A Nearly Sublinear Approximation to $\exp\{P\}e_i$ for Large Sparse Matrices from Social Networks

Kyle Kloster and David F. Gleich

Purdue University

December 14, 2013

Supported by NSF CAREER 1149756-CCF
Columns of the Matrix Exponential

exp{ A } used for link-prediction, node centrality, and clustering. Why?
Columns of the Matrix Exponential

\[ \exp(A) \] used for link-prediction, node centrality, and clustering. Why?

\[
\exp(A) = \sum_{k=0}^{\infty} \frac{1}{k!} A^k
\]

- \((A^k)_{ij}\) gives the number of length-\(k\) walks from \(i\) to \(j\), so...
- Large entries of \(\exp(A)\) denote “important” nodes / links
Columns of the Matrix Exponential

\( \exp(A) \) used for link-prediction, node centrality, and clustering. Why?

\[
\exp(A) = \sum_{k=0}^{\infty} \frac{1}{k!} A^k
\]

- \((A^k)_{ij}\) gives the number of length-\(k\) walks from \(i\) to \(j\), so...
- Large entries of \(\exp(A)\) denote “important” nodes / links
- \(\exp(A)\) is common, but other \(f(A)\) can be used: pagerank and heatkernel PR
- Assume column stochastic, \(P = GD^{-1}\) (more on this later)
Difficulties with current methods:
Sidje, TOMS 1998; Al-Mohy and Higham, SISC 2011

- Leading methods for $\exp\{A\}b$ use Krylov or Taylor methods:
  "basically" repeated mat-vecs

- "Small world" property: graph diameter $\leq 4 \Rightarrow$ repeated mat-vecs fill in rapidly (see picture)

- Not designed specifically for sparse networks.
Vectors $P^k e_i$ for $k = 1, 2, 3, 4$. $n = 1133$
Local Method

New method: avoid mat-vecs! → use a local method.

Local algorithms run in time proportional to size of output:

\[
\text{sparse solution vector} = \text{small runtime}
\]

Instead of matvecs, we do specially-selected vector adds using a relaxation method.
\( \exp \{ P \} e_i \) is a localized vector

x-axis: vector index, y-axis: magnitude of entry

the column of \( \exp \{ P \} \) produced by previous slide’s matvecs
Outline of Nexpokit method

1. Express $\exp\{A\}e_i$ via a Taylor polynomial
2. Form large linear system out of Taylor terms
3. Use sparse solver to approximate each terms’ largest entries
4. Combine approximated terms into a solution
In terms of Taylor terms

Taylor polynomial:

\[
\exp\{A\}e_i \approx \left( I + A + \frac{1}{2}A^2 + \frac{1}{3!}A^3 + \cdots + \frac{1}{N!}A^N \right) e_i
\]
Taylor polynomial:

\[
\exp\{A\}e_i \approx \left( I + A + \frac{1}{2} A^2 + \frac{1}{3!} A^3 + \cdots + \frac{1}{N!} A^N \right) e_i
\]

Compute terms recursively:

\[
v_k = \frac{1}{k!} A^k e_i = \frac{1}{k} A \left( \frac{1}{(k-1)!} A^{k-1} \right) e_i
\]

\[
v_k = \frac{1}{k} Av_{k-1}
\]
Our method

In terms of Taylor terms

Taylor polynomial:

$$\exp\{A\}e_i \approx \left( I + A + \frac{1}{2} A^2 + \frac{1}{3!} A^3 + \cdots + \frac{1}{N!} A^N \right) e_i$$

Compute terms recursively:

$$v_k = \frac{1}{k!} A^k e_i = \frac{1}{k} A \left( \frac{1}{(k-1)!} A^{k-1} \right) e_i$$

$$v_k = \frac{1}{k} A v_{k-1}$$

Then $$\exp\{A\}e_i \approx v_0 + v_1 + \cdots + v_{N-1} + v_N$$

(But we want to avoid computing $$v_j$$ in full...)
Our method

Forming a linear system

So we convert the Taylor polynomial into a linear system:
Our method

Forming a linear system

So we convert the Taylor polynomial into a linear system:

\[
\begin{bmatrix}
  I & & & \\
  -A/1 & I & & \\
  & -A/2 & \ddots & \\
  & & \ddots & I \\
  & & & -A/N & I
\end{bmatrix}
\begin{bmatrix}
  v_0 \\
  v_1 \\
  v_2 \\
  \vdots \\
  v_N
\end{bmatrix}
= \begin{bmatrix}
  e_i \\
  0 \\
  0 \\
  \vdots \\
  0
\end{bmatrix}
\]
Forming a linear system

So we convert the Taylor polynomial into a linear system:

\[
\begin{bmatrix}
I & -A/1 & I \\
-A/1 & I \\
-A/2 & \cdots \\
\vdots & \vdots & I \\
-A/N & I
\end{bmatrix}
\begin{bmatrix}
v_0 \\
v_1 \\
v_2 \\
\vdots \\
v_N
\end{bmatrix}
= \begin{bmatrix}
e_i \\
0 \\
0 \\
\vdots \\
0
\end{bmatrix}
\]

From
\[v_k = \frac{1}{k} Av_{k-1}\]
\[\exp(A)e_i \approx v_0 + v_1 + \cdots + v_{N-1} + v_N\]
(never formed explicitly)
Sparse solver: Gauss Southwell

Basic idea of Gauss Southwell (GS): solving $Mx = b$ when $x$ is “effectively sparse” (i.e. a localized vector)
Sparse solver: Gauss Southwell

Basic idea of Gauss Southwell (GS): solving $Mx = b$ when $x$ is “effectively sparse” (i.e. a localized vector)

1. Set $x^0 = 0$, $r^0 = b$, then iterate:
Our method

Sparse solver: Gauss Southwell

Basic idea of Gauss Southwell (GS): solving $Mx = b$ when $x$ is “effectively sparse” (i.e. a localized vector)

1. Set $x^0 = 0$, $r^0 = b$, then iterate:
2. At step $k$, relax largest entry of $r^k$ (denoted $r^k_i$), add to $x^k$;
   \[ x^{k+1} = x^k + r^k_i \cdot e_i \]
Sparse solver: Gauss Southwell

Basic idea of Gauss Southwell (GS): solving $Mx = b$ when $x$ is “effectively sparse” (i.e. a localized vector)

1. Set $x^0 = 0$, $r^0 = b$, then iterate:
2. At step $k$, relax largest entry of $r^k$ (denoted $r^k_i$), add to $x^k$;
   \[ x^{k+1} = x^k + r^k_i \cdot e_i \]
3. Add corresponding column of $M$ to residual:
   \[ r^{k+1} = (r^k - r^k_i \cdot e_i) + r^k_i \cdot M(:, i) \]
Our method

NEXPOKIT

Apply GS to our linear system, $M\tilde{v} = \tilde{e}_i$:

$$
\begin{bmatrix}
    I \\
    -A/1 & I \\
    -A/2 & \ddots \\
    \vdots & \ddots & I \\
    -A/N & I \\
\end{bmatrix}
\begin{bmatrix}
    v_0 \\
    v_1 \\
    v_2 \\
    \vdots \\
    v_N \\
\end{bmatrix} =
\begin{bmatrix}
    e_i \\
    0 \\
    0 \\
    \vdots \\
    0 \\
\end{bmatrix}
$$

Now residual has index and block section: $r(i, j)$. Iteration reduces to:
(1) adding $r(i, j)^k$ to a single entry of $x$, our approximation;
(2) adding scaled column, $\frac{r(i,j)^k}{j}A(:, i)$, to section $j$ of the residual.
Convergence and Implementation

Scaling \( r(i,j)^k A(:, i) \) guarantees \( \|r^k\|_1 \) decreases, for column stochastic \( A \):
Scaling $r^{(i,j)^k} / j \mathbf{A}(\cdot, i)$ guarantees $\|r^k\|_1$ decreases, for column stochastic $\mathbf{A}$:

$$\|r^{k+1}\|_1 = \|r^k\|_1 - r(i,j)^k + r^{(i,j)^k} = \|r^k\|_1 - r(i,j)^k (1 - \frac{1}{j})$$

Largest entry, $r(i,j)$ is bounded below by average, $r(i,j) > \|r\|_1 / (\# \text{ non zeros in } r)$. 
Scaling $\frac{r(i,j)^k}{j}A(:, i)$ guarantees $\|r^k\|_1$ decreases, for column stochastic $A$:

$$\|r^{k+1}\|_1 = \|r^k\|_1 - r(i, j)^k + \frac{r(i,j)^k}{j} = \|r^k\|_1 - r(i, j)^k (1 - \frac{1}{j})$$

Largest entry, $r(i, j)$ is bounded below by average, $r(i, j) > \|r\|_1/(#)\text{ non zeros in } r$).

No component of large linear system formed explicitly:
- residual vector stored in a heap (alternative: queue with threshold)
- matrix $M$ not formed at all
- blocks $v_j$ not stored separately
Converges for stochastic matrices

“Nearly sublinear” – if $d_{\text{max}} = O(\log\log n)$ (unrealistic)

In practice, sublinear if NNZ = $O(n)$

Less work than a single mat-vec
Converges for stochastic matrices

“Nearly sublinear” – if $d_{\text{max}} = O(\log \log n)$ (unrealistic)

In practice, sublinear if $\text{NNZ} = O(n)$

Less work than a single mat-vec

New: for power-law degree distributed networks, the runtime for an error of $\varepsilon$ is

$$\log \left( \frac{1}{\varepsilon} \right) \left( \frac{1}{\varepsilon} \right)^{3/2} d_{\text{max}} \log(d_{\text{max}})^2$$

Social networks tend to have $d_{\text{max}} = O(n^r)$ for $r < 1$, so this is sublinear in $n$. 
Power-law degree distribution

Intuition for Proof

Our sublinear runtime proof depends on the degree distribution:

- decrease in $\|r\|$ depends on largest value in $r$, $r_i$
- lowerbound $r_i$ using the average value of $r$
- average value $= \|r\|/$(# of nonzeros in $r$)

($\#$ of nonzeros in $r$) upper bounded by $d_{max} \times (#iterations)$
Intuition for Proof

Our sublinear runtime proof depends on the degree distribution:

- decrease in $\|r\|$ depends on largest value in $r$, $r_i$
- lowerbound $r_i$ using the average value of $r$
- average value = $\|r\|/(\# \text{ of nonzeros in } r)$

($\# \text{ of nonzeros in } r$) upper bounded by $d_{\text{max}} \times (\# \text{iterations})$

Power-law network: $\# \text{ of nonzeros in } r$ after $t$ iterations grows like $O(t)$ instead of $d_{\text{max}} \times t$. $\Rightarrow$ average value can’t decay too fast. Hence, $\|r\|$ is guaranteed to decrease ”fast enough”: $\|r^t\| < O(t^{-2/3})$
“GSQ” is a version of our Gauss-Southwell method that stores the residual vector in a queue instead of a heap.
Results and Future Work

Runtime on larger networks

Results and Future Work

Runtime on larger networks

For webbase-2001, $n = 118, 142, 155$, ave degree = 8.6.
For pgp–cc $n = 10,680$, but this is representative of dataset.
For dblp-cc, $n = 226,413$. Again, this is representative.
Future Work

- Adapt the method to other functions: \( \cosh(x), x^{\frac{1}{p}}, \log(x) \).
- Allow for scaling, \( f(tA)e_i \).
- Allow for \( f(A) \) times a vector \( v \) (other than \( e_i \)).
- Improve domain of convergence, \( \rho(A) \in (0, 1] \).
Code and Further Details

Code available at

http://www.cs.purdue.edu/homes/dgleich/codes/nexpokit

For details and references, see our paper at