes in the dense component to the number of nodes in the network (the inputs to PartGraphRecursive). Let b denote the number of blocks in M_{11}, n_1 and n_2 respectively denote number of rows of M_{11} and M_{22} (the outputs of PartGraphRecursive), $|E_{12}|$ and $|E_{21}|$ denote the number of non-zero entriesin M_{12} and M_{21} (also outputs of textitPartGraphRecursive), and $|E_S|$ denote the number of non-zero entries in S (constructed based on the output of PartGraphRecursive). Then, the time required for pre-processing and building the index in* I-Chopper *takes*
$$\mathcal{O}(p|E|log|E| + \sum_{k=1}^{b} {n_1}_k{}^3 + |E_{21}|\sum_{k=1}^{b} {n_1}_k + (T_1+T_2)|E_S|$$
time, where T_1 and T_2 denote the number of iterations performed by the Arnoldi and Lanczos Methods.

Proof. Partitioning $\mathbf{M}$ using the *PartGraphRecursive* graph partitioning tool in Metis takes $\mathcal{O}(p|E|log|E|)$ time [22]. Inverting the block diagonal matrices in M_{11} takes $\mathcal{O}(\sum_{k=1}^{b} {n_1}_k{}^3)$ total time in the worst-case [13]. Constructing the matrix S takes $\mathcal{O}(|E_{21}|\sum_{k=1}^{b} {n_1}_k)$ time, since it involves multiplication of M_{21} with each of the blocks in M_{11}. Once S is computed, application of Arnoldi and Lanczos algorithms on S respectively take $\mathcal{O}(T_1|E_S|)$ and $\mathcal{O}(T_2|E_s|)$ time [13]. □

Theorem 4. *The worst-case runtime of processing a query n* I-Chopper *is*
$$\mathcal{O}(\sum_{k=1}^{b} {n_1}_k{}^2 + |E_{12}| + |E_{21}| + T_3|E_S|)$$
where T_3 denotes number of iterations performed by the Chebyshev acceleration.

Proof. Multiplying a vector with ${U_1}^{-1}$ and ${L_1}^{-1}$ takes $\mathcal{O}(\sum_{k=1}^{p} {n_1}_k{}^2)$ for all block-diagonal matrices in the worst-case. Matrix-vector products involving M_{12} and M_{21} respestively takee $\mathcal{O}(|E_{12}|)$ and $\mathcal{O}(|E_{21}|)$ time. Finally, the iterative phase of query processing takes $\mathcal{O}(T|E_s|)$ [32]. □

Here, we remark that *ILU* serves the purpose of forcing eigenvectors of S to be orthonormal, i.e sparsifying M_{22}. Moreover, Chebyshev polynomial over complex elliptic plane determines the smallest possible T to span the Krylov space for a Neumann Expansion Series formula of a matrix inverse [32]. As a result, the number of iterations performed by I-Chopper during query processing is usually very low, enabling I-Chopper to drastically outperform state-of-art algorithms for top-k proximity querying.

3.5 Summary of the Methods

For a given network, we summarize the proposed methods for indexing and query processing below. Visual illustration of the methods is presented in Figure 1.

Table 1: Descriptive statistics of the network datasets used in the experiments and intermediary statistics of the data structures obtained during preprocessing and indexing each network.

Network	`Email-EU`	`Web-Stanford`	`Web-Google`	`Orkut`	`Cit-Patent`	`Live-Journal`	`Twitter`	`Friendster`
Number of Nodes	266K	282K	876K	3M	3.8M	4.9M	41.7M	65.7M
Number of Edges	420K	2.4M	5.1M	117.2M	16.6M	69M	1.5B	2.6B
Average Node Degree	1.58	8.2	5.82	39.06	4.37	14.08	35.9	38.06
Fraction of Nodes with Degree ≤ 2	94%	24%	43%	21%	70%	41%	77%	68%
Average Degree of 10 Largest Hubs	4277	21342	5071	24904	705	13843	7453	4294
Number of Nodes in the Sparse Component (n_1)	212K	226K	773K	1.8M	3.5M	2.9M	31.5M	49M
Number of Edges in Sparse Component	223K	912K	1.5M	26.7M	3.6M	18.7M	102M	217M
Number of Blocks in Sparse Component	208K	106K	512K	827K	2.79M	2.6M	16M	34M
Bandwidth(Largest Block Size) in Sparse Component	58	18643	7492	11390	1215	4723	14104	3461

Table 2: Runtimes of each operation in preprocessing and indexing each of the networks used in the experiments.

Network/Run time (sec)	`Graph Partitioning`	${L_1}^{-1}\&{U_1}^{-1}$	$ILU(S)$	`Arnoldi`	`Lanczos`
Email-EU	0.67	0.08	0.03	1.56	0.042
Web-Stanford	4.54	2.56	0.10	12.49	1.63
Web-Google	3.51	0.9	0.10	25.83	2.23
Cit-Patent	18.7	2.43	1.4	19.83	9.47
Orkut	97.2	6.46	7.43	38.34	18.21
Live-Journal	78.2	4.53	3.2	34.5	13.37
Twitter	5237.3	235.67	124.5	174.42	148.6
Friendster	4921.2	383.3	231.5	189.4	159.4

3.5.1 Indexing

We first construct $\mathbf{M}$ and use *PartGraphRecursive* in METIS to partition $\mathbf{M}$ in such a way that M_{11} is a sparse block diagonal matrix, and M_{22} is dense but smaller [21, 22]. Next, we reorder the entries of partitioned matrix $\mathbf{M}$ based on an approximate minimum degree ordering (AMD) [1, 12]. After reordering entries of $\mathbf{M}$, we invert the LU decomposition of sparse block-diagonal matrix M_{11} and obtain ${L_1}^{-1}$ and ${U_1}^{-1}$. Then, we construct $S = M_{22} - M_{21}{M^{-1}}_{11}M_{12}$, and perform incomplete LU decomposition as [L,U]=ILU(S). Subsequently, we use Arnoldi's method to find the maximum and minimum eigenvalues of S, and store the resulting values and matrices into an index to use them in query processing phase of our algorithm, I-CHOPPER .

3.5.2 Online Querying

In the query phase, for a given query node, q. We first construct restart vector $\mathbf{s}_q$ and reorder the entries of $\mathbf{s}_q$ using the same ordering of $\mathbf{M}$. Subsequently, we divide $\mathbf{s}_q$ into two parts, $\mathbf{s}_q = \begin{bmatrix} s_{q1} \\ s_{q2} \end{bmatrix}$, based on the partition of $\mathbf{M}$ and set $b_1 = \alpha s_{q1}$ and $b_2 = \alpha s_{q2}$. Next, we use the indexed matrices and extreme eigenvalues of S to compute x_2 in equation (4), the lower part of solution of the linear system, with ILU preconditioned Chebyshev acceleration. Here, the ILU preconditioner serves to refine norms of eigenvectors of S. Finally, using x_2 and the indexed matrices, we compute x_1, the upper part of solution of the linear system. We then merge the entries in x_1 and x_2 and return the nodes that correspond to the top-k entries in the resulting merged vector.

4. EXPERIMENTAL RESULTS

In this section, we systematically evaluate the runtime performance and scalability of the proposed algorithm, I-CHOPPER, in processing top-k proximity queries. As shown in the previous section, I-CHOPPER is "exact" in the sense that it is guaranteed to correctly identify the k nodes that are most proximate to a given query node. For this reason, we focus on computational cost (measured in terms of number of iterations and runtime) in our experiments and compare I-CHOPPER against other exact algorithms.

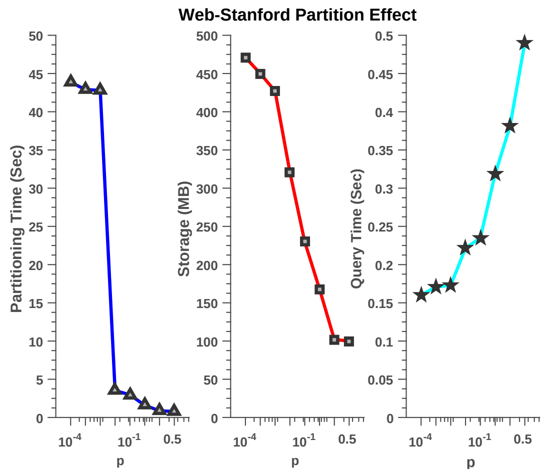

Figure 4: **The effect of the partition ratio in graph partitioning on the trade-off between indexing and query processing.** For values of the parameter p (the ratio of the number of nodes in the dense component to the number of nodes in the network) in graph partitioning ranging from 10^{-4} to 0.5, the left, middle, and right panels respectively show the pre-processing time, index size, and query processing time for the `Web-Stanford` dataset.

We start our discussion by describing the datasets and the experimental setup. We then assess the performance of the indexing phase, where we compute the index that is used in the query processing phase. Subsequently, we compare the performance of I-CHOPPER in processing top-k proximity queries with two state-of-the-art algorithms, K-DASH and CASTANET [14, 15]. For two undirected networks ,namely, `Orkut` and `Friendster` networks, we compare I-CHOPPER against CHOPPER [10] to assess the contribution of indexing.

4.1 Datasets and Experimental Setup

We use eight real-world network datasets from the Stanford Network Analysis Project [23]. Descriptive statistics of

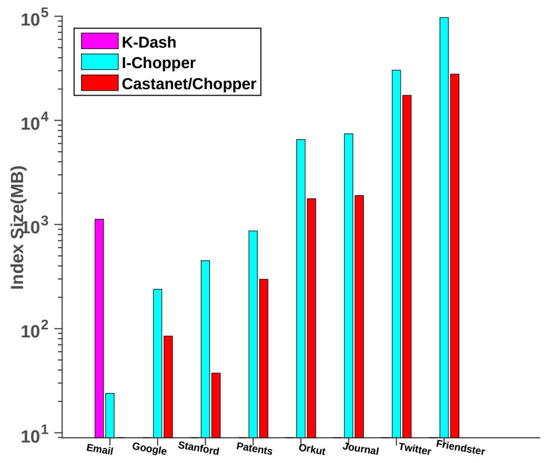

Figure 5: **Comparison of index size (in megabytes) for** K-Dash**,** I-Chopper**, and** Castanet**.** The plot shows total size of matrices required in the indexing phase for each algorithm. For K-Dash , the magenta bar shows the total storage needed for the inverse of LU of reordered $\mathbf{M}$. K-Dash cannot be applied to large datasets due to the computational cost of the indexing phase. Cyan bars show the index size for I-Chopper – total space needed to store ${L_1}^{-1}$, ${U_1}^{-1}$, $ILU(S)$, S, M_{21}, and M_{12} across all datasets. Red bars show the index sizes for Castanet and Chopper, which is simply the size of matrix $\mathbf{M}$ for each dataset.

these eight networks are provided in Table 1. The `Email-EU` dataset represents email data from a large European research institution. `Web-Stanford` and `Web-Google` datasets represent web graphs, where nodes represent web pages and edges represent hyperlinks between them. These datasets were released in 2002 by Stanford University and Google, respectively. The `Cit-Patent` network is derived from the U.S patent dataset in which nodes represent patents and edges represent citations between patents from 1963 to 1999. `Orkut` is a free online social network in which undirected edges represent friendships between users. `LiveJournal` is an online blogging community where directed edges represent friendships between users. `Friendster` is an online gaming network with undirected edges representing friendship among the users. `Twitter` is the directed network of following relationships between users in Twitter. These datasets are chosen as representative samples for network sizes in terms of the number of nodes and of edges. Furthermore, the network proximity querying problem is meaningful on these datasets.

For K-Dash and Castanet algorithms, we use the Matlab implementation downloaded from [29]. We implement I-Chopper in Matlab. We assess the performance of the algorithms for a fixed damping factor (restart probability $\alpha = 0.1$) and varying values of k for top-k proximity queries. In practice, using small α is recommended to fully utilize the information provided by the network [10]. For this reason, we fix α to a small value, 0.1 . For all experiments, we randomly select 1000 query nodes and report the average of the performance figures for these 1000 queries. In all experiments, $\mathbf{s}_q$ 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.

4.2 Indexing

The time required to perform each of the five preprocessing steps is shown in Table 2 for all eight datasets. In all experiments reported here, we use the *PartGraphRecursive* graph partitioning method in Metis [21,22] to identify the sub-networks.

In Figure 4, we report the effect of partitioning ratio, namely, $p = \frac{n_2}{n}$ on the `Web-Stanford` dataset. As it is expected, when this ratio is small, the query time goes down drastically, however, the storage requirement for the index and pre-processing time go up. This is because big portion of inverse of $\mathbf{M}$, ${U_1}^{-1}$ and ${L_1}^{-1}$ are obtained in preprocessing phase, thereby reducing query processing time at the cost of increasing the bandwidth of the sparse component (M_{11}), hence increasing the number of non-zeros in the inverse of this sparse matrix. Since there is a drastic jump in both preprocessing time and storage size around $p = 0.2$, while query time is relatively stable for values of p smaller than this value, we choose $p = 0.2$ to use in our experiments to obtain a reasonable balance between storage size and query processing time. We note here that the choice of this parameter is highly dependent on the characteristics of the network as well as the constraints of the application (e.g. available storage space, the need for real-time query processing etc.), but the analysis presented in Figure 4 can provide a useful starting point on setting this parameter for a given network.

Before we perform ILU or LU decomposition, we reorder the entries of M_{11} and M_{22} based on approximate minimum degree algorithms [1,12]. For ILU decomposition, we set the fill-in ratio to 1. Finally, for `Arnoldi` and `Lanczos` algorithms, we set maximum Krylov space's dimensions to 1000 and the restart parameter to 10. In our experiments, we observe that enlarging these parameters does not provide crucial "offline" performance change due to Gram-Schmidt orthonormalization cost hence we use these as default parameters.

In Figure 5, we report total index sizes for all datasets. I-Chopper's index requires less than 100 GB space for all datasets, including the largest dataset `Friendster`. Since the K-Dash algorithm [14] has cubic computational complexity, it cannot complete its preprocessing phase within 48 hours for very large datasets. Hence, in our experiments, we are only able to perform preprocessing for K-Dash on the `Email-EU` dataset and it requires 50 times storage than I-Chopper. Castanet and Chopper do not store an index, but they still need to store the stochastic matrix $\mathbf{M}$ to perform the proximity computation during query processing. As seen in the figure, the additional space needed to store the extra matrices for I-Chopper's index increases the total space needed by only a fraction of the size of this stochastic matrix, for all datasets except the `Web-Stanford` dataset. Furthermore, Chopper is only applicable to undirected networks and Castanet cannot be applied to billion-scale networks as we show in next section.

We also observe in Figure 5 that the index size required by I-Chopper is larger for networks with fewer loosely-connected nodes. For example, for Web-Stanford, the nodes with degree at most 2 comprise 24% of the network, a number much lower than any other network other than `Orkut`. This makes it difficult for the graph partitioning algorithm to compute a sparse component (M_{11}) with a small band-

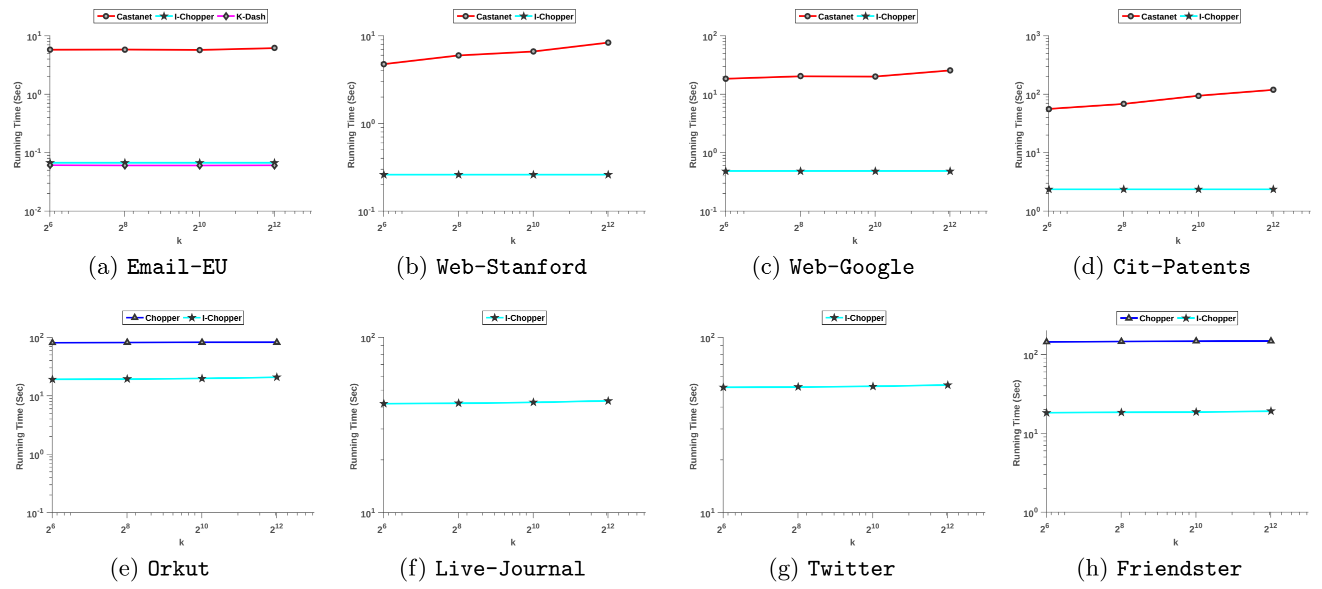

Figure 6: **Runtime in seconds of** I-CHOPPER **K-DASH, and** CASTANET **to process queries for top-k nodes that are most proximate to a query node, as a function of k ranging from 64 to 4096.** In these experiments, the damping factor α is set to 0.1 and the reported numbers are the averages across 1000 randomly chosen query nodes. K-DASH is only applied to smallest graph since the compute time of its indexing phase exceeded 48 hours for graphs larger than `Email-EU`. CASTANET cannot process single query for largest networks in 24 hours hence we eliminate it for very large networks. CHOPPER can only be applied to undirected networks.

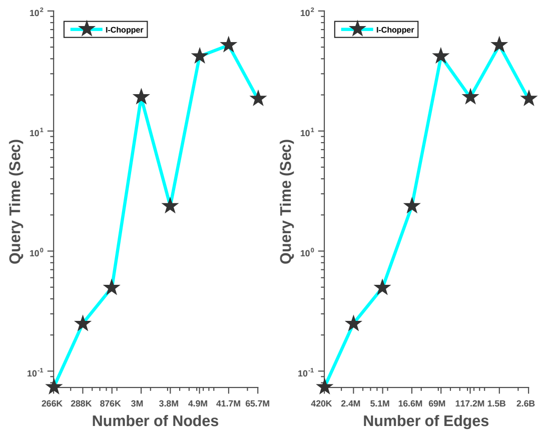

Figure 7: **Visualization of the effect of graph size on runtime for** I-CHOPPER**.** For the value of the parameter $k = 2^8$ (the top-k value), runtime of I-CHOPPER is shown as a function of the number of nodes (left panel) and number of edges (right panel) in the network.

width, thereby increasing the number of non-zeros in the inverse of this matrix. This results in a super-linear increase in index size. Indeed, the bandwidth (largest block size) of M_{11} for the `Web-Stanford` dataset is largest among all networks considered in our experiments, although this network is smaller than most of these networks. This effect is less pronounced for the `Orkut` network since this network is much denser than the `Web-Stanford` network, hence the size of ${M_{11}}^{-1}$ is much less as compared to the size of $\mathbf{M}$.

4.3 Runtime Performance

The performance of I-CHOPPER in comparison to three algorithms, K-DASH ,CASTANET , and CHOPPER as a function of k is shown in Figure 6. In the querying phase, K-DASH slightly outperforms I-CHOPPER for only the smallest dataset. However, its indexing phase cannot be applied to large datasets within 48 hours. CASTANET is more scalable than I-CHOPPER in terms of size of index. However, as seen in the figure, I-CHOPPER outperforms CASTANET achieving more than 25-fold speed-up for all directed networks for which CASTANET can be applied. Also, CASTANET cannot be applied to very large datasets since it processes a query in more than a day for these networks. Hence, we do not consider CASTANET for very large networks. The favorable performance of I-CHOPPER compared to CASTANET and K-DASH is particularly notable for larger graphs, since CASTANET has a large query processing time despite being scalable in terms of memory, and K-DASH cannot be applied to very large graphs due to memory limitations.

For undirected networks, `Orkut` and `Friendster`, we compare I-CHOPPER against CHOPPER [10] to assess the contribution of indexing approach implemented in I-CHOPPER . As seen in the figure, indexing in I-CHOPPER allows us to process billion-scale networks in real time and achieves more than 8-fold speed-up over CHOPPER for the underincted networks.

Finally, we assess the performance of I-CHOPPER as a function of graph size (based on number of nodes and edges) across all datasets in Figure 7. In this experiment, we fix $K = 2^8$. We observe that bandwidth (largest block in the sparse component) has an enormous effect on query processing time. For instance, query time for I-CHOPPER for

`Friendster` dataset is similar to that of `Orkut` dataset although `Friendster` is much larger than `Orkut` dataset. This is because thee bandwidth for `Orkut` is larger than that for `Friendster`.

5. CONCLUSION

In this paper, we present a novel hybrid approach to the processing of node proximity queries. The proposed approach uses graph partitioning based indexing coupled with a Chebyshev polynomial over complex elliptic plane for accelerating the iterative computation. We show that our approach, I-Chopper, yields significantly better convergence times than iterative methods, and much lower memory consumption than direct inversion, both in theory and in practice on real-world problems. Using a number of large real-world networks, and top-k proximity queries as the benchmark problem, we show that I-Chopper outperforms existing methods significantly in terms of scalability and runtime.

Future efforts in this direction include incorporation of other preconditioning techniques into our framework, extensions to other iterative proximity measures, and their applications. Since I-Chopper is an "exact" algorithm, our experiments only compare against other exact methods. There are other approximate methods that sacrifice accuracy for improved runtime. Comparison of I-Chopper against such approximate algorithms to precisely quantify accuracy-time tradeoffs would be an interesting application-specific investigation.

It is also worth to note that I-Chopper does not use the numerical properties of Chebyshev acceleration to achieve an early stop when it becomes certain that no node that is not in the top-k can go into the top-k anymore. This is because, it is difficult to obtain a provable bound on the residual due to the dependency between the proximity vectors that correspond to the two different components of the system (indexed vs. iteratively solved). Derivation of such a bound remains an open problem and would help further improve the performance of the proposed algorithms.

Acknowledgments

This work was supported in part by National Institutes of Health Grant U01-CA198941 from the National Cancer Institute and National Science Foundation Grant CSR-1422338.

6. REFERENCES

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *S*IAM Journal on Matrix Analysis and Applications, 17(4):886–905, 1996.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *F*oundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on, pages 475–486. IEEE, 2006.

[3] R. Andersen, F. Chung, and K. Lang. Using pagerank to locally partition a graph. *I*nternet Math., 4(1):35–64, 2007.

[4] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *P*VLDB, 4(3):173–184, 2010.

[5] R. E. Bank and J. Xu. The hierarchical basis multigrid method and incomplete lu decomposition. *C*ontemporary Mathematics, 180:163–163, 1994.

[6] P. Bogdanov and A. Singh. Accurate and scalable nearest neighbors in large networks based on effective importance. In *P*roceedings of the 22nd ACM international conference on Conference on information & knowledge management, pages 1009–1018. ACM, 2013.

[7] S. Boyd and L. Vandenberghe. *C*onvex optimization. Cambridge university press, 2004.

[8] M. Cao, H. Zhang, J. Park, N. M. Daniels, M. E. Crovella, L. J. Cowen, and B. Hescott. Going the distance for protein function prediction: a new distance metric for protein interaction networks. 2013.

[9] U. V. Catalyurek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *I*EEE Transactions on parallel and distributed systems, 10(7):673–693, 1999.

[10] M. Coskun, A. Grama, and M. Koyuturk. Efficient processing of network proximity queries via chebyshev acceleration. In *P*roceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 1515–1524, New York, NY, USA, 2016. ACM.

[11] M. Coskun and M. Koyutürk. Link prediction in large networks by comparing the global view of nodes in the network. In *D*ata Mining Workshop (ICDMW), 2015 IEEE International Conference on, pages 485–492. IEEE, 2015.

[12] T. A. Davis, P. Amestoy, and I. S. Duff. An approximate minimum degree ordering algorithm. *C*omp. Inform. Sci. Dept. TR-94-039, University of Florida, 1995.

[13] J. W. Demmel. *A*pplied numerical linear algebra. SIAM, 1997.

[14] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and exact top-k search for random walk with restart. *P*VLDB, 5(5):442–453, 2012.

[15] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Efficient ad-hoc search for personalized pagerank. In *P*roceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pages 445–456. ACM, 2013.

[16] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Efficient ad-hoc search for personalized pagerank. In *P*roceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 445–456, New York, NY, USA, 2013. ACM.

[17] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka. Efficient personalized pagerank with accuracy assurance. In *P*roceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 15–23. ACM, 2012.

[18] M. Gander, W. Gander, G. Golub, and D. Gruntz. Scientific computing: An introduction using matlab, 2005.

[19] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang.

Manifold-ranking based image retrieval. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 9–16. ACM, 2004.

[20] V. Hristidis, Y. Wu, and L. Raschid. Efficient ranking on entity graphs with personalized relationships. *IEEE Transactions on knowledge and data engineering*, 26(4):850–863, 2014.

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

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

[23] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. snap.stanford.edu/data.

[24] R. Li and Y. Saad. Low-rank correction methods for algebraic domain decomposition preconditioners. *arXiv preprint arXiv:1505.04341*, 2015.

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

[26] T. A. Manteuffel. The tchebychev iteration for nonsymmetric linear systems. *Numerische Mathematik*, 28(3):307–327, 1977.

[27] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 469–478. ACM, 2008.

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

[29] J. Ni, H. Tong, W. Fan, and X. Zhang. Inside the atoms: Ranking on a network of networks. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1356–1365, New York, NY, USA, 2014. ACM.

[30] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, November 1999. Previous number = SIDL-WP-1999-0120.

[31] Y. Saad. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Mathematics of Computation*, 42(166):567–588, 1984.

[32] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[33] K. Shin, J. Jung, S. Lee, and U. Kang. Bear: Block elimination approach for random walk with restart on large graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1571–1585. ACM, 2015.

[34] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4(11):992–1003, 2011.

[35] T. Tao and C. Zhai. An exploration of proximity measures in information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 295–302. ACM, 2007.

[36] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 404–413. ACM, 2006.

[37] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 613–622, Washington, DC, USA, 2006. IEEE Computer Society.

[38] O. Vanunu, O. Magger, E. Ruppin, T. Shlomi, and R. Sharan. Associating genes and protein complexes with disease via network propagation. *PLoS Comput. Biol.*, 6(1):e1000641, Jan 2010.

[39] Y. Wu, R. Jin, and X. Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1139–1150. ACM, 2014.

[40] W. Yu and X. Lin. Irwr: incremental random walk with restart. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 1017–1020. ACM, 2013.

[41] C. Zhang, S. Jiang, Y. Chen, Y. Sun, and J. Han. Fast inbound top-k query for random walk with restart. In *Machine Learning and Knowledge Discovery in Databases*, pages 608–624. Springer, 2015.

[42] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei. Evaluating geo-social influence in location-based social networks. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1442–1451. ACM, 2012.

[43] J. Zhao and Y. Yun. A proximity language model for information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 291–298. ACM, 2009.