Efficient Processing of Network Proximity Queries via Chebyshev Acceleration

Mustafa Coşkun
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH 44106, USA.
mustafa.coskun@case.edu

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

Mehmet Koyutürk
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH 44106, USA.
mehmet.koyuturk@case.edu

ABSTRACT

Network proximity is at the heart of a large class of network analytics and information retrieval techniques, including node/ edge rankings, network alignment, and random-walk based proximity queries, among many others. Owing to its importance, significant effort has been devoted to accelerating iterative processes underlying network proximity computations. These techniques rely on numerical properties of power iterations, as well as structural properties of the networks to reduce the runtime of iterative algorithms.

In this paper, we present an alternate approach to acceleration of network proximity queries using Chebyshev polynomials. We show that our approach, called CHOPPER, yields asymptotically faster convergence in theory, and significantly reduced convergence times in practice. We also show that other existing acceleration techniques can be used in conjunction with CHOPPER to further reduce runtime. Using a number of large real-world networks, and top-k proximity queries as the benchmark problem, we show that CHOPPER outperforms existing methods for wide ranges of parameter values. CHOPPER is implemented in Matlab and is freely available at http://compbio.case.edu/chopper/.

Keywords

Network proximity, Random walk with restarts, Chebyshev polynomials

1. INTRODUCTION

Proximity measures on networks are at the core of a large number of analytics and information retrieval techniques. In information retrieval, nodes/ edges are ranked based on their random walk distance from other nodes [18,25]. In network alignment, high scoring node alignments (node pairs – one drawn from each network) can be identified by their random walk distance from other high scoring alignments in the product graph of the two input networks [10]. In disease-gene prioritization, genes are ranked according to their network proximity to genes that are previously identified as associated with clinically similar diseases [13].

The general setting for network proximity queries is as follows: For a given query node, we are interested in computing a score for each node that indicates the proximity of that node to the query node. For example, shortest path queries ask for the minimum number of hops (or minimum total weight of edges) connecting two nodes. Random walk based proximity measures, on the other hand, simulate random walks that make frequent restarts on the query node, and estimate proximity to the query node in terms of the probability of being at each node at steady state. In top-k proximity queries, the k closest nodes to a specified source node are returned.

In many applications, random walk based proximity is preferred over shortest path distance because of its robustness to the inherent noise in the network. This noise may be due to inaccuracies in modeling (not all interactions in the underlying system are modeled in the graph) or noise in data (missing or spurious edges). In such cases, random walk based proximity measures provide a more robust estimate of network proximity.

Random walk based proximity measures have been used in a wide variety of applications, including web search [14], link prediction [6,11], clustering [2], disease-gene prioritization [7,13], and integration of disparate "omic" data in systems biology [9]. Some of the well known random walk based proximity measures include discounted hitting time [17], personalized hitting probability [24], network propagation [20], diffusion state distance [5], and random walk with restarts (RWR) [1,19].

Motivated by widespread use of random-walk proximity, significant effort has been devoted to reducing operation counts associated with computation of proximity. These efforts exploit numerical and structural characteristics of the problem to reduce operation counts. For instance, in the context of the top-k proximity queries, during the iterative procedure, if it can be guaranteed that some nodes cannot enter into the top-k set any further, the associated computations can be eliminated [23]. Likewise, owing to the structure of the network, the proximity of each node to the query node can be bounded systematically by considering nodes in a breadth-first-like order [21]. These techniques have been

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939828

Table 1: Notation used in the description of methods.

Symbols	Meaning
$\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{w})$	Undirected weighted graph $\mathbf{G}$ with node set $\mathbf{V}$, edge set $\mathbf{E}$, and weight function $\mathbf{w}$
$\mathbf{P}$	Stochastic matrix derived from the adjacency matrix of $\mathbf{G}$
α	Restart probability (damping factor)
$\mathbf{W}$	Transpose of Stochastic matrix adjusted by the damping factor as $\mathbf{W} = (1-\alpha)\mathbf{P}$
q	The query node
$\mathbf{x}_q^*$	Vector of proximities to q computed by random walk with restarts
$\mathbf{r}_q$	Restart vector used to compute proximity to q
$\mathbf{y}_q^{(t)}$	Linear combination of $\mathbf{x}_q{}^{(t)}$ used for Chebyshev iterations
$\mathbf{e}^{(t)}$	Residual between $\mathbf{y}_q^{(t)}$ and $\mathbf{x}_q^*$
π_t	Polynomial of order t
$\mathbb{P}_t$	Family of polynomials of order t
T_t	Chebyshev polynomial of first kind
p_t	Polynomial that minimizes the norm of residual for iterative computation of random walk with restart
μ	Variable that characterizes the convergence rate of Chebyshev acceleration
$\rho(\mathbf{W})$	Largest eigenvalue of matrix $\mathbf{W}$
$\sigma(\mathbf{W})$	All eigenvalues of matrix $\mathbf{W}$

demonstrated to yield significant improvement in runtime in the context of diverse applications.

In this paper, we take an alternate approach, based on accelerating the convergence of the underlying iterative process. To achieve this, we adapt a result from classical linear algebra, based on Chebyshev polynomials. Traditional iterative procedures use the result from one iteration to compute the next iterate. The iterations are terminated when convergence is detected, i.e., when the proximity vector does not change significantly between two iterations. We can, however, define the next iterate as a linear combination of a predefined set of previous iterates. The coefficients of this linear combination can be optimally computed using Chebyshev polynomials. We demonstrate this process and show that the resulting the iterate converges much faster than the iterate in the original formulation. This results in significant speed-up in the computation of random walk based proximity scores. Furthermore, we show that our method can be combined with existing methods to further improve the processing of top-k proximity queries.

We provide detailed theoretical justification for our results, and experimentally demonstrate the superior performance of our method on a number of real-world benchmark problems. Specifically, we show that (i) our method yields significant performance improvements over state of the art methods (reducing the number of iterations required to compute network proximity to all nodes many-fold on networks with hundreds of thousands of edges); and (ii) for top-K proximity queries, our method yields two-fold improvement in runtime on graphs with millions of nodes and edges. The asymptotic acceleration of our scheme implies that this performance improvement increases as problem sizes are scaled up.

The rest of the paper is organized as follows: in the next section, we provide an overview of the literature on efficient computation of network proximity. In Section 3, we describe the terminology, establish background on random walk proximity and top-k proximity queries, and describe our method. In Section 4, we provide detailed experimental evaluation of our method. We draw conclusions and summarize avenues for further research in Section 5.

2. RELATED WORK

Network proximity querying has received significant attention over the past years. In particular, top-k proximity queries in networks involve identifying the k nodes that are in close (random walk) proximity to a query node (or a set of query nodes). One of the basic approaches to computing random walk based proximity is the power iteration [16]. An alternate approach to power iterations is offline computation through LU decomposition and storing the factors for proximity estimation during online query processng [8, 19, 22]. However, LU decomposition is expensive and usually not feasible for very large networks. Furthermore, since the underlying networks are often dynamic, even small perturbations to the network require repetition of this costly procedure.

Recently, a number of approaches have been proposed to scale top-k proximity queries to very large and sparse networks. These methods take advantage of the numerical properties of the iterative methods to bound the proximity of nodes in the network in early iterations, thereby stopping the computation early when only k contenders are left [23]. Other methods utilize the structure of the network to perform a local search around the query node(s), based on the notion that nodes with high random-walk based proximity to the query node are also in close neighborhood of the query node in terms of the number of hops [3, 4, 12, 24]. However, most of these local search based methods are approximate, in the sense that they do not provide guarantees for identifying the exact set of of k nodes that are most proximate to the query node. Recently, two local search based methods, FLoS [21] and Ripple [23] have been shown to enable exact computation of top-k proximity queries without compromising efficiency.

In this paper, we also focus on exact computation of network proximity. Our method is fundamentally different from existing approaches in that it exploits the numerical properties of the iterative procedure to speed-up its computation. This method can be used to efficiently compute the random walk with restarts based proximity of all nodes in the network to a given node, or to speed up the processing of top-k proximity queries.

3. METHODS

In this section, we first define random walk with restarts (RWR) and and top-K network proximity queries based on RWR. We then present insights from numerical linear algebra to motivate the use of polynomials for accelerating convergence of iterative procedures used to compute RWR based proximity. Subsequently, we show that Chebyshev polynomials can be used to optimize the convergence of iterative computation of RWR, and bound the relative error in each iteration. Finally, we discuss how these bounds can be used to efficiently process top-K network proximity queries. We conclude this section by showing that our method generalizes to any proximity measure that can be iteratively computed, for which error in each iteration is bounded.

3.1 RWR-Based Proximity

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{w})$ represent a given network, directed or undirected, where $\mathbf{V}$ denotes the set of nodes, $\mathbf{E}$ denotes the set of edges, and $\mathbf{w} : \mathbf{E} \rightarrow \mathbb{R}$ denotes the weight function assigned to the edges. Given a node $q \in \mathbf{V}$, the random walk with restart based proximity to q is defined as follows:

$$\mathbf{x}_q^* = (1-\alpha)\mathbf{P}\mathbf{x}_q^* + \alpha \mathbf{r}_q. \tag{1}$$

Here, $\mathbf{P}$ denotes the column-normalized stochastic matrix derived from the adjacency matrix of $\mathbf{G}$ by dividing each entry by the corresponding column sum, $\mathbf{r}_q$ denotes the restart vector that contains a 1 at its qth entry and a 0 in all other entries, and $0 < \alpha < 1$ denotes the damping factor. This parameter determines the probability of restarting at q in a random walk of the network. 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.

While we focus on random walk with restarts here for clarity of discussion, the method described in this paper directly applies to any proximity measure that can be written in the form $\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{c}$, where $\mathbf{A}$ is a matrix with largest eigenvalue $\rho(\mathbf{A}) < 1$ and $\mathbf{c}$ is constant vector. Such proximity measures include penalized hitting probability [24], effective importance [4], and discounted hitting time [17].

In practice, $\mathbf{x}_q$ is computed iteratively, by setting ${\mathbf{x}_q}^{(0)} = \mathbf{0}$ and performing the following computation:

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

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. As we discuss below, the residual in this iterative procedure is proportional to the tth power of the largest eigenvalue of the matrix $(1-\alpha)\mathbf{P}$. Therefore the iterative procedure is guaranteed to converge since the largest eigenvalue of $\mathbf{P}$ is equal to 1 and $(1-\alpha) < 1$. Throughout this paper, we refer to this procedure as the standard power iteration.

3.2 Top-k Proximity Queries

Given undirected network $\mathbf{G} = (\mathbf{V},\ \mathbf{E},\ \mathbf{w})$, a query node $q \in \mathbf{V}$, and a positive number k, the top-k proximity query for RWR proximity returns the k nodes in $\mathbf{V}$ corresponding to the largest values in $\mathbf{x}_q^*$ [23]. Note that the largest values correspond to the highest proximity (smallest RWR distance). The current state of the art in efficiently processing top-k proximity queries is based on two approaches. The first approach uses the bound on the residual in the iterative computation of $\mathbf{x}_q^*$ to eliminate nodes whose proximity values cannot exceed that of the notes that are already among the k. This process terminates when all but the top k nodes are eliminated. This approach improves efficiency by reducing the number of iterations, and is implemented by the SQUEEZE algorithm developed by Zhang et al. [23]. The second approach uses network structure, in combination with the residual, to bound the proximity of each node to the query node and avoids computing proximity scores for nodes that are sufficiently distant from the query node in terms of the number of hops. This approach improves efficiency by further reducing the number iterations, as well as the number of operations in each iteration. It was proposed by Wu et al. [21] and is also implemented in the RIPPLE algorithm developed by Zhang et al. [23].

All of these methods are based on the standard power iterations, i.e., the computation of the current vector iterate only uses its value from the previous iteration. Here, we show that the convergence rate of the iterative procedure can be significantly improved by utilizing the information gathered from all previous iterations. Specifically, we use Chebyshev polynomials to aggregate the values of $\mathbf{x}_q$ across iterations, obtaining a better approximation to the steady state vector in each iteration. This results in faster convergence, which in turn also yields more effective pruning of nodes. The number of iterations required for such a procedure to process top-k proximity queries is consequently much lower.

3.3 The CHOPPER Algorithm

The core idea behind the proposed **Ch**ebyshev **Po**lynomial Based **E**fficient **P**roximity **R**etrieval (CHOPPER) algorithm is to utilize the previously computed ${\mathbf{x}_q}^{(t)}$ vectors in Equation (2) to obtain a better approximation to $\mathbf{x}_q^*$ in the next iteration. To describe this idea, for a given query node q, we first define $\mathbf{W} = (1-\alpha)\mathbf{P}$ and rewrite the (2) as follows:

$${\mathbf{x}_q}^{(t+1)} = \mathbf{W}{\mathbf{x}_q}^{(t)} + \alpha \mathbf{r}_q. \tag{3}$$

Observe that, since $\mathbf{P}$ is a stochastic matrix, we have $|\rho(\mathbf{W})| \leq (1-\alpha) < 1$, hence the iterative procedure described by (3) converges to the solution of (1), $\mathbf{x}_q^*$.

The idea behind Chebyshev acceleration is as follows: For the tth iteration of the iterative procedure, and a given series $\gamma_t(m)$ for $0 \leq m \leq t$, we define vector $\mathbf{y}_q^{(t)}$:

$$\mathbf{y}_q^{(t)} = \sum_{m=0}^{t} \gamma_t(m){\mathbf{x}_q}^{(m)} \tag{4}$$

Our objective is to choose a sequence γ_t such that the sequence $\mathbf{y}_q^{(t)}$ converges to $\mathbf{x}_q^*$ faster than ${\mathbf{x}_q}^{(t)}$, and as rapidly as possible. Observe that, if we use $\mathbf{y}_q^{(t)}$ to approximate $\mathbf{x}_q^*$, the residual in the tth iteration is given by:

$$\mathbf{e}^{(t)} = \mathbf{y}_q^{(t)} - \mathbf{x}_q^*. \tag{5}$$

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 the linear combination also converges to $\mathbf{x}_q^*$.

We can reformulate the residual at the tth iteration as a

Algorithm 1: The Chopper Algorithm

```
Input  : $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, $q$, $k$, $\alpha$, and $\mathbf{r}_q$
Output: a set $R \subseteq \mathbf{V}$ that contains the top-$K$ most
        proximate nodes to $q$
1  Construct matrix $\mathbf{W}$
2  $R \leftarrow \mathbf{V}$, $t \leftarrow 1$, $\mathbf{y}_q^{(0)} \leftarrow \mathbf{0}$
3  $\mathbf{y}_q^{(1)} \leftarrow A\mathbf{x}_q{}^{(0)} + \alpha\mathbf{r}_q$
4  $\kappa \leftarrow \frac{2-\alpha}{\alpha}$, $\mu \leftarrow \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$
5  $\zeta_0 \leftarrow 1, \zeta_1 \leftarrow \frac{1}{1-\alpha}$
6  while $|R| > k$ do
7      $\zeta_{t+1} \leftarrow \frac{2}{1-\alpha}\zeta_t - \zeta_{t-1}$
8      $\mathbf{y}_q^{(t+1)} \leftarrow \frac{2\zeta_t}{(1-\alpha)\zeta_{t+1}}\left(\mathbf{W}\mathbf{y}_q^{(t)} + \alpha\mathbf{r}_q\right) - \frac{\zeta_{t-1}}{\zeta_{t+1}}\mathbf{y}_q^{(t-1)}$
9      $\tau \leftarrow$k−th largest score in $\mathbf{y}_q^{(t+1)}(u)$;
10     foreach $u \in R$ do
11         if $(\mathbf{y}_q^{(t+1)}(u) + 4\mu^t) < \tau$ then
12             Remove $u$ from $R$;
13         end
14     end
15 end
```

polynomial as follows:

$$\begin{aligned}
\mathbf{e}^{(t)} &= \mathbf{y}_q^{(t)} - \mathbf{x}_q^* \\
&= \sum_{m=0}^{t} \gamma_t(m)(\mathbf{x}_q{}^{(m)} - \mathbf{x}_q^*) \\
&= \sum_{m=0}^{t} \gamma_t(m)\mathbf{W}^m(\mathbf{x}_q{}^{(0)} - \mathbf{x}_q^*) \\
&= p_t(\mathbf{W})\mathbf{e}^{(0)}.
\end{aligned} \tag{6}$$

This observation suggests that, if we can find a sequence of polynomials p_t such that $\|p_t(\mathbf{W})\| \ll \|\mathbf{W}^t\|$, the error in each iteration is much smaller than that in the standard power iteration.

3.3.1 Chebyshev Polynomials

It is an established result in linear algebra that for any matrix $\mathbf{W}$ and polynomial p_t, we can write $\sigma(p_t(\mathbf{W})) = p_t(\sigma(\mathbf{W}))$, where $\sigma(.)$ denotes the set of eigenvalues of the matrix [15]. Therefore, as described by Saad [15], $||\mathbf{e}^{(t)}||$ can be minimized by solving the following minimization problem:

$$\min_{p_t \in \mathbb{P}_t, p_t(1)=1} \quad \max_{\lambda \in \sigma(\mathbf{W})} |p_t(\lambda)| \tag{7}$$

Here, $\mathbb{P}_t$ denotes the family of all polynomials of order t.

Clearly, computing the eigenvalues of $\mathbf{W}$ would defeat our purpose, since this computation is at least as expensive as solving (1). However, since $\mathbf{P}$ is column normalized, we know that $\|\mathbf{W}\| \leq \rho(\mathbf{W}) \leq (1-\alpha)$ [23]. Hence, we can relax (7), and rewrite it as:

$$\min_{p_t \in \mathbb{P}_t, p_t(\lambda)=1} \quad F(p_t) \tag{8}$$

where

$$F(p_t) = \max_{\lambda \in [-1+\alpha, 1-\alpha]} |p_t(\lambda)|. \tag{9}$$

As stated in the following theorem, the solution to minimax optimization problems of the form of (9) is provided by the well known Chebyshev polynomial of first kind, given by the following recurrence: $T_0(z) = 1, T_1(z) = z, T_{t+1}(z) = 2zT_t(z) - T_{t-1}(z)$.

Theorem 1. *Let $[a, b] \subseteq \mathbb{R}$ be non-empty interval and ξ be a real number such that $a < b < \xi$. Then,*

$$\underset{p_t \in \mathbb{P}_t, p_t(\xi)=1}{\mathit{argmin}} \quad F(p_t) = \frac{T_t(1 + 2\frac{\lambda - b}{b - a})}{T_t(1 + 2\frac{\xi - b}{b - a})} \quad = \pi_t(\lambda)$$

and

$$F(\pi_t) \quad = \frac{1}{T_t(1 + 2\frac{\xi - a}{b - a})} = 2\frac{\mu^t}{1 + \mu^{2t}}$$

where $\kappa = \frac{\xi - a}{\xi - b}$ *and* $\mu = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$.

Detailed proof of this theorem can be found in [16].

We can immediately apply this result to our problem. From the construction of $\mathbf{W}$, we know that $\rho(\mathbf{W}) = 1 - \alpha$, where $\rho(.)$ denotes the largest eigenvalue of a matrix. Then, letting $a = \alpha - 1, b = 1 - \alpha$, and $\xi = 1$, we have

$$p_t(\lambda) = \frac{T_t(\frac{\lambda}{1-\alpha})}{T_t(\frac{1}{1-\alpha})} \tag{10}$$

Therefore, using Theorem 1 and (6), we obtain

$$\begin{aligned}
\frac{||\mathbf{e}^{(t)}||}{||\mathbf{e}^{(0)}||} &\leq ||p_t(\mathbf{W})|| \leq \rho(p_t(\mathbf{W})) = \max_{\lambda \in \sigma(\mathbf{W})} |p_t(\lambda)| \\
&\leq \max_{\lambda \in [a,b]} |p_t(\lambda)| = 2\frac{\mu^t}{1 + \mu^{2t}} \leq 2\mu^t
\end{aligned} \tag{11}$$

where $\kappa = \frac{2-\alpha}{\alpha}$ and

$$\mu = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} = \frac{2(1-\alpha)}{2 + \sqrt{2\alpha - \alpha^2}} < (1 - \alpha). \tag{12}$$

This result demonstrates the power of Chebyshev polynomials in accelerating the computation of random walk with restart based proximity scores. Specifically, for standard power iteration, we have

$$\frac{\|x_q^{(t)} - \mathbf{x}_q^*\|}{\|x_q^{(0)} - \mathbf{x}_q^*\|} \leq \|\mathbf{W}^t\| \leq \rho^t = (1 - \alpha)^t. \tag{13}$$

In contrast, if we use Chebyshev acceleration, we have

$$\frac{\|y_q^{(t)} - \mathbf{x}_q^*\|}{\|y_q^{(0)} - \mathbf{x}_q^*\|} \leq \|p_t(\mathbf{W})\| = \rho(p_t(\mathbf{W})) \leq 2\mu^t. \tag{14}$$

Since $\mu < 1 - \alpha$, the sequence $y_q^{(t)}$ converges much faster than $x_q^{(t)}$ to $\mathbf{x}_q^*$.

3.3.2 Implementation of Chebyshev Acceleration

We now describe an efficient technique for computing $\mathbf{y}_q^{(t)} = \sum_{m=0}^{t} \gamma_m \mathbf{x}_q{}^{(m)}$. The definition of $\mathbf{y}_q^{(t)}$ suggests that its computation requires the addition of t vectors in the tth iteration, and that we need to store all $\mathbf{x}_q{}^{(t)}$ vectors computed throughout the power iteration. However, exploiting the

Table 2: Network data sets used in the experiments

Network	`Enron E-mail`	`Brightkite`	`Gowalla`	`Skitter`
Number of Nodes	36,692	58,228	196,591	1,696,415
Number of Edges	183,831	214,078	950,327	11,095,298
Average Node Degree	10.02	7.35	9.67	13.08

observation that Chebyshev polynomials are defined as a recurrence, we can show that $\mathbf{y}_q^{(t+1)}$ can be computed from $\mathbf{y}_q^{(t-1)}$ and $\mathbf{y}_q^{(t)}$.

For this purpose, let $\zeta_t = T_t(\frac{1}{1-\alpha})$. Observing that $\zeta_{t+1} = \frac{2}{1-\alpha}\zeta_t - \zeta_{t-1}$ and by the definition of p_t, it satisfies three-term recurrence. Therefore, we can write

$$\zeta_{t+1}p_{t+1}(\mathbf{W}) = \frac{2}{(1-\alpha)}\zeta_t\mathbf{W}p_t(\mathbf{W}) - \zeta_{t-1}p_{t-1}(\mathbf{W}). \quad (15)$$

Finally, we can use above last equation and basic algebraic manipulations to obtain [16]:

$$\begin{aligned}\mathbf{y}_q^{(t+1)} &= \mathbf{x}_q^* + \mathbf{e}^{(t+1)} \\ &= \mathbf{x}_q^* + p_{t+1}(\mathbf{W})\mathbf{e}^{(0)} \\ &= \mathbf{x}_q^* + \frac{2\zeta_t}{(1-\alpha)\zeta_{t+1}}\mathbf{W}p_t(\mathbf{W})\mathbf{e}^{(0)} \\ &- \frac{\zeta_{t-1}}{\zeta_{t+1}}p_{t-1}(\mathbf{W})\mathbf{e}^{(0)} \\ &= \frac{2\zeta_t}{(1-\alpha)\zeta_{t+1}}\left(\mathbf{W}\mathbf{y}_q^{(t)} + \alpha\mathbf{r}_q\right) - \frac{\zeta_{t-1}}{\zeta_{t+1}}\mathbf{y}_q^{(t-1)}.\end{aligned} \quad (16)$$

In other words, we can compute $\mathbf{y}_q^{(t+1)}$ from $\mathbf{y}_q^{(t)}$ and $\mathbf{y}_q^{(t-1)}$, without requiring storage of the previous iterates.

3.3.3 Processing Top-k Proximity Queries

While Chebyshev acceleration can be used to efficiently compute the proximity of all nodes in $\mathbf{V}$ to a query node q, it often suffices to identify the k nodes that are most proximate to q, where $k \ll |\mathbf{V}|$. As discussed above, existing algorithms for efficiently processing top-k proximity queries use the convergence properties of power iteration and the topology of the network to quickly identify nodes that are not sufficiently proximate to k, so that such nodes can be pruned out [21, 23]. In Squeeze, Zhang et al. [23] use the bound on the norm of the residual for standard power iteration ($(1-\alpha)^t$) to obtain a bound on the proximity score of a node. Subsequently, for "in-bound proximity queries" (i.e., the proximity is quantified in terms of the probability of being at the query node for a random walk that makes frequent restarts at each other node), they show that the proximity score of each node forms a monotonically non-decreasing sequence throughout the power iterations. They utilize this monotonicity, along with the bound on the norm of the residual, to identify the nodes that can be pruned.

When Chebyshev acceleration is used to compute proximity scores, $\mathbf{y}_q^{(t)}(u)$ does not produce a monotonically non-decreasing sequence for all nodes $u \in \mathbf{V}$. However, as we show by the following theorem, this is not required, and that the error bound provided by Chebyshev acceleration can indeed be used to quickly identify nodes that are not sufficiently proximate to q. More importantly, this theorem shows that the idea is not limited to "in-bound proximity queries"; rather, it can be applied to any proximity measure that can be computed via power iterations, for which the error in each iteration can be bounded.

Theorem 2. *Let S_q denote the set of the k nodes in $\mathbf{V}$ that are most proximate to q. Let u_t be the node such that $\mathbf{y}_q^{(t)}(u_t)$ is the kth largest value in $\mathbf{y}_q^{(t)}$. Then, for every node u in $\mathbf{y}_q^{(t)}(u)$, we must have $\mathbf{y}_q^{(t)}(u) \geq \mathbf{y}_q^{(t)}(u_t) - 4\mu^t$.*

Proof. There are two possible cases to consider. In the first case, $\mathbf{y}_q^{(t)}(u)$ is among the top k values in $\mathbf{y}_q^{(t)}$. In this case, it is clear that $\mathbf{y}_q^{(t)}(u) \geq \mathbf{y}_q^{(t)}(u_t) - 4\mu^t$, since $\mathbf{y}_q^{(t)}(u) \geq \mathbf{y}_q^{(t)}(u_t)$ by definition of u_t.

In the second case, $\mathbf{y}_q^{(t)}(u)$ is not among the top k values in $\mathbf{y}_q^{(t)}$. In this case, there must be at least one node $v \in \mathbf{V}$ such that $\mathbf{y}_q^{(t)}(v)$ is among the top k values in $\mathbf{y}_q^{(t)}$, but $v \notin S_q$ (at least one node must drop out of the top-k list to make space for u in the top-k list). Now, using $\|\mathbf{y}_q^{(t)} - \mathbf{x}_q^*\| \leq 2\mu^t$, we obtain the following inequalities:

$$\mathbf{y}_q^{(t)}(v) \leq \mathbf{x}_q^*(v) + 2\mu^t \Rightarrow \mathbf{x}_q^*(v) \geq \mathbf{y}_q^{(t)}(v) - 2\mu^t$$

and

$$\mathbf{y}_q^{(t)}(u) \geq \mathbf{x}_q^*(u) - 2\mu^t \Rightarrow \mathbf{x}_q^*(u) \leq \mathbf{y}_q^{(t)}(u) + 2\mu^t$$

Since $u \in S_k$ but $v \notin S_k$, we have $\mathbf{x}_q^*(u) \geq \mathbf{x}_q^*(v)$, so it follows that

$$\mathbf{y}_q^{(t)}(u) + 2\mu^t \geq \mathbf{x}_q^*(u) \geq \mathbf{x}_q^*(v) \geq \mathbf{y}_q^{(t)}(v) - 2\mu^t.$$

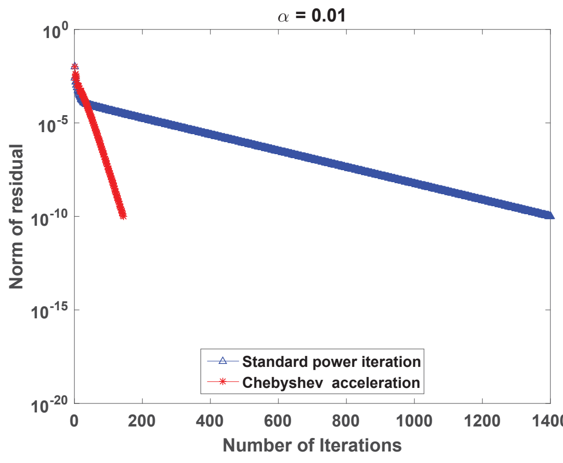
Since v is in the top-k at the tth iteration, we have $\mathbf{y}_q^{(t)}(v) \geq \mathbf{y}_q^{(t)}(u_t)$, so we obtain

$$\mathbf{y}_q^{(t)}(u) + 2\mu^t \geq \mathbf{y}_q^{(t)}(u_t) - 2\mu^t \Rightarrow \mathbf{y}_q^{(t)}(u) \geq \mathbf{y}_q^{(t)}(u_t) - 4\mu^t.$$

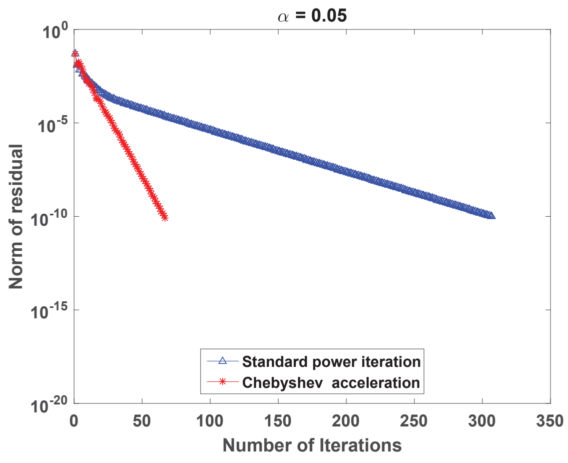
□

Using this result, at any step of the Chebyshev iteration, we can identify nodes that are not sufficiently proximate to q to make it to the top-k list. Specifically, at iteration t, if $\mathbf{y}_q^{(t)}(u) < \mathbf{y}_q^{(t)}(u_t) - 4\mu^t$ for a node u, then u can san safely be pruned out from the list of candidates for the top-k list. The resulting algorithm for computing the top-k most proximate nodes to q is given in Algorithm 1.

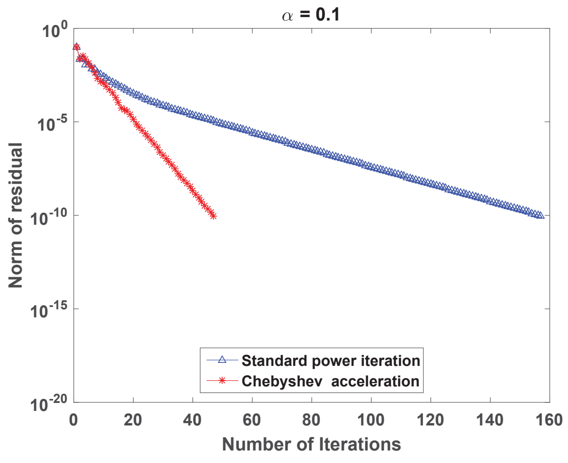
Since $\mu < (1-\alpha)$, we have $4\mu^t \ll (1-\alpha)^t$ for the values of t that are of interest (i.e., the number of iterations is large enough for very large networks). Therefore, Chebyshev acceleration provides a more efficient method for processing top-K queries than algorithms that utilize the convergence characteristics of standard power iteration, e.g., Squeeze [23]. Furthermore, as we demonstrate in the next section via comprehensive experimental results, although Chopper does not directly utilize information on network structures to speed up computation, it also outperforms algorithms that utilize the network structure.



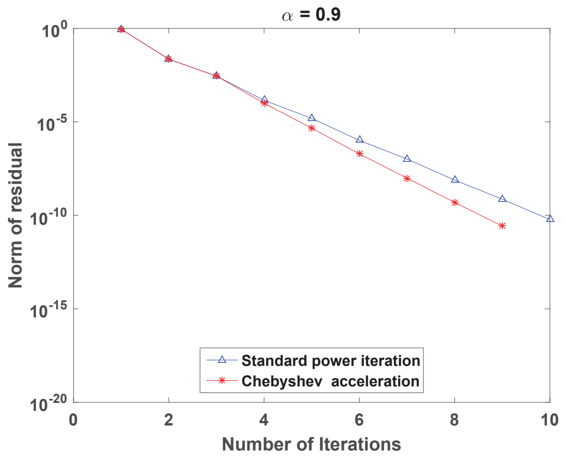
(a) $\alpha = 0.01$



(b) $\alpha = 0.05$



(c) $\alpha = 0.1$



(d) $\alpha = 0.9$

Figure 1: **Comparison of the convergence rates of standard power iteration (RWR) and Chebyshev acceleration on the `Enron Email` dataset.** The plots show the norm of the difference between successive values of the proximity vector as function of the number of iterations, for damping factor $\alpha = 0.01, 0.05, 0.1, 0.9$.

4. EXPERIMENTAL RESULTS

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

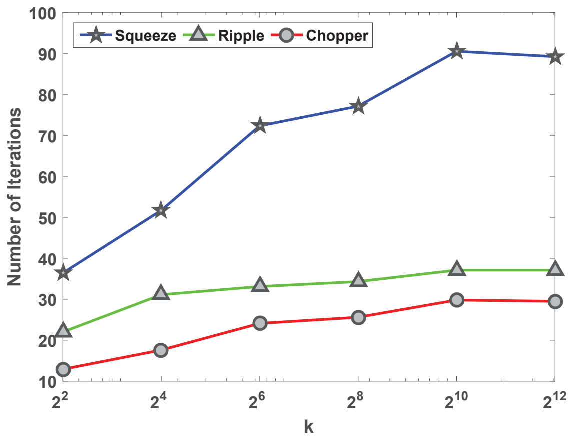
We start our discussion by describing the datasets and the experimental setup. We then assess the performance of Chebyshev acceleration in the computation of random walk based proximity globally, i.e., for all nodes in the network. Subsequently, we compare the performance of CHOPPER in processing top-k proximity queries with two state-of-the-art algorithms, SQUEEZE and RIPPLE [23], using both number of iterations and runtime as performance criteria.

4.1 Datasets and Experimental Setup

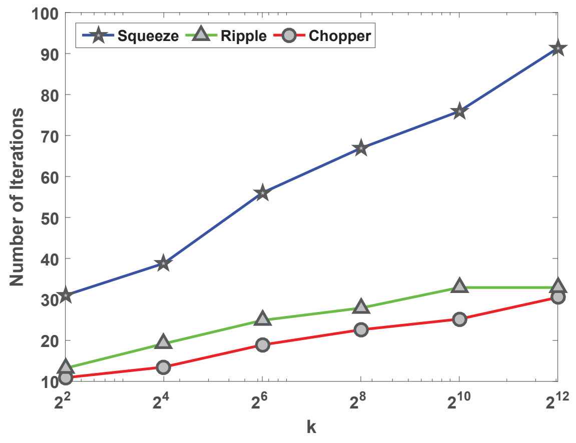
We use four real-world network datasets from the Stanford Network Analysis Project[1] for our experiments. Details of these four networks are given on Table 2. The **`Enron E-mail`** dataset represents the undirected e-mail communication network at Enron. **`Brightkite`** and **`Gowalla`** datasets represent the Brightkite and Gowalla location based online social networks. The **`Skitter`** dataset represents the undirected internet topology graph, constructed from traceroutes run daily in 2005. These datasets are selected as representative samples for network sizes in terms of the number of nodes and of edges. Furthermore, network proximity queries are meaningful on all of these networks.

For SQUEEZE and RIPPLE algorithms, we use the Java implementation provided by Zhang et al [23]. We implement CHOPPER in both Matlab and Java, and the results reported for runtime reflect that of the implementation in Java. We assess the performance of the algorithms for different values

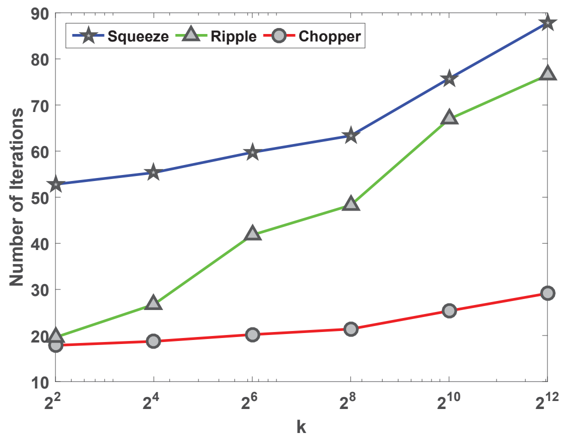
[1]https://snap.stanford.edu/data/



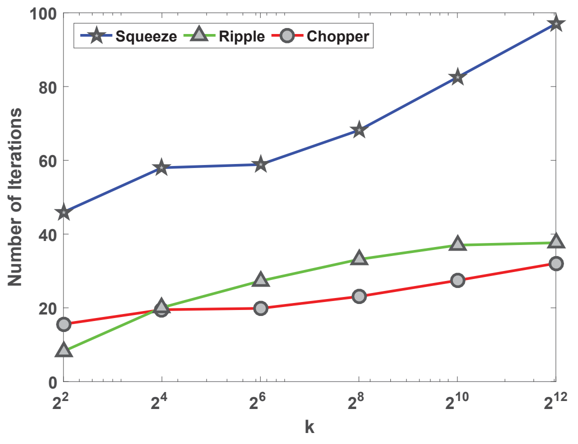
(a) `Enron Email`



(b) `Brightkite`



(c) `Gowalla`



(d) `Skitter`

Figure 2: **The number of iterations required for Chopper, Squeeze, and Ripple in computing the top-k nodes that are most proximate to a query node, as a function of k ranging from 4 to 4096.** In these experiments, damping factor $\alpha = 0.2$ and the reported numbers are the averages across 1000 randomly chosen query nodes.

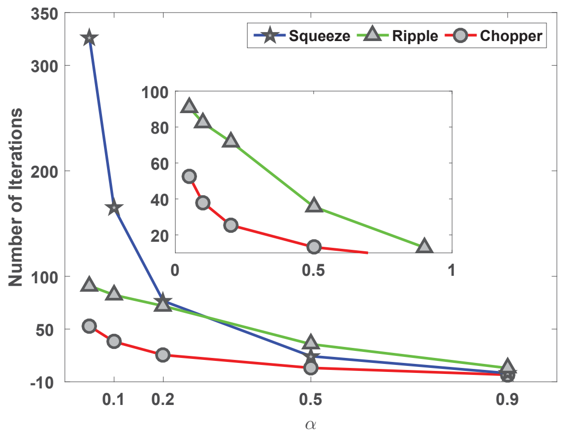
of damping factor (restart probability), as well as the parameter k for top-k proximity queries. For all experiments involving top-k proximity queries, we randomly select 1000 query nodes and report the average of the performance figures for these 1000 queries. In all experiments, $\mathbf{r}_q$ is set to the identity vector for node q. All of the experiments are performed on a Dell PowerEdge T5100 server with two 2.4 GHz Intel Xeon E5530 processors and 32 GB of memory.

4.2 Global Network Proximity Computation

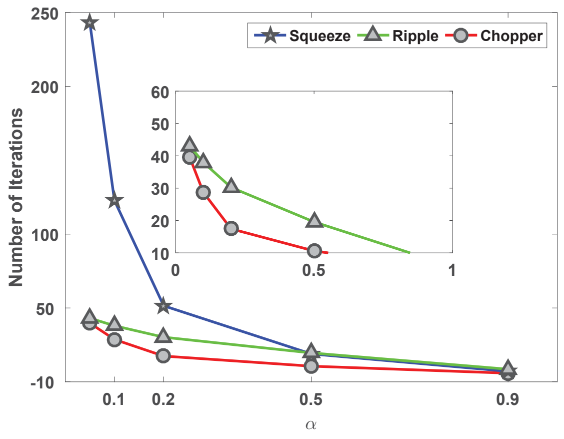
We assess the performance of Chebyshev polynomials in accelerating the computation of network proximity scores for all nodes in the network. The purpose of this analysis is to quantify the improvement using Chebyshev acceleration beyond specific applications such as top-k proximity queries, and to demonstrate its applicability to a broader range of problems that involve iterative computation of network proximity scores.

Figure 1 shows the convergence rate for Chebyshev acceleration in comparison to the standard power iteration for computing random walk based restart based proximity. We limit this analysis to the `Enron Email` dataset, since global proximity computation is expensive (and often unnecessary) for larger networks. In the figure, for the restart probability α ranging from 0.01 to 0.9, we report the norm of the difference between two successive values of the proximity vector at each iteration. In all cases, the threshold for convergence is set to 10^{-10}. As seen in the figure, Chebyshev acceleration significantly reduces the number of iterations required to compute RWR based proximity.

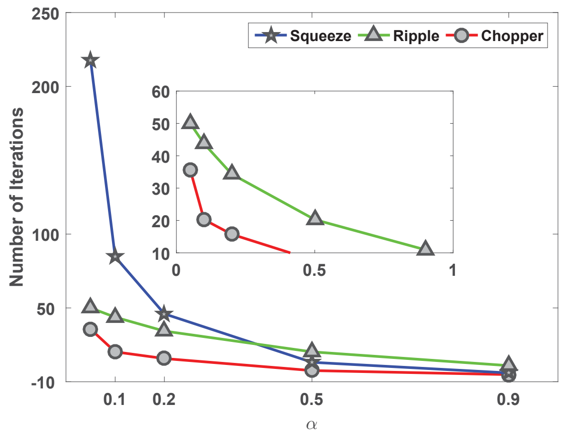
It is important to note that Chebyshev acceleration provides larger performance gains for smaller values of α. This observation is consistent with the theoretical analysis of the convergence of Chebyshev acceleration, reflected in the relationship between the variable μ, which characterizes the convergence of Chebyshev acceleration and the parameter α. Intuitively, for large values of α, the random walk is largely localized – thus acceleration of convergence is of limited benefit. However, queries that involve large α are of limited practical interest, since they do not fully utilize the information provided by the network (e.g., for $\alpha > 0.5$ the random walk is expected to move away from the query node



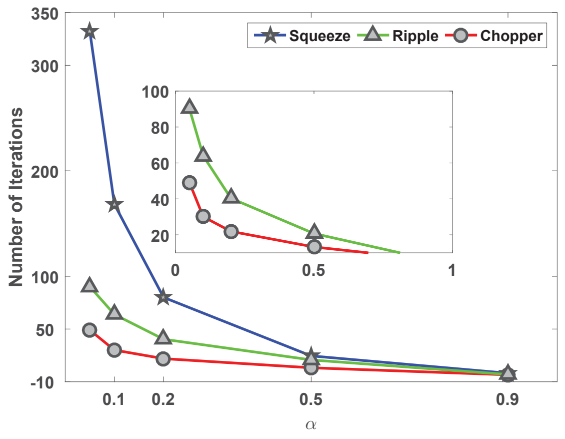
(a) `Enron Email`



(b) `Brightkite`



(c) `Gowalla`



(d) `Skitter`

Figure 3: **The number of iterations required for** Chopper**,** Squeeze**, and** Ripple **in computing the top-$k = 20$ nodes as a function of α ranging from 0.05 to 0.9.** In these experiments, top-$k = 20$ and the reported numbers are the averages across 1000 randomly chosen query nodes for each α .

by two hops on the average). For this reason, larger performance gains for smaller values of α are valuable for practical applications.

4.3 Top-k Proximity Queries

In top-k proximity queries, the two major parameters are the damping factor, α, and the number of nodes to be identified as most proximate to the query node, k. We evaluate the performance of Chopper as a function of both parameters, using four large real-world networks.

The performance of Chopper in comparison to two algorithms, Squeeze and Ripple as a function of k is shown in Figure 2. For this analysis, we fix the damping factor α to 0.2. Recall that Squeeze uses the convergence characteristics of the standard power iteration to terminate the power iterations early. Ripple, on the other hand, also uses network structure to bound the proximity of the nodes in the network to the query node, thereby reducing the number of iterations further. As seen in the figure, Chopper consistently outperforms both algorithms for all four networks. The favorable performance of Chopper as compared to Ripple, is particularly notable, since Ripple also uses information on network structure to speed up computation, whereas Chopper only utilizes the convergence properties of the numerical scheme.

The only case in which Ripple requires fewer iterations than Chopper is for $k = 4$ on the `Skitter` data set. When the network diameter is large and the query results are localized around the seed node, utilization of network structure is particularly beneficial. However, even for such networks, if the query seeks a larger number of proximate nodes (≥ 8), leveraging network structure does not reduce the number of iterations for standard power iterations as much as Chebyshev acceleration does.

We also compare the performance of Chopper with Squeeze and Ripple as a function of the damping factor α. For this purpose, we fix $k = 20$ and process top-k proximity queries for 1000 randomly chosen queries for α ranging from 0.01 to 0.9. The results of this analysis for all four data sets is shown in the Figure 3. In each panel, close-ups of the curves for Chopper and Ripple are also shown for smaller values of α to facilitate better comparison. As seen in the figure, the number of iterations required for processing top-k queries is similar for all algorithms on all datasets for $\alpha \geq 0.5$. This

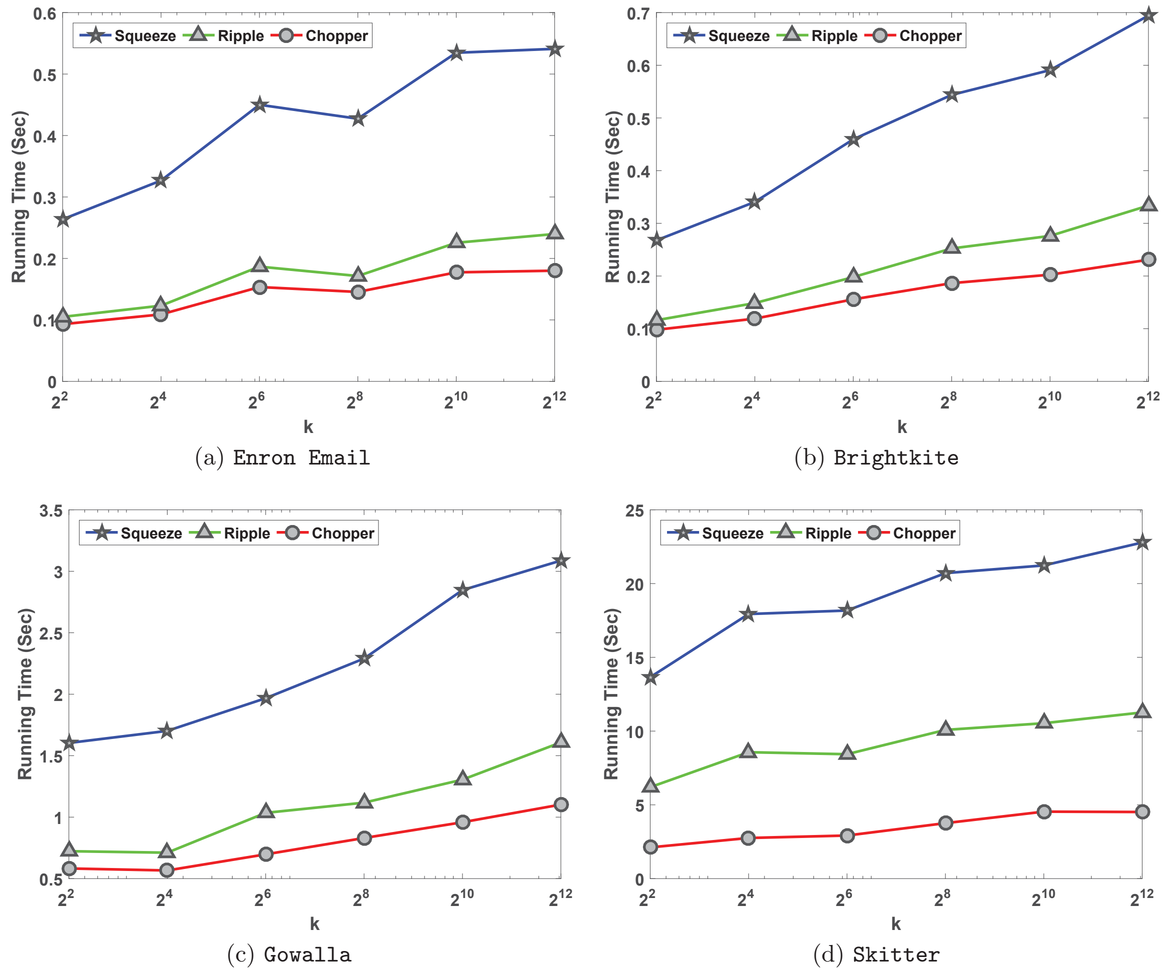


Figure 4: **The runtime in seconds required by** Chopper**,** Squeeze**, and** Ripple **to process queries for top k nodes that are most proximate to a query node, as a function of k ranging from 4 to 4096.** In these experiments, the damping factor α is set to 0.2 and the reported numbers are the averages across 1000 randomly chosen query nodes.

is expected since the search is "easier", i.e., it is limited to nodes that are very close to the query node in this case. However, for smaller (and more relevant) values of α, Chopper consistently achieves best performance.

4.4 Runtime Performance

Finally, we compare the performance of Chopper with Squeeze and Ripple in terms of running time as a function of k. For this purpose, we fix $\alpha = 0.2$ and process top-k proximity queries for 1000 randomly chosen queries for k ranging from 4 to 4096. The results of this analysis for all four data sets is shown in the Figure 4. As expected, the time required to process top-k queries increases with k for all methods. However, across all datasets and for all values of α, Chopper consistently delivers best performance, with two-fold improvement over the runtime of Ripple for the largest dataset `Skitter`.

5. CONCLUSION

In this paper, we propose an alternate approach to accelerating network proximity queries. The proposed approach is based on Chebyshev polynomials, an established acceleration technique for iterative methods in numerical linear algebra. We show that our approach, Chopper, produces asymptotically faster convergence in theory, and significantly decreased convergence times 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 Chopper outperforms existing methods significantly for wide ranges of parameter values. When integrated with existing methods, Chopper yields further improvement in performance over state of the art techniques.

Future efforts in this direction would include incorporation of other acceleration techniques into our framework, extensions to other iterative proximity measures, and their applications. Furthermore, while Chopper 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. Comparison of Chopper against such approximate algorithms can provide further insights into the trade-off between runtime and accuracy in the context of network proximity problems.

Acknowledgements

We would like to thank Xiang Zhang, Yubao Wu, and Sean Maxwell for many useful discussions, and anonymous reviewers for their valuable comments. This work was supported in part by US National Science Foundation (NSF) grants CCF-1533795, CCF-0953195, and CNS 1422338, and US National Institutes of Health (NIH) grant U01-CA198941.

6. REFERENCES

[1] D. Aldous and J. A. Fill. Reversible markov chains and random walks on graphs, 2002. Unfinished monograph, recompiled 2014, available at http://www.stat.berkeley.edu/~aldous/RWG/book.html.

[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] 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.

[5] 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. *P*loS one, 8(10):e76339, 2013.

[6] M. Coşkun and M. Koyutürk. Link prediction in large networks by comparing the global view of nodes in the network. In *2*015 IEEE International Conference on Data Mining Workshop (ICDMW), pages 485–492. IEEE, 2015.

[7] S. Erten, G. Bebek, R. M. Ewing, and M. Koyutürk. Dada: Degree-aware algorithms for network-based disease gene prioritization. *B*ioData mining, 4(1):1–20, 2011.

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

[9] M. Hofree, J. P. Shen, H. Carter, A. Gross, and T. Ideker. Network-based stratification of tumor mutations. *N*ature Methods, 10(11):1108–1115, Sept. 2013.

[10] H. Jeong and B. Yoon. Accurate multiple network alignment through context-sensitive random walk. *BMC* Systems Biology, 9(S-1):S7, 2015.

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

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

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

[14] 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.

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

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

[17] P. Sarkar and A. W. Moore. Fast nearest-neighbor search in disk-resident graphs. In *P*roceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 513–522. ACM, 2010.

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

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

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

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

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

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

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

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