Practical Acceleration for Computing the HITS ExpertRank Vectors

Yunkai Zhou*

Dedicated to Prof. Danny C. Sorensen on the occasion of his 65th birthday in 2011

Abstract

A meaningful rank as well as efficient methods for computing such a rank are necessary in many areas of applications. Major methodologies for ranking often exploit principal eigenvectors. Kleinberg's HITS model is one of such methodologies. The standard approach for computing the HITS rank is the power method. Unlike the PageRank calculations where many acceleration schemes have been proposed, relatively few works on accelerating HITS rank calculation exist. This is mainly because the power method often works quite well in the HITS setting. However, there are cases where power method is ineffective, moreover, a systematic acceleration over the power method is desirable even when power method works well. We propose a practical acceleration scheme for HITS rank calculations based on filtered power method by adaptive Chebyshev polynomials. For cases where the gap-ratio is below 0.85 for which power method works well, our scheme is about twice faster than the power method. For cases where gap-ratio is unfavorable for the power method, our scheme can provide significant speedup. When the ranking problems are of very large scale, even a single matrix-vector product can be expensive, for which accelerations are highly necessary. The scheme we propose is desirable in that it provides consistent reduction in number of matrix-vector products as well as CPU time over the power method, with little memory overhead.

Key words: HITS, ranking, principal eigenvector, Chebyshev filter, symmetric semidefinite matrix, filter bound, Lanczos

1 Introduction

Using principal eigenvector of non-negative matrices for ranking purpose has a history of over half a century ([31], see [35] for a recent survey). The methodology is best represented by two significant and extremely successful modern applications: the Google PageRank [26] and the Hyperlink-Induced Topic Search (HITS) [19].

HITS, developed by Kleinberg in the 90's, is used in the `ask.com` search engine. The resulting two ranking vectors from HITS provide the so-called ExpertRanks. HITS method has broad applications in areas where a certain ranking is sought, e.g., similarity ranking, academic citation ranking, product quality ranking [21, 22, 10, 4, 9]. We refer to [22] for discussions on the strengths

*Department of Mathematics, Southern Methodist University, Dallas, TX 75275 (`yzhou@smu.edu`). Supported in part by the National Science Foundation under Grant No. CMMI-0727194 and OCI-0749074.

and weaknesses of HITS, together with literature on modifications to overcome the weaknesses. The algorithm we discuss in this paper is applicable to the original HITS model as well as its modifications.

The eigenproblems related to web search and data mining can be of enormous dimension. Because of memory constraint, the dominant method for solving the PageRank and the HITS eigenproblems has been the power method. Krylov subspace methods, which converge faster than the power method, can quickly become impractical if the subspace dimension becomes relatively large.

It is well-known that power method can be inefficient if the gap-ratio (the second largest eigenvalue over the largest eigenvalue, in magnitude) is close to 1. For the standard PageRank model, the gap-ratio is ingeniously engineered to be 0.85, which results in relatively fast convergence of the power method. Many acceleration methods have been proposed for PageRank calculations, most of them are geared to the case where the gap-ratio is close to 1. These include extrapolation [17, 18, 7], linear system approach [13, 20], Krylov subspace methods [15, 36], and relaxation type methods [14].

In contrast, relatively few works have appeared for accelerating HITS ranking vectors computations. One reason is that the matrices for search-dependent HITS is usually not of very large dimension [22]; the second reason is that the gap-ratio of HITS matrices is often not close to 1, which makes the power method work reasonably well in practice.

Here we focus on accelerating the computation of the HITS ranking vectors. We propose a practical acceleration scheme based on *filtered power* method, which means it can be used for acceleration wherever power method is applicable: either for the search-dependent HITS, or the search-independent HITS for which the matrices are usually of huge dimension and acceleration is very desirable. Moreover, our acceleration scheme can significantly speed up calculation when the gap-ratio is not favorable for the power method; even in the cases where the gap-ratio is small, the acceleration scheme can still provide meaningful speedup.

We mention that the proposed scheme does not apply to accelerating PageRank calculations, this is because the PageRank matrix is non-hermitian, which can have complex eigenvalues, while the proposed scheme requires that all eigenvalues are real.

However, we point out that, besides the significance of HITS, the importance of accelerating the solution of the eigenproblems related to matrices of form LL^{T} or $L^{\mathrm{T}}L$ can go far beyond the HITS ranking. One main reason is that matrices of form LL^{T} or $L^{\mathrm{T}}L$ in the HITS eigenproblems are closely related to the covariance or Gram matrices, which are central to modern large scale statistical computations and data mining (e.g., [3, 16]). The acceleration scheme via filtering described in this paper can be extended for computing more principal eigenvectors, such as in truncated SVD calculations (essentially, computing principal eigenvectors of hermitian matrices of form LL^{T} or $L^{\mathrm{T}}L$). The extension has broad applications in recent data mining and machine learning methods which utilize partial SVD (e.g. [2, 24, 8]).

The organization of the paper is as follows: Section 2 briefly reviews the HITS model; Section 3 describes the main acceleration algorithms; Section 4 presents formulas that lead to the main algorithms together with related analysis of the algorithms; and Section 5 presents numerical results.

2 The HITS model

The HITS model expresses the hyperlink structure of the web using directed graphs. It divides web pages into two categories: Hubs and Authorities. The premise of HITS is that appropriate Hub and Authority ranking can be obtained by mutual enforcement between the two ranks, since in general good Hubs tend to point to good Authorities, and good Authorities tend to be pointed to by good Hubs [9, 22]. The ranking of Hubs and Authorities, when combined, provides a balanced quality rank. Therefore the HITS model computes two ranking vectors, one for the Hubs and one for the Authorities. Pages with high Authority ranks are expected to have relevant content, while pages with high Hub ranks indicate that they contain hyperlinks to relevant content.

Let $\mathbf{v}_h$ and $\mathbf{v}_a$ denote respectively the Hub rank vector and the Authority rank vector. Let the adjacency matrix of the directed graph be L, with dimension $n \times n$. We can express the HITS method as an iterative procedure for updating $\mathbf{v}_h$ and $\mathbf{v}_a$ starting from given initials $\mathbf{v}_a^{(0)}$ and $\mathbf{v}_h^{(0)}$,

$$\mathbf{v}_a^{(k)} = L^{\mathrm{T}}\mathbf{v}_h^{(k-1)}, \quad \mathbf{v}_h^{(k)} = L\mathbf{v}_a^{(k)}, \quad \text{for } k = 1, 2, 3, ... \tag{1}$$

Combining the two equations in (1) gives

$$\mathbf{v}_a^{(k)} = L^{\mathrm{T}}L\mathbf{v}_a^{(k-1)}, \quad \mathbf{v}_h^{(k)} = LL^{\mathrm{T}}\mathbf{v}_h^{(k)}. \tag{2}$$

Equations in (2) represent the power method without normalization applied to $L^{\mathrm{T}}L$ and LL^{T}. Therefore, with normalization, the converged $\mathbf{v}_a$ and $\mathbf{v}_h$ will be respectively the unit principal eigenvector of $L^{\mathrm{T}}L$ and LL^{T}. That is,

$$LL^{\mathrm{T}}\mathbf{v}_h = \lambda_{\max}\mathbf{v}_h, \tag{3}$$

$$L^{\mathrm{T}}L\ \mathbf{v}_a = \lambda_{\max}\mathbf{v}_a, \tag{4}$$

where $\lambda_{\max}$ is the largest eigenvalue of both $L^{\mathrm{T}}L$ and LL^{T}.

We assume that $\lambda_{\max}$ is unique, which guarantees that power method will converge. If $\lambda_{\max}$ is not unique, one can employ a *primitive trick* ([22, p. 120], [11]) which applies power method to the following modified HITS matrices for which $\lambda_{\max}$ is unique,

$$\xi L^{\mathrm{T}}L + \frac{1-\xi}{n}\mathbf{e}\mathbf{e}^{\mathrm{T}}, \quad \xi LL^{\mathrm{T}} + \frac{1-\xi}{n}\mathbf{e}\mathbf{e}^{\mathrm{T}}, \quad \text{where } e = \underbrace{[1, 1, ..., 1]}_{n}^{\mathrm{T}}. \tag{5}$$

The matrices in (5) are also used for the query-independent HITS model [22, p. 124]. In the query-independent scenario, the dimension n can be extremely large.

The gap-ratio of $L^{\mathrm{T}}L$ or LL^{T} in the HITS model is observed to be reasonably small, which means that power method may converge reasonably fast. However, there is no theoretical result that guarantees the gap-ratio to be small, and clearly the gap-ratio can be problem dependent. In fact we encounter several realistic models for which the gap-ratio is unfavorable for the power method. Therefore, it is desirable to be able to accelerate over the power method consistently, regardless of the gap-ratio.

3 The main algorithm

We mainly discuss computing the Hub rank vector $\mathbf{v}_h$ of (3). The Authority rank vector $\mathbf{v}_a$ can be readily obtained by a matrix vector product $L^{\mathrm{T}}\mathbf{v}_h$ once $\mathbf{v}_h$ converges. The same procedure can be applied to converge $\mathbf{v}_a$ first, then obtain $\mathbf{v}_h$ as $L\mathbf{v}_a$.

For the HITS eigenproblem (3), the $L^{\mathrm{T}}L$ and LL^{T} are symmetric positive semidefinite, their spectra are on the real line. This property suits ideally for using Chebyshev filters to accelerate the computation of the ExpertRank vectors.

The proposed algorithm is based on Chebyshev filter accelerated power method. Our acceleration scheme is practical in that it uses only one more vector than the power method, because of the 3-term recurrences needed for Chebyshev polynomials. Therefore the memory usage is moderate comparing with the power method, and more economical comparing with Krylov subspace methods that use a larger subspace dimension. Computational-wise, our scheme needs only one additional inner product to compute a Rayleigh quotient per m matrix-vector multiplications, where m is the polynomial degree; therefore the cost per iteration is similar to that of the power method.

To apply Chebyshev polynomials for targeted filtering, we need to determine what filtering bounds to use. Our numerical experiences show that applying Chebyshev filters in the standard way with fixed filtering bounds does not work well. However, with dynamic, adaptively adjusted filtering bounds as proposed in [41, 39], we can construct highly effective Chebyshev filters. This is shown in our density functional theory (DFT) calculations [41, 40, 34], where adaptively constructed Chebyshev filters can routinely obtain an order of magnitude speedup over eigenvector based approaches. But unlike in the DFT calculations, here the wanted spectrum is located at the higher end instead of the lower end. This causes no problem and will be addressed in the next subsection.

3.1 Estimating the bounds for filtering

It is well-known that Chebyshev polynomials are bounded by 1 on the interval $[-1, 1]$ and increase exponentially outside $[-1, 1]$. To employ the polynomials for accelerating eigenvalue calculations, we mainly need to linearly map the unwanted spectrum into the $[-1, 1]$ interval. The wanted spectrum will be automatically mapped outside $[-1, 1]$, thus magnified by the Chebyshev polynomial filters.

For HITS ExpertRank we want to compute the eigenvector associated with the largest eigenvalue. Therefore the filters need to dampen the lower end of the spectrum. Since $L^{\mathrm{T}}L$ and LL^{T} are positive semidefinite, their eigenvalues are nonnegative, hence the filtering *lower* bound of the spectrum can be conveniently set to 0.

The upper bound of the total spectrum can be obtained by the estimator for hermitian matrices [38].

However, in the HITS setting, it is not this total upper bound that determines the interval to be dampened. Instead, it is what we call the *filtering upper bound*, i.e., the upper bound of the unwanted spectrum, that determines filter efficiency. This filtering upper bound must be smaller than the largest eigenvalue of LL^{T}, denoted as $\lambda_{\max}(LL^{\mathrm{T}})$, so that $\lambda_{\max}(LL^{\mathrm{T}})$ will be mapped outside $[-1, 1]$ interval, by this it will be properly magnified instead of dampened by

the filter constructed from the bound.

The main idea of the adaptive filtering is on adjusting the filtering upper bound. For a procedure that can adaptively adjust this bound, we exploit the variational property related to the Courant-Fisher minmax theorem [33, 27]. This is done in two stages.

At the first stage, we modify the bound estimator in [38] to output two bounds u_l and u_L, with $0 < u_l < u_L$. The u_l is an estimate of the filtering upper bound, which is essential for the filtering and needs to satisfy $u_l < \lambda_{\max}(LL^{\mathrm{T}})$. The u_L is less essential and only needs to be a rough estimate of $\lambda_{\max}(LL^{\mathrm{T}})$, it can be less than $\lambda_{\max}(LL^{\mathrm{T}})$, which is to be contrasted with [38] where the upper bound estimate must bound the full spectrum from above. This is because the wanted eigenvalues in [38] are located at the lower end of the spectrum, the goal of filtering is to dampen all the higher end of the spectrum. While here the wanted eigenvalue is the largest one, so we need to dampen the full lower end of the spectrum. In other words, the upper bound sought in [38] serves the same purpose as the total lower bound 0 here. By utilizing the positive semidefiniteness of LL^{T}, we obtained this filtering lower bound 0 for free. The essential bound to estimate is the filtering upper bound u_l.

Pseudocode in Matlab style of the modified estimator are listed in Algorithm 3.1. This estimator uses a k-step Lanczos procedure to compute a standard Lanczos decomposition [27, 30]

$$LL^{\mathrm{T}}V_k = V_k T_k + f_k e_k^{\mathrm{T}} \ , \ \text{with } V_k^{\mathrm{T}} V_k = I_k \text{ and } V_k^{\mathrm{T}} f_k = \mathbf{0},$$

where V_k contains the k orthonormal Lanczos vectors. Exterior eigenvalues of LL^{T} may be roughly estimated by the exterior eigenvalues of the tridiagonal Rayleigh quotient matrix T_k.

Since here only a rough upper bound of $\lambda_{\max}(LL^{\mathrm{T}})$ is needed, the Lanczos step k can be set very small. In practice we set $k = 3$ so as to reduce memory requirement. Moreover, since $u_L \geq \lambda_{\max}(LL^{\mathrm{T}})$ is not necessary here, we simply use the largest Ritz value (plus an error term which is standard in Lanczos error estimate) to estimate u_L, as done at the last step in Algorithm 3.1. The post-processing step after the Lanczos decomposition as used in [38] to safeguard an upper bound of the total spectrum is not needed here. The initial vector v_0 input to Algorithm 3.1 can be a random vector, a vector with all one's, or any user provided nonzero vector.

Algorithm 3.1. $[u_l, u_L, v_0] = \texttt{Lanczos\_bounds}(k, v_0)$

1. *Set* $V(:,1) = v_0/\|v_0\|_2$*;*
2. $V(:,2) = L(L^T V(:,1))$*;* $\alpha = V(:,2)^T V(:,1)$*;* $V(:,2) \leftarrow V(:,2) - \alpha V(:,1)$*;* $T(1,1) = \alpha$*;*
3. *Do* $j = 2$ *to* $\min(k,5)$
4. $\quad \eta = \|V(:,j)\|_2$*;* $\quad V(:,j) \leftarrow V(:,j)/\eta$*;*
5. $\quad V(:,j+1) = L(L^T V(:,j))$*;* $\quad V(:,j+1) \leftarrow V(:,j+1) - \eta V(:,j-1)$*;*
6. $\quad \alpha = V(:,j+1)^T V(:,j)$*;* $\quad V(:,j+1) \leftarrow V(:,j+1) - \alpha V(:,j)$*;*
7. $\quad T(j,j-1) = \eta$*;* $\quad T(j-1,j) = \eta$*;* $\quad T(j,j) = \alpha$*;*
8. *End Do*
9. *Compute* $[Q, D] = eig(T(1:k, 1:k))$*;* *Find* $[u_L, idx] = \max_i \lambda_i(T)$*.*
10. *Return* $u_l = (\min(diag(D)) + u_L)/2$*,* $u_L = u_L + \eta Q(end, idx)$*, and the Ritz vector* $V(:,1:k)Q(:,idx)$ *corresponding to the largest Ritz value as* v_0*.*

To obtain the filtering upper bound u_l and make it satisfy the necessary condition $u_l < \lambda_{\max}(LL^{\mathrm{T}})$, we utilize the two extreme Ritz values from the Lanczos procedure: At the last step of Algorithm 3.1, we set $u_l = (\lambda_{\min}(T_k) + \lambda_{\max}(T_k))/2$. The u_l obtained this way satisfies $u_l < \lambda_{\max}(T_k) \leq \lambda_{\max}(LL^{\mathrm{T}})$, according to the property of a Rayleigh quotient.

In addition to returning a filtering upper bound u_l, Algorithm 3.1 serves another purpose: Starting from v_0, it provides a better initial vector than $(LL^{\mathrm{T}})^k v_0$ to start the filtered power method. This is because the Ritz vector corresponding to the largest Ritz value returned from k-step Lanczos (Algorithm 3.1) is generally better than the vector obtained from applying k-step power method to the same initial vector, thanks to the subspace acceleration provided by the Lanczos iteration over the power method. Therefore it is better to use the Ritz vector returned from Algorithm 3.1 as the initial vector for the Chebyshev filtered power iteration.

Algorithm 3.1 is called only once to provide the initial bound u_l and an initial vector for the first Chebyshev filtering. The other bound u_L is used only for scaling purpose. There is absolutely no need to run more Lanczos steps in Algorithm 3.1 in order to get a relatively good estimate of the largest eigenvalue. In fact, we do not use the u_L in the simplified filter (Algorithm 3.2).

At the second stage, which contains the main steps of the Chebyshev filtered power iteration, we only need to adjust the filtering upper bound u_l at each iteration (since the lower bound is perfectly 0 and need not be changed). A simple yet practically effective adjustment is to use a convex combination of the previous filtering upper bound u_l and an easily computed Rayleigh quotient (denoted as u_u) returned from the Chebyshev filtering (Algorithm 3.2 or 3.3),

$$u_l \leftarrow \beta u_l + (1-\beta)u_u, \quad \text{where} \quad \beta \in (0,1). \tag{6}$$

By the Courant-Fisher minmax theorem [33, 27], the Rayleigh quotient u_u (returned from Algorithm 3.3 or Algorithm 3.2) satisfies $u_u < \lambda_{\max}(LL^{\mathrm{T}})$ before convergence. Together with the fact that $u_l < \lambda_{\max}(LL^{\mathrm{T}})$ at each iteration, we see that the u_l updated by (6) will always be smaller than the true largest eigenvalue of LL^{T}. By linearly mapping $[0, u_l]$ to $[-1, 1]$ to construct a Chebyshev filter, we can guarantee that the constructed filter will dampen the unwanted spectrum of LL^{T} enclosed by $[0, u_l]$. The same filter will automatically magnify the region surrounding the wanted eigenvalue $\lambda_{\max}(LL^{\mathrm{T}})$, resulting in much improved gap-ratio for the filtered matrix, which leads to consistent faster convergence than the power method.

3.2 The Chebyshev accelerated HITS algorithm

The well-known degree-k Chebyshev polynomial of the first kind is defined as (see e.g. [1, p.180])

$$C_k(t) = \begin{cases} \cos(k\ \cos^{-1}(t)), & |t| \leq 1, \\ \cosh(k\ \cosh^{-1}(t)), & t > 1, \\ (-1)^k \cosh(k\ \cosh^{-1}(-t)), & t < -1. \end{cases} \tag{7}$$

The intrinsic 3-term recurrence related to the orthogonal polynomials (7) is

$$C_{k+1}(t) = 2t\ C_k(t) - C_{k-1}(t), \qquad t \in \mathbb{R},\ k = 1, 2, ... \tag{8}$$

Starting from $C_0(t) = 1$ and $C_1(t) = t$, one can apply (8) to conveniently compute higher order $C_k(t)$.

An important property of (7) is the exponential growth of the polynomial outside the $[-1, 1]$ interval [27, p.371]. In fact, under comparable conditions, $|C_k(x)|$ as defined in (7) grows fastest outside $[-1, 1]$ among all polynomials with degree $\leq k$ ([28, p.31]).

The main goal of our acceleration scheme is to utilize the exponential growth property outside the $[-1, 1]$ interval, so that gap-ratio of the filtered matrix will be optimal for convergence. However, the design of the scheme does not focus on what is outside the $[-1, 1]$ interval. Instead we mainly focus on linearly mapping unwanted spectrum to the $[-1, 1]$ interval for damping. The exponential growth property will be automatically applied to the wanted spectrum that is now mapped to an interval outside $[-1, 1]$.

For the HITS rank calculation, we want to dampen the unwanted eigenvalues located at the lower end $[0, u_l]$ of the spectrum. For this purpose we only need to map $[0, u_l]$ into $[-1, 1]$ by $\mathcal{L}(t) = (t - \frac{u_l}{2}) * \frac{2}{u_l}$. This affine mapping applied to any matrix A will transform the eigenvalues of A located in $[0, u_l]$ into the eigenvalues of $\mathcal{L}(A)$ located in $[-1, 1]$. The associated matrix-vector product with a vector v is implemented as

$$\mathcal{L}(A)v = (Av - \frac{u_l}{2}v)(2/u_l) \ .$$

The Chebyshev filter constructed to accelerate computing the HITS Hub rank vector $\mathbf{v}_h$ is listed in Algorithm 3.2. This algorithm uses a simplified Chebyshev filter without scaling, as those used in [41, 37].

Only a single bound u_l is needed to construct a filter via Algorithm 3.2. For constructing the first filter, the u_l can be obtained from Algorithm 3.1. For the remaining filtering steps, each new u_l can be readily adjusted by a convex combination (formula (6)) of the previous u_l and the Rayleigh quotient u_u computed from previous call to Algorithm 3.2.

As pointed out at the end of Section 3.1, the u_l updated by (6) is always smaller than $\lambda_{\max}(LL^{\mathrm{T}})$. This guarantees that the filter constructed will dampen $[0, u_l]$ but magnify the region near $\lambda_{\max}(LL^{\mathrm{T}})$, which results in better gap-ratio than that of the power method.

Algorithm 3.2. $[Y,\ u_u] =$ `Chebyshev_filter`$(X,\ m,\ u_l)$.

1. $e = u_l/2$;
2. $Y = (L * (L^T * X))/e - X$;
3. *Do* $i = 2$ *to* $m - 1$
4. $\quad Y_t = (L * (L^T * Y) - e * Y) * (2/e) - X$;
5. $\quad X = Y$; $Y = Y_t$;
6. *End Do*
7. $Y_t = L * (L^T * Y)$;
8. $u_u = (Y_t^T * Y)/(Y^T * Y)$;
9. $Y = (Y_t - e * Y) * (2/e) - X$;

Given an input vector X, Algorithm 3.2 essentially computes the filtered vector $Y = C_m(\mathcal{L}(LL^{\mathrm{T}}))X$ using a degree-m Chebyshev polynomial. The 3-term recurrence (8) is used

to update $C_m(\mathcal{L}(LL^{\mathrm{T}}))X$. The output u_u is the current Rayleigh quotient, which is used to adjust the filtering upper bound u_l for the next step filtering.

A scaled Chebyshev polynomial filter based on [29, 39] can be developed for the HITS rank calculation. The formula derivation is discussed in the next section, we list the pseudocode in Algorithm 3.3. The scaled version may be useful in term of preventing overflow, i.e., when the degree m is large and no other scaling is applied.

Algorithm 3.3. $[Y,\ u_u] =$ `Chebyshev_filter_scaled`$(X,\ m,\ u_l,\ u_L)$.

1. $e = u_l/2; \quad \sigma = e/(u_L - e); \quad \tau = 2/\sigma;$
2. $Y = (L * (L^T * X) - e * X) * (\sigma/e);$
3. *Do* $\ i = 2 \ \ to \ \ m - 1$
4. $\quad \sigma_{new} = 1/(\tau - \sigma);$
5. $\quad Y_t = (L * (L^T * Y) - e * Y) * (2 * \sigma_{new}/e) - (\sigma * \sigma_{new}) * X;$
6. $\quad X = Y; \quad Y = Y_t; \quad \sigma = \sigma_{new};$
7. *End Do*
8. $\sigma_{new} = 1/(\tau - \sigma);$
9. $Y_t = L * (L^T * Y);$
10. $u_u = (Y_t^T * Y)/(Y^T * Y);$
11. $Y = (Y_t - e * Y) * (2 * \sigma_{new}/e) - (\sigma * \sigma_{new}) * X;$

The purpose of Algorithm 3.3 is again to filter the input vector X by a degree-m Chebyshev polynomial that dampens on the interval $[0, u_l]$. But now we apply $C_m(\frac{1}{e}(u_L - e))$ as a scaling factor. The input bounds need to satisfy $0 < u_l < u_L$, with $u_l < \lambda_{\max}(LL^{\mathrm{T}})$. Here the input u_L only needs to be a crude estimate of $\lambda_{\max}(LL^{\mathrm{T}})$.

The $e = u_l/2$ at the first step of Algorithm 3.3 is the only variable needed for the affine transform which maps $[0, u_l]$ onto the $[-1, 1]$ interval, the other two variables σ and τ are used to realize the update of the scaled Chebyshev polynomial.

Essentially, Algorithm 3.3 computes

$$Y = \frac{C_m(\frac{1}{e}(LL^{\mathrm{T}} - eI))}{C_m(\frac{1}{e}(u_L - e))} X \ . \tag{9}$$

The degree-m matrix-vector product is again obtained via a 3-term recurrence, in which the matrix needs to be the scaled matrix as in (9).

The scaled filter not only dampens on the $[0, u_l]$, it also simultaneously magnifies eigenvalues located on $[u_l, \lambda_{\max}(LL^{\mathrm{T}})]$, with those close to $\lambda_{\max}(LL^{\mathrm{T}})$ being magnified most significantly. This is due to the continuity of the Chebyshev polynomial and its fastest growth property among the degree-m polynomials outside the $[-1, 1]$ interval. The filter improves the gap-ratio of the filtered matrix, which is the essential reason for faster convergence.

Our main algorithm uses Algorithm 3.2, which is the simplified version of Algorithm 3.3, as the default filter. The advantage of the simplified version is that there is no need to estimate the

largest eigenvalue of LL^{T}, only the filtering upper bound u_l needs to be estimated. Moreover, the computation costs slightly less than with scaling. The rationality of the simplified Algorithm is discussed in Section 4.

The Chebyshev accelerated algorithm for computing HITS ExpertRank Hub vector is described in Algorithm 3.4. It mainly relies on matrix-vector products $L(L^{\mathrm{T}}v)$, which can be called from a user provided subroutine.

The structure of Algorithm 3.4 contains an inner-iteration and an outer-iteration. This structure is common in many algorithms that rely on matrix-vector products for solving large scale problems. In web search algorithms, recent papers that have this inner-outer iteration structure include [15, 14].

A ranking vector is a principal eigenvector of the nonnegative matrix LL^{T}. Scaling of this eigenvector by a constant does not change the rank. Usually the normalized principal eigenvector with all *nonnegative* components is used for ranking. (Here, the adjacency matrix is likely non-irreducible because the web graph is very likely not strongly connected. Therefore the Perron-Frobenius theorem [25, p.673], which under irreducibility condition guarantees the existence of an eigenvector with all *positive* components, may not apply.) We start Algorithm 3.4 with a positive vector X_0. However, the polynomial filtered matrix-vector product may flip the direction of a vector, which results in negative components. We address this possibility at step 5 of Algorithm 3.4 by picking the vector with nonnegative components. The converged vector will have nonnegative components and can be used for ranking purpose.

Algorithm 3.4. $[X] =$ `HITS_chebyshev`$(X_0,\ m,\ \beta,\ \tau,$ `itmax`, `method`$)$.

1. *Call* $[u_l, u_L, X_0] =$ `Lanczos_bounds`(k, X_0)*; set* $u_u = u_L$*;*
2. *Normalize the initial:* $X_0 = X_0/\|X_0\|_1$.
3. *Do* $i = 1$ *to* `itmax`
4. *If (* `method == 'scaled'` *)*
   Set $u_L = \max(u_u, u_L)$*;*
   Call $[X,\ u_u] =$ `Chebyshev_filter_scaled`$(X_0,\ m,\ u_l,\ u_L)$.
   Else `%` *default to non-scaled filtering*
   Call $[X,\ u_u] =$ `Chebyshev_filter`$(X_0,\ m,\ u_l)$.
   End If
5. $X = \pm X/\|X\|_1$ (*choose one from* $\pm$ *such that* X *has no negative components*).
6. *If* $\|X - X_0\|_1 < \tau$*, Stop.*
7. *Set* $X_0 = X$.
8. *Set* $u_l = \beta u_l + (1-\beta)u_u$.
9. *End Do*

Algorithm 3.4 is only slightly more complicated than the power method. Essentially, for each m matrix-vector products, the $(LL^{\mathrm{T}})^m X_0$ in power method is replace by (scaled) $C_m(\mathcal{L}(LL^{\mathrm{T}}))X_0$. Memory-wise, since we use $k = 3$ for the Lanczos iteration, and use only 3 vectors for the Chebyshev filtering, the memory usage is essentially one more vector than that of the power method, which makes the memory usage of the filtered method practical for very large n.

4 Algorithm derivation and Analysis

Algorithms 3.3 and 3.2 construct filters that dampen on the interval $[0, u_l]$ for LL^{T}. In fact it is straightforward to construct Chebyshev filters that dampen on a general interval $[a, b]$ for a general hermitian matrix A. Now the affine mapping $\mathcal{L}(t)$ that maps $[a, b]$ into $[-1, 1]$ is

$$\mathcal{L}(t) = \frac{t - c}{e}, \quad \text{where } \mathrm{c} = \frac{\mathrm{b} - \mathrm{a}}{2},\ \mathrm{e} = \frac{\mathrm{b} + \mathrm{a}}{2}\ .$$

Any interval $[a, b]$ is uniquely determined by its center c and its half-width e.

The 3-term recurrence for updating Chebyshev polynomials (7) applied to $\mathcal{L}(t)$ is

$$C_{k+1}(\mathcal{L}(t)) = 2\mathcal{L}(t)\ C_k(\mathcal{L}(t)) - C_{k-1}(\mathcal{L}(t)), \qquad t \in \mathbb{R},\ k = 1, 2, ..., \tag{10}$$

with $C_0(t) \equiv 1$ and $C_1(\mathcal{L}(t)) = \mathcal{L}(t)$.

Define $x_k := C_k(\mathcal{L}(A))x_0$ for any initial vector x_0, (note $x_1 = \frac{1}{e}(A - cI)x_0$), then (10) leads to the iteration

$$x_{k+1} = C_{k+1}(\mathcal{L}(A))x_0 = \frac{2}{e}(A - cI)x_k - x_{k-1}, \quad k = 1, 2, ...\ . \tag{11}$$

Stacking two consecutive vectors together, iteration (11) can be written as

$$\begin{bmatrix} x_k \\ x_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I \\ -I & \frac{2}{e}(A - cI) \end{bmatrix}}_{\mathcal{B}} \begin{bmatrix} x_{k-1} \\ x_k \end{bmatrix}\ . \tag{12}$$

Equation (12) shows that the Chebyshev iteration (11) is essentially a power iteration on a transformed matrix $\mathcal{B}$. The convergence of the Chebyshev iteration can be understood by the eigenvalues of $\mathcal{B}$.

First we need the following lemma.

Lemma 4.1. *Let M be an $n \times n$ matrix with eigenvalues $d_i, i = 1, ..., n$. Then the eigenvalues of the $2n \times 2n$ matrix $\begin{bmatrix} 0 & I \\ -I & M \end{bmatrix}$ are $\frac{d_i \pm \sqrt{d_i^2 - 4}}{2}$, i = 1, 2, ..., n .*

Proof: The matrix $\begin{bmatrix} 0 & I \\ -I & M \end{bmatrix}$ is full rank, therefore it has no zero eigenvalues. Let μ's be its eigenvalues, then $\mu \neq 0$. By some properties of determinant, especially that $det \begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix} = det(D_{11})det(D_{22} - D_{21}D_{11}^{-1}D_{12})$, where D_{11} is nonsingular and D_{ij} are square, we get

$$det \begin{pmatrix} -\mu I & I \\ -I & M - \mu I \end{pmatrix} = det(-\mu M + \mu^2 I + I) = \prod_{i=1}^{n} (\mu^2 - d_i\mu + 1)\ .$$

Therefore the eigenvalues μ's should be the roots of $\mu^2 - d_i\mu + 1 = 0$, which are $\frac{d_i \pm \sqrt{d_i^2 - 4}}{2}$, $i = 1, ..., n$. ∎

Denote the eigenvalues of A as $\lambda_i(A)$, we can apply Lemma 4.1 to $M = \frac{2}{e}(A - cI)$ to get the eigenvalues of $\mathcal{B}$ as

$$\lambda_i^{(1,2)}(\mathcal{B}) = \frac{d_i \pm \sqrt{d_i^2 - 4}}{2}, \quad \text{where} \quad d_i = \frac{2}{e}(\lambda_i(A) - c), \ i = 1, 2, ..., n. \tag{13}$$

If $\lambda_i(A) \in [a, b]$, then $\frac{2}{e}(\lambda_i(A) - c) \leq 2$, the corresponding $\lambda_i^{(1,2)}(\mathcal{B})$ are complex conjugate and $|\lambda_i^{(1,2)}(\mathcal{B})| = 1$. While if $\lambda_i(A) \in (-\infty, a) \cup (b, \infty)$, then the corresponding $\lambda_i^{(1,2)}(\mathcal{B})$ are real and the one with larger magnitude is located outside the $[-1, 1]$ interval. This easily establishes the convergence of the Chebyshev iteration.

Lemma 4.2. *Assume that the largest eigenvalue of the hermitian matrix A is unique, assume that the filtering interval $[a, b]$ satisfies $a \leq \min_{i=1,...,n} \lambda_i(A)$ and $b < \max_{i=1,...,n} \lambda_i(A)$. Then the x_k from the Chebyshev iteration (11) converges to the principal eigenvector of A.*

Proof: Let the eigenvalues of A in nonincreasing order be $\lambda_1(A) > \lambda_2(A) \geq ... \geq \lambda_n(A)$, with corresponding d_i defined as in (13). Since $a \leq \lambda_n(A)$ and $b < \lambda_1(A)$, by lemma 4.1 we have $\max_{i=1,...,n} \lambda_i^{(1,2)}(\mathcal{B}) = \frac{d_1 + \sqrt{d_1^2 - 4}}{2} > 1$, which is the unique principal eigenvalue of $\mathcal{B}$. Thus from the power iteration (12) we get the convergence of x_k, which is equivalent to the convergence of x_k in (11). Furthermore, x_k converges to the principal eigenvector of A because $x_k = C_k(\mathcal{L}(A))x_0 = C_k(\frac{1}{e}(A - cI))x_0$, and both scaling and shifting by a constant to A do not change its eigenvectors. ∎

The Chebyshev iteration (11) can be further scaled for stability purpose. A simple strategy discussed in [30, p. 223] is to replace the $C_k(\mathcal{L}(A))$ used in (11) for calculating $x_k = C_k(\mathcal{L}(A))x_0$ by

$$\tilde{C}_k(\mathcal{L}(A)) := \frac{C_k\left(\frac{1}{e}(A - cI)\right)}{\rho_k}, \qquad \text{where} \quad \rho_k := C_k\left(\frac{1}{e}(\tilde{b} - c)\right). \tag{14}$$

Here $\tilde{b}$ is a value outside $[a, b]$. In this case $|\frac{1}{e}(\tilde{b} - c)| > 1$ and the scaling factor ρ_k will increase exponentially as k increases, enhancing numerical stability of the Chebyshev filters. However, if ρ_k is made too large (e.g., if the eigenvalues of $\tilde{C}_k(\mathcal{L}(A))$ all become negligibly small), then the application of $\tilde{C}_k(\mathcal{L}(A))$ becomes insignificant, which is counter-productive to convergence acceleration. This means that $|\tilde{b}|$ cannot be much larger than $\lambda_{\max}(LL^{\mathrm{T}})$.

Using same technique as in [30, p. 223], let $\sigma_{k+1} := \rho_k/\rho_{k+1}$, we get the 3-term recurrence for the scaled Chebyshev polynomial (14),

$$\tilde{C}_{k+1}(\mathcal{L}(A)) = 2\sigma_{k+1}\frac{A - cI}{e}\tilde{C}_k(\mathcal{L}(A)) - \sigma_{k+1}\sigma_k\tilde{C}_{k-1}(\mathcal{L}(A)), \ k = 1, 2, ... \tag{15}$$

The formula for updating σ_k is straightforward to derive using (10),

$$\sigma_1 = \frac{\rho_0}{\rho_1} = \frac{e}{\tilde{b} - c}, \quad \sigma_{k+1} = \frac{C_k(\frac{1}{e}(\tilde{b} - c))}{C_{k+1}(\frac{1}{e}(\tilde{b} - c))} = \frac{1}{2/\sigma_1 - \sigma_k}. \tag{16}$$

Starting with a given $\tilde{x}_0$ and $\tilde{x}_1 = \tilde{C}_1(\mathcal{L}(A))\tilde{x}_0 = \frac{\sigma_1}{e}(A - cI)\tilde{x}_0$, applying (16) and (15), the Chebyshev filtering $\tilde{x}_{k+1} = \tilde{C}_k(\mathcal{L}(A))\tilde{x}_0$ on a given interval $[a, b]$ can be realized as

$$\tilde{x}_{k+1} = 2\frac{\sigma_{k+1}}{e}(A - cI)\tilde{x}_k - \sigma_{k+1}\sigma_k\tilde{x}_{k-1}, \quad k = 1, 2, \ldots . \tag{17}$$

We can immediately express (17) as the following power iteration

$$\begin{bmatrix} \tilde{x}_k \\ \tilde{x}_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I \\ -\sigma_{k+1}\sigma_k I & \frac{2\ \sigma_{k+1}}{e}(A - cI) \end{bmatrix}}_{\tilde{\mathcal{B}}} \begin{bmatrix} \tilde{x}_{k-1} \\ \tilde{x}_k \end{bmatrix} . \tag{18}$$

Using same technique and argument as that for Lemmas 4.1 and 4.2, we readily have the following result:

Lemma 4.3. *The eigenvalues of the matrix $\tilde{\mathcal{B}}$ in (18) are $\lambda_i^{(1,2)}(\tilde{\mathcal{B}}) = \frac{\sigma_{k+1}d_i \pm \sqrt{\sigma_{k+1}^2 d_i^2 - 4\sigma_{k+1}\sigma_k}}{2}$, where $d_i = \frac{2}{e}(\lambda_i(A) - c)$, $i = 1, 2, \ldots, n$. Under the same condition as in Lemma 4.2, and that $\tilde{b} > b$, then the x_k from the Chebyshev iteration (17) converges to the principal eigenvector of A.*

Algorithm 3.3 implements the scaled iteration (17), with $A = LL^{\mathrm{T}}$, $[a, b] = [0, u_l]$, and $\tilde{b} = u_L$, where $0 < u_l < \lambda_{\max}(LL^{\mathrm{T}})$, and $u_l < u_L$.

Since u_L (or $\tilde{b}$) is used only for scaling purpose, we can make a simplification by passing $u_L = u_l$ to Algorithm 3.3. In this case we only need to guarantee that $0 < u_l < \lambda_{\max}(LL^{\mathrm{T}})$, which is true by the bound choices of u_l discussed in section 3.1. Now that $u_l < u_L$ is not kept true, the scaling factor is $\rho_k = C_k(1) = 1$, which corresponds to the non-scaled iteration (11). In this case the scaling is actually done by the normalization step (*step 5*) in Algorithm 3.4. This normalization in practice is enough to avoid any potential overflow problem, especially when the polynomial degree m is relatively small.

Algorithm 3.2 implements the simplified Chebyshev filter with damping bounds 0 and u_l. The output u_u from Algorithm 3.2 or Algorithm 3.3 is used to update u_l according to (6).

By some algebra and applying Lemmas 4.2 and 4.3, we can readily establish the convergence of the main Algorithm 3.4. This is because Algorithm 3.4 is repeated application of Algorithm 3.3 or Algorithm 3.2 with normalization, while Algorithms 3.3 and 3.2 are essentially Chebyshev iterations — with suitably chosen filtering bounds, they are guaranteed to converge.

Theorem 4.1. *Assume LL^T has a unique principal eigenvalue. Then Algorithm 3.4 produces a vector that converges to the principal eigenvector of LL^T.*

Proof: Denote the filtering upper bound at the j-th step as u_{l_j}, then by property of Rayleigh quotients, the u_{l_j}'s generated in Algorithm 3.4 for calling the Chebyshev filters (Algorithms 3.3 and 3.2) always satisfy $0 < u_{l_j} < \lambda_{\max}(LL^{\mathrm{T}})$ and $u_l < u_L$.

Denote $\mathcal{L}_j(t) = (t - \frac{u_{l_j}}{2}) * \frac{2}{u_{l_j}}$. The unique largest eigenvalue of LL^{T} is guaranteed to be mapped into the unique largest eigenvalue of the filtered matrix: For Algorithm 3.2, the matrix is $C_m(\mathcal{L}_j(LL^{\mathrm{T}}))$; for Algorithm 3.3, it is the scaled matrix $\frac{C_m(\mathcal{L}_j(LL^{\mathrm{T}}))}{C_m(\mathcal{L}_j(u_L))}$.

Then, by repeated application of Lemma 4.2 and Lemma 4.3 (essentially , each of them is a power method applied to a dynamically updated filtered matrix), we conclude that Algorithm 3.4 converges to the principal eigenvector of LL^{T}. ∎

As the converge factor is concerned, at the j-th iteration of Algorithm 3.4 with damping bounds 0 and u_{l_j}, the gap-ratio of the degree-m polynomial filtered matrix, $C_m(\mathcal{L}_j(LL^{\mathrm{T}}))$, is $\xi_{m_j} = \dfrac{\max\limits_{i\neq 1}\left|C_m\left(\frac{2}{u_{l_j}}\left(\lambda_i(LL^{\mathrm{T}})-\frac{u_{l_j}}{2}\right)\right)\right|}{C_m\left(\frac{2}{u_{l_j}}\left(\lambda_1(LL^{\mathrm{T}})-\frac{u_{l_j}}{2}\right)\right)}$, where $\lambda_1(LL^{\mathrm{T}})$ is assumed as before the unique largest eigenvalue of LL^{T}. (The scaling of the filtered matrix, $\frac{C_m(\mathcal{L}_j(LL^{\mathrm{T}}))}{C_m(\mathcal{L}_j(u_L))}$ as in Algorithm 3.4, has the same gap-ratio.) Thanks to the fastest growth outside $[-1,1]$ property of Chebyshev polynomial among degree-m polynomials, it is quite easy to make ξ_{m_j} smaller than the gap-ratio of the power method. If we denote $\xi_m = \sup_j \xi_{m_j}$, then it takes $O(\ln\tau/\ln\xi_m)$ iteration steps for Algorithm 3.4 to bring the error below a given tolerance τ.

Ideally, at iteration step j one would wish to find an optimal u_{l_j} for filtering such that the gap-ratio ξ_{m_j} will be minimal for a given degree-m. However, this is not easy since the $\lambda_i(LL^{\mathrm{T}})$'s are unknown. The convex combination (6) that we use for adaptively adjusting u_l is simple and convenient to realize, it also works well in practice. In the next section we present numerical performance of Algorithm 3.4 and compare with the power method which is the *de facto* standard method for HITS ranking calculations.

5 Numerical Results

Adjacency matrices of some realistic web graphs are used for the numerical tests. Relevant information of these matrices are listed in Table 1. Except the commonly used Stanford-Berkeley matrix, the remaining matrices[†] were generated using UbiCrawler [5, 6].

Graph-name	dimension (#nodes)	`nnz` (#arcs)	sparsity	λ_2/λ_1
stanford-berkeley	683,446	7,583,376	1.624e-05	0.9787
eu-2005	862 664	19,235,140	2.585e-05	0.7226
wikipedia-2006-09	2,983,494	37,269,096	4.187e-06	0.7559
wikipedia-2006-11	3,148,440	39,383,235	3.973e-06	0.7733
wikipedia-2007-02	3,566,907	45,030,389	3.539e-06	0.8119
arabic-2005	22,744,080	639,999,458	1.237e-06	0.6236
uk-2007-05	105,896,555	3,738,733,648	3.334e-07	0.8596
webbase-2001	118,142,155	1,019,903,190	7.307e-08	0.7971

Table 1: Dimension, number of nonzeros, and approximate sparsity of the adjacency matrices for some realistic web graphs. Each graph-name contains the name of the web domain crawled. Except the first graph, the graph-name also shows when the crawl was performed. The gap-ratio λ_2/λ_1 is listed only as reference since it is unknown beforehand.

[†] Available at http://law.dsi.unimi.it and http://www.cise.ufl.edu/research/sparse/mat/Gleich/.

The gap-ratio $\lambda_2(LL^{\mathrm{T}})/\lambda_1(LL^{\mathrm{T}})$ of each test matrix listed in Table 1 is computed by Matlab `eigs`, which provides mex interface to ARPACK [32, 23]. This ratio is generally not available beforehand without extra computation, therefore it is listed only for reference and it is not used for the design of our algorithm. An interesting observation from the calculation by `eigs` is that $\lambda_{\max}(LL^{\mathrm{T}})$ is unique for all these matrices, hence the primitive trick need not be applied.

The calculations were performed in Matlab on a Dell-R710 computer with two quad-core Intel X5550 Xeon CPU (clock speed 2.66 GHz) and 144 Gb RAM, at the SMU HPC center.

To accelerate the matrix vector products we use the Bvgraph package [12], which provides convenient mex interfaces for processing the adjacency matrices generated by UbiCrawler [5, 6].

For all the numerical experiments using Algorithm 3.4, the performance of the scaled filter (Algorithm 3.3) is close to identical with the simplified filter (Algorithm 3.2). Therefore in Figures 1 and 2, we only present results using the simplified filter in Algorithm 3.4.

Essentially there are two parameters to provide to Algorithm 3.4: m and β. These two parameters largely determine the performance of the algorithm. For the adjacency matrices of all the web graphs that we had access to, a small m ($4 \leq m \leq 6$) and $\beta \in (0.75, 0.85)$ usually provide decent acceleration over the power method. One exception is the Stanford-Berkeley matrix, for which far better acceleration is achieved using a small β such as $\beta = 0.2$, which means Algorithm 3.4 prefers u_l to be increased fast instead of slowly for this problem.

In Figures 1 and 2, the `cpu` records CPU seconds, and the `mvp` counts total number of matrix-vector products. The reported `mvp` count and CPU time for the filtered method (`cheb`) include the call to the Lanczos bound estimator (Algorithm 3.1), whose cost is insignificant since it performs only three Lanczos steps.

The convergence tolerance is set to 10^{-10}, and the initial vector set to the vector with all elements one. As seen from these plots, if a larger tolerance is used, say 10^{-6}, the acceleration scale over the power method is consistent with the scale obtained by using a lower tolerance.

The error labeled as "residual norm" denotes the norm of the difference between two consecutive vectors $\|x_{k+1} - x_k\|_1$, obtained at `step` 6 of Algorithm 3.4 when determining convergence.

The gap-ratio in Table 1 shows that the Stanford-Berkeley matrix is difficult for the power method. It is in this scenario that acceleration by suitable filtering is particularly desirable. The first subgraph in Figure 1 shows that Algorithm 3.4 is about ten times faster than the power method. Moreover, even for the cases where the gap-ratio is below 0.85, for which the power method performs reasonably well, Algorithm 3.4 still can be about twice faster. This is seen from Figures 1 and 2.

6 Concluding Remarks

We presented an acceleration scheme based on adaptive Chebyshev polynomials for HITS ExpertRank vector calculations. The adaptiveness is realized by a simple convex combination (6) of two readily available bounds at each filtering step. Power method is the standard practical approach for HITS calculations, our proposed scheme can be considered as filter accelerated power method, therefore it can be applied to accelerate convergence where a power method is applicable. The filtered scheme provides consistent speedup, using mainly one more vector than that of a power method. The speedup is significant especially when the gap-ratio is close to 1

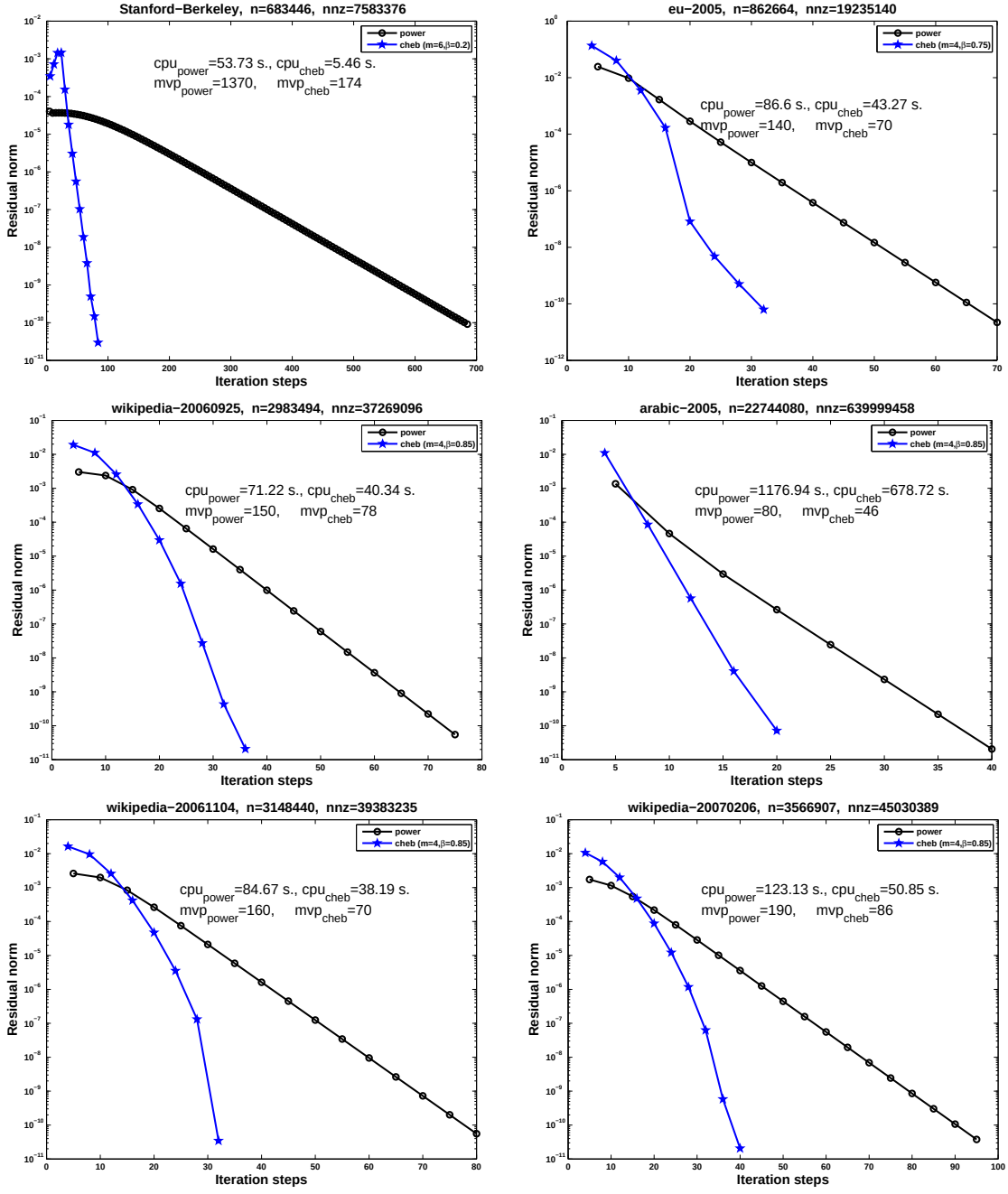

Figure 1: Acceleration of the filtered method over the power method on several web graphs listed in Table 1. The filtered method is usually twice faster than the power method.

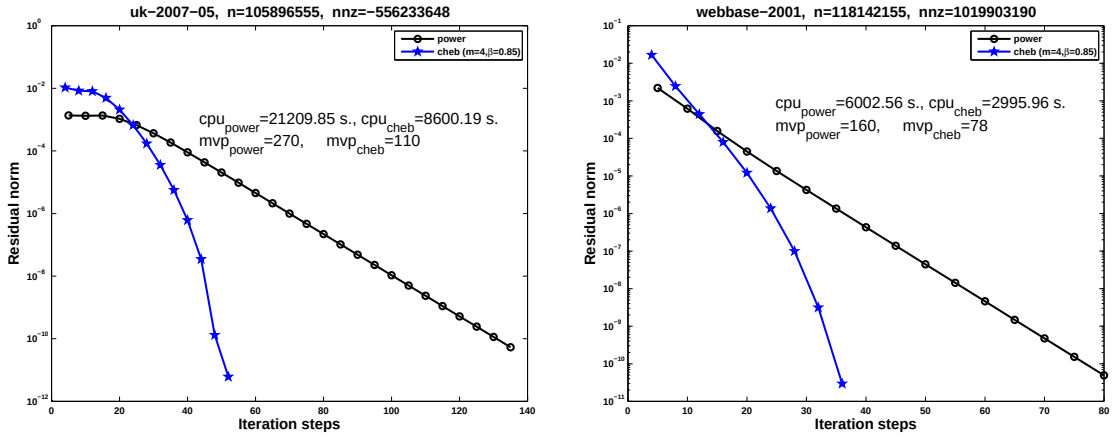

Figure 2: Acceleration on two relatively large web graphs with over 100 million nodes and over 1 billion arcs. The `nnz` count from Bvgraph for `uk-2007-05` is incorrect, possibly because the `nnz` is so large that it caused a scalar overflow. But this count has no impact on the accuracy since Bvgraph processes the matrix vector products correctly.

for which power method converges slowly.

The acceleration scheme requires two parameters: the polynomial degree m and the convex combination scalar β as in (6). Both parameters are quite easy to select in practice. However, analytical formulas that can guide the optimal choices of these parameters appear difficult to derive. It may be more practical to construct adaptive procedures to adjust these parameters during the iteration process, this will further enhance the acceleration but potentially make the algorithm complicated. Another direction for future work is to extend the acceleration scheme for approximating principal singular vectors, this has many applications in large scale statistical computing and data mining where principal singular vectors play an important role (e.g., in [8, 2, 24]).

Acknowledgement: We thank the authors of the open software Bvgraph[12] and UbiCrawler [5, 6]. Without the efficiency provided by these packages, especially the convenient functionalities of Bvgraph, the calculations in Matlab for the large matrices reported in this manuscript would not be possible.

References

[1] O. Axelsson. *Iterative Solution Methods.* Cambridge Univ. Press, 1994.

[2] M.-A. Belabbas and P. J. Wolfe. Spectral methods in machine learning and new strategies for very large datasets. *Proc. Natl. Acad. Sci.*, 106(2):369–374, 2009.

[3] C. Bishop. *Pattern Recognition and Machine Learning.* Information Science and Statistics. Springer, 2007.

[4] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Rev.*, 46(4):647–666, 2004.

[5] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[6] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[7] C. Brezinski and M. Redivo-Zaglia. Rational extrapolation for the PageRank vector. *Math. Comp.*, 77(263):1585–1598, 2008.

[8] J.F. Cai, E.J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. Optim.*, 20(4):1956–1982, 2010.

[9] C. H. Q. Ding, H. Zha, X. He, P. Husbands, and H. D. Simon. Link analysis: Hubs and authorities on the world wide web. *SIAM Rev.*, 46(2):256–268, 2004.

[10] L. Eldén. *Matrix Methods in Data Mining and Pattern Recognition*. SIAM, Philadelphia, PA, 2007.

[11] A. Farahat, T. LoFaro, J. C. Miller, G. Rae, and L. A. Ward. Authority Rankings from HITS, PageRank, and SALSA: Existence, Uniqueness, and Effect of Initialization. *SIAM J. Sci. Comput.*, 27(4):1181–1201, 2006.

[12] D. Gleich. The bvgraph class package, Updated July 2009. Available at http://www.mathworks.com/matlabcentral/fileexchange/16248.

[13] D. Gleich, L. Zhukov, and P. Berkhin. Fast parallel PageRank: A linaer system approach. Technical Report YRL-2004-038, Yahoo! Research Labs, 2004.

[14] D. F. Gleich, A. P. Gray, C. Greif, and T. Lau. An inner-outer iteration for computing pagerank. *SIAM J. Sci. Comput.*, 32(1):349–371, 2010.

[15] G. H. Golub and C. Greif. An Arnoldi-type algorithm for computing PageRank. *BIT*, 46(4):759–771, 2006.

[16] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2009.

[17] S.D. Kamvar, T.H. Haveliwala, and G.H. Golub. Extrapolation methods for accelerating pagerank computations. Technical report, Stanford University, 2003.

[18] S.D. Kamvar, T.H. Haveliwala, and G.H. Golub. Adaptive methods for the computation of PageRank. *Linear Algebra Appl.*, 386:51–65, 2004.

[19] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[20] A. N. Langville and C. D. Meyer. Deeper inside PageRank. *Internet Mathematics*, I(3):335–380, 2004.

[21] A. N. Langville and C. D. Meyer. A survey of eigenvector methods for web information retrieval. *SIAM Rev.*, 47(1):135–161, 2005.

[22] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

[23] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK User's Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, 1998.

[24] M. W. Mahoney and P. Drineas. CUR matrix decomposition for improved data analysis. *Proc. Natl. Acad. Sci.*, 106(3):697–702, 2009.

[25] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM Press, 2000.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[27] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Number 20 in Classics in Applied Mathematics. SIAM, Philadelphia, PA, 1998.

[28] T. J. Rivlin. *An Introduction to the Approximation of Functions*. (1969 1st edition), Dover, 2003.

[29] Y. Saad. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Math. Comp.*, 42(166):567–588, 1984.

[30] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. John Wiley, New York, 1992. Available at http://www.cs.umn.edu/~saad/books.html.

[31] J. R Seeley. The net of reciprocal influence: a problem in treating sociometric data. *Canadian Journal of Psychology*, 3(4):234–240, 1949.

[32] D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13:357–385, 1992.

[33] G. W. Stewart and J. G. Sun. *Matrix perturbation theory*. Academic Press Inc., Boston, MA, 1990.

[34] M. L. Tiago, Y. Zhou, M. Alemany, Y. Saad, and J. R. Chelikowsky. Evolution of magnetism in iron from the atom to the bulk. *Phys. Rev. Lett.*, 97:147201, 2006.

[35] S. Vigna. Spectral ranking. Technical report, Computer Science Dept., University of Milano, 2010.

[36] G. Wu and Y. Wei. A Power-Arnoldi algorithm for computing PageRank. *Numer. Linear Algebra Appl.*, 14(7):521–546, 2007.

[37] Y. Zhou. A block Chebyshev-Davidson method with inner-outer restart for large eigenvalue problems. *J. Comput. Phys.*, 229(24):9188–9200, 2010.

[38] Y. Zhou and R.-C. Li. Bounding the spectrum of large hermitian matrices. *Linear Algebra Appl.*, 435(3):480–493, 2011.

[39] Y. Zhou and Y. Saad. A Chebyshev-Davidson algorithm for large symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 29(3):954–971, 2007.

[40] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Parallel self-consistent-field calculations using Chebyshev-filtered subspace acceleration. *Phys. Rev. E*, 74(6):066704–1–8, 2006.

[41] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Self-consistent-field calculation using Chebyshev-filtered subspace iteration. *J. Comput. Phys.*, 219(1):172–184, 2006.