A Nearly-Sublinear Method for Approximating a Column of the Matrix Exponential for Matrices from Large, Sparse Networks

Kyle Kloster[1,⋆] and David F. Gleich[2,⋆]

[1] Purdue University, Mathematics Department
[2] Purdue University, Computer Science Department
{kkloste,dgleich}@purdue.edu
http://www.cs.purdue.edu/homes/dgleich/codes/nexpokit

Abstract. We consider random-walk transition matrices from large social and information networks. For these matrices, we describe and evaluate a fast method to estimate one column of the matrix exponential. Our method runs in sublinear time on networks where the maximum degree grows doubly logarithmic with respect to the number of nodes. For collaboration networks with over 5 million edges, we find it runs in less than a second on a standard desktop machine.

Keywords: Matrix exponential, Gauss-Southwell, local algorithms.

1 Introduction

The matrix exponential is a standard tool in network analysis. Its uses include node centrality [9,11,10], link-prediction [15], graph kernels [14], and clustering [8]. For the particular problems of node centrality, graph kernels, and clustering, what is most valuable is a coarse estimate of a column of the matrix exponential. In this paper, we consider computing $\exp\{\mathbf{P}\}\mathbf{e}_c$ where $\mathbf{P}$ is the random-walk transition matrix for a directed or undirected graph and $\mathbf{e}_c$ is the cth column of the identity matrix. More precisely, and to establish some notation for the paper, let $\mathbf{G}$ be an $n \times n$ adjacency matrix for a *directed or undirected* graph, let $\mathbf{e} = [1, \cdots, 1]^T$ the vector (of appropriate dimensions) of all 1s, and let $\mathbf{D} = \mathrm{diag}(\mathbf{G}\mathbf{e})$ so that $\mathbf{D}_{ii} = d_i$ is the degree of node i (and $d = \max_i\{d_i\}$ is the maximum degree). We consider $\mathbf{P} = \mathbf{G}\mathbf{D}^{-1}$. This case suffices for many of the problems studied in the literature and allows us to compute exponentials of the negative normalized Laplacian $-\hat{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{G}\mathbf{D}^{-1/2} - \mathbf{I}$ as well. Observe that

$$\begin{aligned}\exp\{\mathbf{D}^{-1/2}\mathbf{G}\mathbf{D}^{-1/2} - \mathbf{I}\}\mathbf{e}_c &= e^{-1}\mathbf{D}^{-1/2}\exp\{\mathbf{G}\mathbf{D}^{-1}\}\mathbf{D}^{1/2}\mathbf{e}_c \\ &= \sqrt{d_c}e^{-1}\mathbf{D}^{-1/2}\exp\{\mathbf{G}\mathbf{D}^{-1}\}\mathbf{e}_c,\end{aligned}$$

so computing a column of either $-\hat{\mathbf{L}}$ or $\mathbf{P}$ allows computation of the other at the cost of scaling the solution vector.

⋆ Supported by NSF CAREER award 1149756-CCF.

A. Bonato, M. Mitzenmacher, and P. Prałat (Eds.): WAW 2013, LNCS 8305, pp. 68–79, 2013.

Computing accurate matrix exponentials has a lengthy and "dubious" history [17]. Let $\mathbf{A}$ be a general $n \times n$ matrix that is large and sparse and let $\mathbf{b}$ be a general vector. A popular method for computing $\exp\{\mathbf{A}\}\mathbf{b}$ involves using an m-step Krylov approximation of the matrix $\mathbf{A}$ yielding $\mathbf{A} \approx \mathbf{V}_m\mathbf{H}_m\mathbf{V}_m^T$. If we use this form, we can approximate $\exp\{\mathbf{A}\}\mathbf{b} \approx \mathbf{V}_m\exp\{\mathbf{H}\}\mathbf{e}_1$. For $m \ll n$, the computation is reduced to the much smaller $\exp\{\mathbf{H}\}$ at the cost of using m matrix-vector products to generate $\mathbf{V}_m$. Such an approximation works well for computing both the entire exponential $\exp\{\mathbf{A}\}$ and its action on a vector: $\exp\{\mathbf{A}\}\mathbf{b}$. This idea underlies the implementation of the Matlab package ExpoKit [21], which has been a standard for computing $\exp\{\mathbf{A}\}\mathbf{b}$ for some time.

While these algorithms are fast and accurate (see references [13], [12], and [2], for the numerical analysis), they depend on matrix-vector products with the matrix $\mathbf{P}$ and orthogonalization steps between successive vectors. When a Krylov method approximates a sparse matrix arising from a graph with a small diameter, then the vectors involved in the matrix-vector products become dense after two or three steps, even if the vector starting the Krylov approximation has only a single non-zero entry. Subsequent matrix-vector products take $O(|E|)$ work where $|E|$ is the number of edges in the graph. For networks with billions of edges, we want an alternative to Krylov-based methods that prioritizes speed and sparsity over accuracy. In particular, we would like an algorithm to estimate a column of the matrix exponential in less work than a single matrix-vector product.

Local methods perform a computation by accessing a small region of a matrix or graph. These are a practical alternative to Krylov methods for solving massive linear systems from network problems that have sparse right hand sides; see, for instance, references [3,7]. Rather than matrix-vector products, these procedures use steps that access only a single row or column of the matrix. We design a local algorithm for computing $\exp\{\mathbf{P}\}\mathbf{e}_c$ by translating the problem of computing the exponential into solving a linear system, and then using a local algorithm.

We present an algorithm that approximates a specified column of $\exp\{\mathbf{P}\}$ for column stochastic $\mathbf{P}$ (Section 4, Figure 1). The algorithm uses the Gauss-Southwell method (Section 2) for solving a linear system to approximate a degree N Taylor polynomial (Section 3). The error after l iterations of the algorithm is bounded by $\frac{1}{N!N} + l^{-1/(2d)}$ as shown in Theorem 2, and the runtime is $O(ld + ld\log(ld))$ (Section 5.3). Given an input error ε, the runtime to produce a solution vector with error less than ε is sublinear in n for graphs with $d \leq O(\log\log n)$. We acknowledge that this doubly logarithmic scaling of the maximum degree is unrealistic for social and information networks where the maximum degree typically scales almost linearly with n. Nevertheless, the existence of a bound suggests that it may be possible to improve or establish a matching lower-bound.

2 Local Computations and the Gauss-Southwell Method

The Gauss-Southwell (GS) iteration is a classic stationary method for solving a linear system related to the Gauss-Seidel and coordinate descent methods [16]. It is especially efficient when the desired solution vector is sparse or localized [7,5]

and the goal is a coarse $O(10^{-4})$ approximation. In these cases, GS produces a sparse approximation that is accurate for only the largest entries. The coarse nature of the GS approximation is acceptable because the primary use is to find those nodes having the largest values in link-prediction or clustering problems.

The GS iteration is simple. Select the largest entry in the residual vector and perform a coordinate descent step in that component of the solution. Let $\mathbf{x}^{(l)}$ and $\mathbf{r}^{(l)}$ be the solution and residual iterates for $\mathbf{Ax} = \mathbf{b}$ after l steps. In the $(l+1)$st step, pick q so that $m_l = \mathbf{r}_q^{(l)}$ is the maximum magnitude entry of the residual vector. Next update:

$$\begin{aligned} \mathbf{x}^{(l+1)} &= \mathbf{x}^{(l)} + m_l \mathbf{e}_q \\ \mathbf{r}^{(l+1)} &= \mathbf{r}^{(l)} - m_l \mathbf{A}\mathbf{e}_q. \end{aligned} \tag{1}$$

The iteration consists of a single-entry update to $\mathbf{x}$ and a few updates to the residual vector if $\mathbf{A}$ is sparse. This method converges for diagonally dominant and symmetric positive definite linear systems [16].

Applied to a matrix from a graph, each iteration of GS requires at most $O(d \log n)$ operations, where the $\log n$ term comes from heap updates to maintain the largest residual entry. This procedure underlies Berkhin's bookmark coloring algorithm [5] for PageRank and a related method avoids the heap [3].

3 Exponentials via the Taylor Series Approximation

The GS method is most effective on sparse linear systems. We now design a large, sparse linear system to compute a Taylor polynomial appromation of $\exp\{\mathbf{P}\}\mathbf{e}_c$.

3.1 The Truncated Taylor Series of the Exponential

The Taylor series for the matrix $\exp\{\mathbf{A}\}$ is

$$\exp\{\mathbf{A}\} = \sum_{k=0}^{\infty} \tfrac{1}{k!} \mathbf{A}^k.$$

It converges for any matrix $\mathbf{A}$. Truncating to N terms, we arrive at an algorithm. If $\|\mathbf{A}\|$ is large with mixed sign then the summands $\mathbf{A}^k$ may be large and cancel only in exact arithmetic, resulting in poor accuracy. However, a stochastic matrix $\mathbf{P}$ is non-negative and has $\|\mathbf{P}\|_1 = 1$, so the approximation converges quickly and reliably (Lemma 1). Using an N-degree Taylor approximation to compute the cth column results in a simple iteration. Let $\mathbf{x}_N$ be the N-degree Taylor approximation:

$$\mathbf{x}_N = \sum_{k=0}^{N} \tfrac{1}{k!} \mathbf{P}^k \mathbf{e}_c \approx \exp\{\mathbf{P}\}\mathbf{e}_c.$$

Then

$$\mathbf{x}_N = \sum_{k=0}^{N} \mathbf{v}_k \quad \mathbf{v}_0 = \mathbf{e}_c, \quad \mathbf{v}_1 = \mathbf{P}\mathbf{v}_0, \quad \mathbf{v}_{k+1} = \mathbf{P}\mathbf{v}_k/k \quad \text{for } k = 1, \cdots, N.$$

If $\mathbf{x}_{\text{true}}$ is the actual column of $\exp\{\mathbf{P}\}$ we are trying to compute, note that $\mathbf{x}_N$ converges to $\mathbf{x}_{\text{true}}$ as N tends to infinity. For practical purposes, we want to ensure that $\|\mathbf{x}_N - \mathbf{x}_{\text{true}}\|_1$ is small so that our approximation of $\mathbf{x}_N$ is near $\mathbf{x}_{\text{true}}$. The next Lemma shows that $N = 11$ yields a 1-norm error of 2.3×10^{-9}. This is sufficiently small for our purposes and from now on, we use $N = 11$ throughout.

Lemma 1. *The degree N Taylor approximation satisfies $\|\mathbf{x}_{true} - \mathbf{x}_N\|_1 \leq \frac{1}{N!N}$.*

Proof. The truncation results in a simple error analysis:

$$\|\mathbf{x}_{\text{true}} - \mathbf{x}_N\|_1 = \left\| \sum_{k=N+1}^{\infty} \frac{\mathbf{P}^k}{k!} \mathbf{e}_c \right\|_1 = \sum_{k=N+1}^{\infty} \frac{1}{k!}, \tag{2}$$

which follows because $\mathbf{P}$ and $\mathbf{e}_c$ are nonnegative and $\mathbf{P}$ is column stochastic. By factoring out $\frac{1}{(N+1)!}$ and majorizing $\frac{(N+1)!}{(N+1+k)!} \leq \left(\frac{1}{N+1}\right)^k$ for $k \geq 0$, we finish:

$$\|\mathbf{x}_{\text{true}} - \mathbf{x}_N\|_1 \leq \left(\frac{1}{(N+1)!}\right) \sum_{k=0}^{\infty} \left(\frac{1}{N+1}\right)^k = \frac{1}{(N+1)!} \frac{N+1}{N} \tag{3}$$

after substituting the limit for the convergent geometric series. □

3.2 Forming a Linear System

We now devise a linear system to compute the intermediate terms $\mathbf{v}_k$ and $\mathbf{x}_N$. From the identity $\mathbf{v}_{k+1} = \frac{\mathbf{P}}{k+1} \cdot \mathbf{v}_k$ we see that the $\mathbf{v}_k$ satisfy

$$\begin{bmatrix} \mathbf{I} & & & & \\ -\mathbf{P}/1 & \mathbf{I} & & & \\ & -\mathbf{P}/2 & \ddots & & \\ & & \ddots & \mathbf{I} & \\ & & & -\mathbf{P}/N & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \\ \vdots \\ \mathbf{v}_N \end{bmatrix} = \begin{bmatrix} \mathbf{e}_c \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}. \tag{4}$$

For convenience of notation, let $\mathbf{S}_{N+1}$ be the $(N+1) \times (N+1)$ zero matrix with first subdiagonal $[\frac{1}{1}, \frac{1}{2}, \cdots, \frac{1}{N}]$. Let $\mathbf{v} = [\mathbf{v}_0; \cdots; \mathbf{v}_N]$. Then we can rewrite (4):

$$(\mathbf{I}_{N+1} \otimes \mathbf{I}_n - \mathbf{S}_{N+1} \otimes \mathbf{P})\, \mathbf{v} = \mathbf{e}_1 \otimes \mathbf{e}_c. \tag{5}$$

The left- and right-hand sides of (5) are sparse, making this linear system a candidate for a Gauss-Southwell iteration. Note also that we need never form this large block-wise system explicitly and can work with it implicitly. Each row of the system is uniquely defined by a node index i and a step index k.

We now show that approximating $\mathbf{v}$ will help approximate $\mathbf{x}_N$. Let $\mathbf{M} = \mathbf{I}_{N+1} \otimes \mathbf{I}_n - \mathbf{S}_{N+1} \otimes \mathbf{P}$ from (5). With an approximation $\hat{\mathbf{v}}$ to the solution $\mathbf{v}$, we can approximate $\mathbf{x}_N$ by summing components of $\hat{\mathbf{v}}$: $\hat{\mathbf{x}}_N = \sum_{k=0}^{N} \hat{\mathbf{v}}_k$. Given that our primary purpose is computing $\mathbf{x}_N$, we want to know how accurately $\hat{\mathbf{x}}_N$ approximates $\mathbf{x}_N$. With that in mind, we state the following:

Lemma 2. *If* $\hat{\mathbf{v}} \leq \mathbf{v}$ *component-wise, then* $\|\mathbf{x}_N - \hat{\mathbf{x}}_N\|_1 = \|\mathbf{v} - \hat{\mathbf{v}}\|_1$.

Proof. The vector $\mathbf{x}_N$ is the sum of the block-vectors composing $\mathbf{v} = [\mathbf{v}_0; \cdots; \mathbf{v}_N]$, and similarly $\hat{\mathbf{x}}_N$ is the sum of the block-vectors of $\hat{\mathbf{v}}$. Thus, $\mathbf{e}^T\mathbf{v} = \mathbf{e}^T\mathbf{x}_N$, and $\mathbf{e}^T\hat{\mathbf{v}} = \mathbf{e}^T\hat{\mathbf{x}}_N$. Since $\hat{\mathbf{v}}$ approaches $\mathbf{v}$ from below (by assumption), we have that $\|\mathbf{x}_N - \hat{\mathbf{x}}_N\|_1 = \mathbf{e}^T(\mathbf{x}_N - \hat{\mathbf{x}}_N) = \mathbf{e}^T(\mathbf{v} - \hat{\mathbf{v}}) = \|\mathbf{v} - \hat{\mathbf{v}}\|_1$. □

4 Approximating the Taylor Polynomial with GS

We apply Gauss-Southwell (GS) to $\mathbf{M}\mathbf{v} = \mathbf{e}_1 \otimes \mathbf{e}_c$ staring with $\mathbf{x}^{(0)} = 0$. The block structure of $\mathbf{M}$ makes the solution and residual update simple. Recall that q was the index of the largest entry in the residual and m_l was the value of the entry. Let $\mathbf{e}_q = \mathbf{e}_k \otimes \mathbf{e}_i$ where $\mathbf{e}_k$ is a length $N+1$ vector indicating the step number and $\mathbf{e}_i$ indicates the node number. By substituting this into (1), we find:

$$\mathbf{v}^{(l+1)} = \mathbf{v}^{(l)} + m_l(\mathbf{e}_k \otimes \mathbf{e}_i) \tag{6}$$

$$\mathbf{r}^{(l+1)} = \mathbf{r}^{(l)} - m_l\mathbf{M}(\mathbf{e}_k \otimes \mathbf{e}_i). \tag{7}$$

Note that (6) simply adds $m_l\mathbf{e}_i$ to the block of $\mathbf{v}$ corresponding to $\mathbf{v}_{k-1}$. However, since we intend to add together the $\mathbf{v}_i$ at the end (to produce the full Taylor approximation), in practice we simply add $m_l\mathbf{e}_i$ directly to to our matrix-exponential-column approximation vector $\hat{\mathbf{x}}^{(l)}$. Thus we satisfy the requirements of Lemma 2. When $k < N+1$ the residual update can also be adapted using

$$\mathbf{M}(\mathbf{e}_k \otimes \mathbf{e}_i) = \mathbf{e}_k \otimes \mathbf{e}_i - (\tfrac{1}{k}\mathbf{e}_{k+1} \otimes (\mathbf{P}\mathbf{e}_i)) \tag{8}$$

In the case that $k = N+1$, then $\mathbf{S}_{N+1}\mathbf{e}_{N+1} = 0$, so we have simply $\mathbf{M}(\mathbf{e}_{N+1} \otimes \mathbf{e}_i) = \mathbf{e}_{N+1} \otimes \mathbf{e}_i$. Substituting (8) into the residual update in (7) gives

$$\mathbf{r}^{(l+1)} = \mathbf{r}^{(l)} - m_l\mathbf{e}_k \otimes \mathbf{e}_i + \tfrac{m_l}{k}(\mathbf{e}_{k+1} \otimes \mathbf{P}\mathbf{e}_i). \tag{9}$$

Because the indices k and i are chosen so that the entry in the vector $\mathbf{e}_k \otimes \mathbf{e}_i$ corresponds to the entry of $\mathbf{r}^{(l)}$ that is largest, we have $m_l = (\mathbf{e}_k \otimes \mathbf{e}_i)^T\mathbf{r}^{(l)}$. Thus, $(\mathbf{e}_k \otimes \mathbf{e}_i)^T\mathbf{r}^{(l+1)} = (\mathbf{e}_k \otimes \mathbf{e}_i)^T\mathbf{r}^{(l)} - m_l(\mathbf{e}_k \otimes \mathbf{e}_i)^T\mathbf{e}_k \otimes \mathbf{e}_i = 0$, and this iteration zeros out the largest entry of $\mathbf{r}^{(l)}$ at each step. See Figure 1 for working code.

5 Convergence Results for Gauss-Southwell

Our convergence analysis has two stages. First, we show that the algorithm in Figure 1 produces a residual that converges to zero (Theorem 1). Second, we establish the rate at which the error in the computed solution $\hat{\mathbf{x}}_N$ converges to zero (Theorem 2). From this second bound, we arrive at a sublinear runtime bound in the case of a slowly growing maximum degree.

```
function x = gexpm(P,c,tol)
n = size(P,1); N = 11; sumr=1;
r = zeros(n,N+1); r(c,1) = 1; x = zeros(n,1);    % the residual and solution
while sumr >= tol % use max iteration too
  [ml,q]=max(r(:)); i=mod(q-1,n)+1; k=ceil(q/n); % use a heap in practice for max
  r(q) = 0; x(i) = x(i)+ml; sumr = sumr-ml;% zero the residual, add to solution
  [nset,~,vals] = find(P(:,i)); ml=ml/k;   % look up the neighbors of node i
  for j=1:numel(nset)                                % for all neighbors
    if k==N, x(nset(j)) = x(nset(j)) + vals(j)*ml;     % add to solution
    else, r(nset(j),k+1) = r(nset(j),k+1) + vals(j)*ml;% or add to next residual
          sumr = sumr + vals(j)*ml;
end, end, end                                % end if, end for, end while
```

Fig. 1. Pseudo-code for our nearly sublinear time algorithm as Matlab code. In practice, the solution vector `x` and residual `r` should be stored as hash-tables, and the entries of the residual as a heap. Note that the command `[nset,~,vals] = find(P(:,i))` returns the neighbors of the ith node (`nset`) along with values of $\mathbf{P}$ for those neighbors.

5.1 Convergence of the Residual

Theorem 1. *Let* $\mathbf{P}$ *be column-stochastic and* $\mathbf{v}^{(0)} = 0$. *(Nonnegativity) The iterates and residuals are nonnegative:* $\mathbf{v}^{(l)} \geq 0$ *and* $\mathbf{r}^{(l)} \geq 0$ *for all* $l \geq 0$. *(Convergence) The residual satisfies the following bound and converges to 0:*

$$\|\mathbf{r}^{(l)}\|_1 \leq \prod_{k=1}^{l} \left(1 - \frac{1}{2dk}\right) \leq l^{\left(-\frac{1}{2d}\right)} \tag{10}$$

Proof. (Nonnegativity) Since $\mathbf{v}^{(0)} = 0$ we have $\mathbf{r}^{(0)} = \mathbf{e}_1 \otimes \mathbf{e}_c - \mathbf{M} \cdot 0 = \mathbf{e}_1 \otimes \mathbf{e}_c \geq 0$, establishing both base cases. Now assume by way of induction that $\mathbf{v}^{(l)} \geq 0$ and $\mathbf{r}^{(l)} \geq 0$. Then the GS update gives $\mathbf{v}^{(l+1)} = \mathbf{v}^{(l)} + m_l \mathbf{e}_k \otimes \mathbf{e}_i$, and since $m_l \geq 0$ (because it is taken to be the largest entry in $\mathbf{r}^{(l)}$, which we have assumed is nonnegative) we have that $\mathbf{v}^{(l+1)} \geq 0$.

From (9) we have $\mathbf{r}^{(l+1)} = \mathbf{r}^{(l)} - m_l \mathbf{e}_k \otimes \mathbf{e}_i + \frac{m_l}{k} \mathbf{e}_{k+1} \otimes \mathbf{P}\mathbf{e}_i$, but we have assumed $\mathbf{P}$ is stochastic, so $\mathbf{e}_{k+1} \otimes \mathbf{P}\mathbf{e}_i \geq 0$. Then, note that $\mathbf{r}^{(l)} - m_l \mathbf{e}_k \otimes \mathbf{e}_i \geq 0$ because by the inductive hypothesis $\mathbf{r}^{(l)} \geq 0$ and subtracting m_l simply zeros out that entry of $\mathbf{r}^{(l)}$. Thus, $\mathbf{r}^{(l+1)} \geq 0$, as desired.

(Convergence) Because the residual is always nonnegative, we can use the identity $\|\mathbf{r}^{(l)}\|_1 = \mathbf{e}^T \mathbf{r}^{(l)}$. Left multiplying by $\mathbf{e}^T$ in (9) yields $\|\mathbf{r}^{(l+1)}\|_1 = \|\mathbf{r}^{(l)}\|_1 - m_l + \frac{m_l}{k} \left(\mathbf{e}^T \mathbf{e}_{k+1} \otimes \mathbf{e}^T \mathbf{P}\mathbf{e}_i\right)$. Since we've assumed $\mathbf{P}$ is column stochastic, $\mathbf{e}^T \mathbf{P}\mathbf{e}_i = 1$, and so this simplifies to $\|\mathbf{r}^{(l+1)}\|_1 = \|\mathbf{r}^{(l)}\|_1 - m_l + \frac{m_l}{k} = \|\mathbf{r}^{(l)}\|_1 - m_l \left(1 - \frac{1}{k}\right)$.

Since m_l is the largest entry in $\mathbf{r}^{(l)}$, we know it must be at least as big as the average value of an entry of $\mathbf{r}^{(l)}$. After l iterations, the residual can have no more than dl nonzero entries, because no more than d nonzeros can be added each iteration. Hence we have $m_l \geq \frac{\|\mathbf{r}^{(l)}\|_1}{dl}$. After the first iteration, we know $k \geq 2$

because the $k = 1$ block of $\mathbf{r}$ (denoted $\mathbf{r}_0$ in the notation of Section 3.2) is empty. If we put these bounds together, we have: $\|\mathbf{r}^{(l+1)}\|_1 \leq \|\mathbf{r}^{(l)}\|_1 - \frac{\|\mathbf{r}^{(l)}\|_1}{dl}\left(1 - \frac{1}{2}\right) = \|\mathbf{r}^{(l)}\|_1 \left(1 - \frac{1}{2dl}\right)$. Iterating this inequality yields the bound:

$$\|\mathbf{r}^{(l)}\|_1 \leq \prod_{j=1}^{l} (1 - \tfrac{1}{2dj}) \cdot \|\mathbf{r}^{(0)}\|_1, \tag{11}$$

and since $\mathbf{r}^{(0)} = \mathbf{e}_1 \otimes \mathbf{e}_c$ we have $\|\mathbf{r}^{(0)}\|_1 = 1$, proving the first inequality.

The second inequality of (10) follows from using the facts $(1 + x) \leq e^x$ (for $x > -1$) and $\log(l) < \sum_{j=1}^{l} \frac{1}{j}$ to write

$$\prod_{j=1}^{l} (1 - \tfrac{1}{2dj}) \leq \exp\{-\tfrac{1}{2d} \sum_{j=1}^{l} \tfrac{1}{j}\} \leq \exp\{-\tfrac{\log l}{2d}\} = l^{\left(-\frac{1}{2d}\right)}. \qquad \square$$

The inequality $(1 + x) \leq e^x$ follows from the Taylor series $e^x = 1 + x + o(x^2)$, and the lowerbound for the partial harmonic sum $\sum_{j=1}^{l} \frac{1}{j}$ follows from the left-hand rule integral approximation $\log(l) = \int_1^l \frac{1}{x} \mathrm{dx} \leq \sum_{j=1}^{l} \frac{1}{j}$.

5.2 Convergence of Error

Although the previous theorem establishes that GS will converge, we need a more precise statement about the error to bound the runtime. We will first state such a bound and use it to justify the claim of "nearly" sublinear runtime before formally proving it.

Theorem 2. *In the notation described above, the error of the approximation from l iterations of GS satisfies*

$$\|\hat{\mathbf{x}}^{(l)} - \mathbf{x}_{true}\|_1 \leq \tfrac{1}{N!N} + e \cdot l^{-\frac{1}{2d}} \quad \textit{where} \quad e = \exp(1). \tag{12}$$

Nearly Sublinear Runtime Given an input tolerance ε, Theorem 2 shows that the number of iterations l required to produce $\hat{\mathbf{x}}^{(l)}$ satisfying $\|\mathbf{x}_{\text{true}} - \hat{\mathbf{x}}^{(l)}\|_1 < \varepsilon$ depends on d alone (since we have control over N, and so can just choose N such that $\frac{1}{N!N} < \varepsilon/2$). For $\frac{1}{N!N} < \frac{\varepsilon}{2}$ to hold, it suffices to take $N = 2\log(1/\varepsilon)$ because $\frac{1}{N!N} < e^{-N}/2 = \frac{\varepsilon}{2}$ for $N > 5$. So $N = 2\log(1/\varepsilon)$.

Next, we need a value of l for which $e \cdot l^{-\frac{1}{2d}} < \frac{\varepsilon}{2}$ holds. Taking logs and exponentiating yields the bound

$$l > \exp\{2d(1 + \log(2) + \log(1/\varepsilon))\}. \tag{13}$$

If $d = C \log\log n$ for a constant C, then the desired error is guaranteed by

$$l > (\log n)^{2C(1+\log(2)+\log(1/\varepsilon))} \tag{14}$$

which grows sublinearly in n.

Proof of Theorem 2 By the triangle inequality $\|\mathbf{x}_{\text{true}} - \hat{\mathbf{x}}^{(l)}\|_1 \leq \|\mathbf{x}_{\text{true}} - \mathbf{x}_N\|_1 + \|\mathbf{x}_N - \hat{\mathbf{x}}^{(l)}\|_1$, so we can apply Lemma 1 to get $\|\mathbf{x}_{\text{true}} - \hat{\mathbf{x}}^{(l)}\|_1 \leq \frac{1}{N!N} + \|\mathbf{x}_N - \hat{\mathbf{x}}^{(l)}\|_1$, and then Lemma 2 to obtain

$$\|\mathbf{x}_{\text{true}} - \hat{\mathbf{x}}^{(l)}\|_1 \leq \tfrac{1}{N!N} + \|\mathbf{v} - \hat{\mathbf{v}}^{(l)}\|_1. \tag{15}$$

Theorem 1 gives the residual of the GS solution vector after l iterations, $\hat{\mathbf{v}}^{(l)}$, but we want the *error*, i.e. the difference between $\hat{\mathbf{v}}^{(l)}$ and $\mathbf{v}$. To obtain this quantity, we use a standard relationship between the residual and error specialized to our system: $\|\mathbf{v} - \hat{\mathbf{v}}^{(l)}\|_1 \leq \|\mathbf{M}^{-1}\|_1 \|\mathbf{r}^{(l)}\|_1$. To complete the proof of Theorem 2, it suffices, then, to show that $\|\mathbf{M}^{-1}\|_1 \leq e$ and use Theorem 1. The next lemma establishes this remaining bound. We suspect that the following result is already known in the literature and regret the tedious proof.

Lemma 3. *Matrices* $\mathbf{M}$ *of the form* $\mathbf{M} = \mathbf{I}_{N+1} \otimes \mathbf{I}_n - \mathbf{S}_{N+1} \otimes \mathbf{P}$ *for column-stochastic* $\mathbf{P}$ *satisfy* $\|\mathbf{M}^{-1}\|_1 \leq e$.

Proof. Write $\mathbf{M} = \mathbf{I} - \mathbf{S}_{N+1} \otimes \mathbf{P}$ and note that, since $\mathbf{S}_{N+1}^{N+1} = 0$ we have $(\mathbf{S}_{N+1} \otimes \mathbf{P})^{N+1} = 0$, i.e. $\mathbf{S}_{N+1} \otimes \mathbf{P}$ is nilpotent, so $\mathbf{M} = \mathbf{I} - \mathbf{S}_{N+1} \otimes \mathbf{P}$ has inverse

$$\mathbf{M}^{-1} = \mathbf{I} + (\mathbf{S}_{N+1} \otimes \mathbf{P}) + (\mathbf{S}_{N+1} \otimes \mathbf{P})^2 + \cdots + (\mathbf{S}_{N+1} \otimes \mathbf{P})^N. \tag{16}$$

To upperbound $\|\mathbf{M}^{-1}\|_1$ observe that each term $(\mathbf{S}_{N+1} \otimes \mathbf{P})^j$ is nonnegative, and so $\mathbf{M}^{-1}$ is itself nonnegative. Thus, $\|\mathbf{M}^{-1}\|_1$ is simply the largest column-sum of $\mathbf{M}^{-1}$, i.e. $\max_{k,i}\{\mathbf{e}^T \mathbf{M}^{-1} (\mathbf{e}_k \otimes \mathbf{e}_i)\}$. For convenience of notation, define $t_k = \mathbf{e}^T \mathbf{M}^{-1} (\mathbf{e}_k \otimes \mathbf{e}_i)$ (we will show that this value, t_k, is independent of i). Multiplying (16) by $\mathbf{e}^T$ and $(\mathbf{e}_k \otimes \mathbf{e}_i)$ and distributing produces

$$t_k = \left(\mathbf{e}^T (\mathbf{e}_k \otimes \mathbf{e}_i) + (\mathbf{e}^T \mathbf{S}_{N+1} \mathbf{e}_k) \otimes (\mathbf{e}^T \mathbf{P} \mathbf{e}_i) + \cdots + (\mathbf{e}^T \mathbf{S}_{N+1}^N \mathbf{e}_k) \otimes (\mathbf{e}^T \mathbf{P}^N \mathbf{e}_i)\right) \tag{17}$$

but since $\mathbf{e}^T \mathbf{P}^j \mathbf{e}_i = 1$ for all j, we end up with

$$t_k = \mathbf{e}^T \mathbf{S}_{N+1}^0 \mathbf{e}_k + \mathbf{e}^T \mathbf{S}_{N+1}^1 \mathbf{e}_k + \cdots + \mathbf{e}^T \mathbf{S}_{N+1}^N \mathbf{e}_k. \tag{18}$$

This justifies the notation t_k, since the value is seen to be independent of i here.

We set out to upperbound $\|\mathbf{M}^{-1}\|_1$, and we've now reduced the problem to simply bounding the t_k above. To do this, first note that $\mathbf{S}_{N+1} \mathbf{e}_k = \frac{1}{k} \mathbf{e}_{k+1}$ for all $1 \leq k \leq N$. Repeatedly left multiplying by $\mathbf{S}_{N+1}$ establishes that for $k+j > N+1$ we have $\mathbf{S}_{N+1}^j \mathbf{e}_k = 0$ if $j \geq 1$, but for $k+j \leq N$ we have $\mathbf{S}_{N+1}^j \mathbf{e}_k = \frac{(k-1)!}{(k-1+j)!} \mathbf{e}_{k+j}$ for $j = 0, \cdots, N-k+1$ and $k = 1, \cdots, N$. Using these we rewrite (18):

$$t_k = \sum_{j=0}^{N+1-k} \tfrac{(k-1)!}{(k-1+j)!}. \tag{19}$$

Observe that $t_1 = \sum_{j=0}^{N} \frac{1}{j!} \leq e$. The inequality $t_{k+1} \leq t_k$ implies that $t_k \leq t_1 < e$, and so to prove Lemma 3 it now suffices to prove this inequality. From (19) we have $t_k = \sum_{j=0}^{N-k} \frac{(k-1)!}{(k-1+j)!} + \frac{(k-1)!}{N!}$ and $t_{k+1} = \sum_{j=0}^{N-k} \frac{k!}{(k+j)!}$. The general terms satisfy $\frac{(k-1)!}{(k-1+j)!} \geq \frac{k!}{(k+j)!}$ because multiplying both sides by $\frac{(k-1+j)!}{(k-1)!}$ yields $1 \geq \frac{k}{k+j}$. Hence $t_k \geq t_{k+1}$, and so the lemma follows. □

5.3 Complexity

Each iteration requires updating the residual for each adjacent node, which in turn requires updating the residual heap. This step involves $O(d\log(ld))$ work to maintain the heap, since there are at most d entries added to $\mathbf{r}$, and each update of the heap requires $O(\log(ld))$ work, where ld is an upperbound on the size of the heap after l iterations. Since each vector add takes at most $O(d)$ operations, the total operation count for l iterations of the algorithm is $O(ld + ld\log(ld))$. If we do not have a sublinear number of steps l, then note that heap updates never take more than $\log(nN)$ work.

6 Experimental Results

We evaluate this method on a desktop computer with an Intel i7-990X, 3.47 GHz CPU and 24 GB of RAM. As described below, we implement two variations of our algorithm in C++ using the Matlab MEX interface. The graphs we use come from a variety of sources and range between 10^3 and 10^7 nodes and edges. All are undirected and connected. They include the dblp and flickr graphs [7], Newman's netscience, condmat-2003, and condmat-2005 graphs [18,19], and Arenas's pgp graph [6]. These results are representative of a larger collection. In the spirit of reproducible research, we make our code available. See the URL in the title.

6.1 Notes on Implementation

We do not yet have a fully efficient implementation of our algorithm. In particular, we maintain full solution vectors and full residual vectors that take $O(n)$ and $O(nN)$ work to initialize and store. In the future, we plan to use a hash table for these vectors. For the current scale of our graphs – $500,000$ nodes – we do not expect this change to make a large difference. We found that the runtime of Gauss-Southwell varied widely due to our use of the heap structure (see the TSGS line in Figure 3). To address this performance, we implemented an idea inspired by Andersen et al.'s replacement of a heap with a queue [3]. Rather than choose the largest element in the residual at each step, we pick an element from a queue that stores all residual elements larger than $\tau/(nN)$. Note that once all elements have been removed from the queue, the norm of the residual will be less than τ. We have not conducted an error analysis to determine how many steps are necessary to satisfy this condition, but empirically we find it performs similarly. For the accuracy results below, we use the method with the queue.

6.2 Accuracy and Runtime

While we know that our method converges as the number of steps runs to infinity, in Figure 2, we study the precision of approximate solutions. Recall that precision is the size of the intersection of the set of indices of the k largest entries from our approximation vector with the set of indices of the k largest entries from the

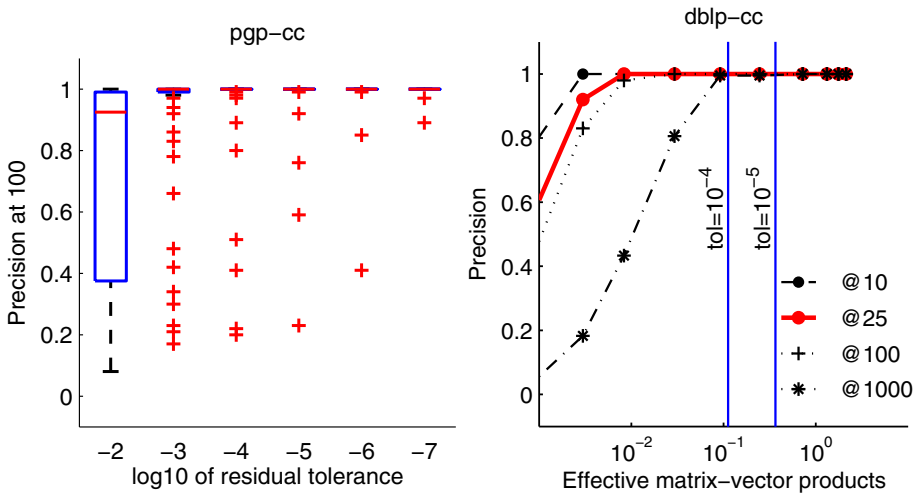

Fig. 2. At left, the precision of the top 100 for the largest entries in a column of the matrix exponential of the pgp graph (without the node's immediate neighbors). We show a boxplot summarizing the results over 50 trials. The "+" marks indicate outliers outside of the 75th percentile. At right, we vary the number of steps of the method and find we get high precision for the top-25 set on the large dblp graph (226,413 nodes) after 1% of the work involved in a matrix vector product. This second plot is indicative of results we see for other choices of the column as well. Solving to tolerance 10^{-5} takes 33% of the work of a matrix-vector product.

true solution, divided by k. In the vector $\exp\{\mathbf{P}\}\mathbf{e}_c$ the entry with index c and the entries corresponding to neighbors of node c are always large in both the true and the approximate solution vectors and so artificially inflate the scores. Because of this, we ignore those entries in both solution vectors and instead look at the indices of the next k largest entries (excluding node c and its neighbors).

We show results for the pgp network (10k nodes) with a boxplot representing 50 random columns. The plot suggests that a residual tolerance of 10^{-4} yields useful results in the majority of cases. We note that this figure represents the worst results we observed and many graphs showed no errors at low tolerances. Next, for a larger graph, we show how the precision in the top-k sets evolves for the dblp graph (226k nodes) as we perform additional steps. Based on both of these results, we suggest a residual tolerance of 10^{-5}.

Next, we compare the runtime of our method using a heap (TSGS) and using a queue (TSGSQ) to the runtime of three other methods solved to tolerance of 10^{-5}. We present the results for all 6 graphs. Each runtime is an average of 50 randomly chosen columns. Both EXPV and MEXPV are from ExpoKit [21] where MEXPV is a special routine for stochastic matrices. The Taylor method simply computes $\lceil 3\log_2(n) \rceil$ terms of the Taylor expansion. These results show that TSGSQ is an order of magnitude faster than the other algorithms and this performance difference persists over graphs spanning four orders of magnitude.

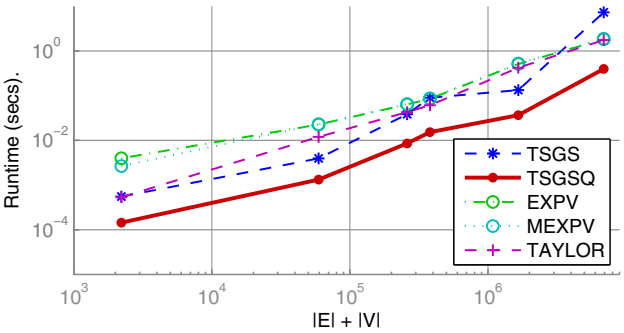

Fig. 3. We show the average runtime of our methods (TSGS, TSGSQ) for 50 columns of the matrix exponential selected at random compared with the runtime of methods from ExpoKit (EXPV and MEXPV) and a Taylor polynomial approximation. Our method with a queue is an order of magnitude faster.

7 Related Work and Discussion

Virtually all recent work on computing $\exp\{\mathbf{A}\}\mathbf{b}$ or even a single element of $\exp\{\mathbf{A}\}$ involves a Krylov or Lanczos approximation [21,4,20,1]. These methods all have runtimes that are $O(|E|)$, or worse, when the matrix comes from a graph. Given our strong assumption about the scaling of the maximum degree, it is possible that these algorithms would also enjoy sublinear runtimes and we are currently studying their analysis. We are also currently searching for additional work that may be related to our current local method to approximate the exponential.

As mentioned before, the doubly logarithmic bound on the maximum degree is unrealistic for social and information networks. We are currently working to improve the bound and believe that using a residual in a weighted ∞-norm, as used by Andersen et al. [3], may succeed. We also note that this method outperforms state of the art Krylov solvers on networks with nearly 10 million nodes and edges. Thus, we believe it to be useful independently of the runtime bound. We are currently working to extend the analysis to functions of scaled stochastic matrices, $\exp\{\alpha\mathbf{P}\}$ with $\alpha < 1$, and the adjacency matrix, $\exp\{\mathbf{A}\}$, for other link prediction methods.

References

1. Afanasjew, M., Eiermann, M., Ernst, O.G., Güttel, S.: Implementation of a restarted Krylov subspace method for the evaluation of matrix functions. Linear Algebra Appl. 429(10), 2293–2314 (2008)
2. Al-Mohy, A.H., Higham, N.J.: Computing the action of the matrix exponential, with an application to exponential integrators. SIAM J. Sci. Comput. 33(2), 488–511 (2011)

3. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using PageRank vectors. In: FOCS 2006 (2006)
4. Benzi, M., Boito, P.: Quadrature rule-based bounds for functions of adjacency matrices. Linear Algebra and its Applications 433(3), 637–652 (2010)
5. Berkhin, P.: Bookmark-coloring algorithm for personalized PageRank computing. Internet Mathematics 3(1), 41–62 (2007)
6. Boguñá, M., Pastor-Satorras, R., Díaz-Guilera, A., Arenas, A.: Models of social networks based on social distance attachment. Phys. Rev. E 70(5), 056122 (2004)
7. Bonchi, F., Esfandiar, P., Gleich, D.F., Greif, C., Lakshmanan, L.V.: Fast matrix computations for pairwise and columnwise commute times and Katz scores. Internet Mathematics 8(1-2), 73–112 (2012)
8. Chung, F.: The heat kernel as the PageRank of a graph. Proceedings of the National Academy of Sciences 104(50), 19735–19740 (2007)
9. Estrada, E.: Characterization of 3d molecular structure. Chemical Physics Letters 319(5-6), 713–718 (2000)
10. Estrada, E., Higham, D.J.: Network properties revealed through matrix functions. SIAM Review 52(4), 696–714 (2010)
11. Farahat, A., LoFaro, T., Miller, J.C., Rae, G., Ward, L.A.: Authority rankings from HITS, PageRank, and SALSA: Existence, uniqueness, and effect of initialization. SIAM Journal on Scientific Computing 27(4), 1181–1201 (2006)
12. Gallopoulos, E., Saad, Y.: Efficient solution of parabolic equations by Krylov approximation methods. SIAM J. Sci. Stat. Comput. 13(5), 1236–1264 (1992)
13. Hochbruck, M., Lubich, C.: On Krylov subspace approximations to the matrix exponential operator. SIAM J. Numer. Anal. 34(5), 1911–1925 (1997)
14. Kondor, R.I., Lafferty, J.D.: Diffusion kernels on graphs and other discrete input spaces. In: ICML 2002, pp. 315–322 (2002)
15. Kunegis, J., Lommatzsch, A.: Learning spectral graph transformations for link prediction. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, pp. 561–568. ACM, New York (2009)
16. Luo, Z.Q., Tseng, P.: On the convergence of the coordinate descent method for convex differentiable minimization. J. Optim. Theory Appl. 72(1), 7–35 (1992)
17. Moler, C., Van Loan, C.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM Review 45(1), 3–49 (2003)
18. Newman, M.E.J.: The structure of scientific collaboration networks. Proceedings of the National Academy of Sciences 98(2), 404–409 (2001)
19. Newman, M.E.J.: Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E 74(3), 036104 (2006)
20. Orecchia, L., Sachdeva, S., Vishnoi, N.K.: Approximating the exponential, the Lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator. In: STOC 2012, pp. 1141–1160 (2012)
21. Sidje, R.B.: ExpoKit: a software package for computing matrix exponentials. ACM Trans. Math. Softw. 24, 130–156 (1998)