A sub-linear method for computing columns of functions of sparse matrices

Kyle Kloster and David F. Gleich

Purdue University

March 3, 2014

Supported by NSF CAREER 1149756-CCF

Overview

1. $f(\mathbf{A})$: problem description and applications
2. Description of “sub-linear” results
3. The Algorithm for $f(\mathbf{A})\mathbf{b}$
4. Intuition for proof
5. Experiments on real-world social networks

Functions of Matrices: background

We can apply most functions, e.g. $f(x) = cos(x)$, to any square matrices $\mathbf{A}$ if f is defined on the eigenvalues of $\mathbf{A}$. One definition: Taylor series!

$$cos(x) = \frac{1}{0!} + \frac{-x^2}{2!} + \frac{x^4}{4!} + \cdots$$

$$cos(\mathbf{A}) = \frac{\mathbf{I}}{0!} + \frac{-\mathbf{A}^2}{2!} + \frac{\mathbf{A}^4}{4!} + \cdots$$

Then we can think of $f(\mathbf{A})\mathbf{b}$ as the **action** of the operator $f(\mathbf{A})$ on $\mathbf{b}$, or as a **diffusion** on a graph underlying the matrix $\mathbf{A}$.

Functions of Matrices: applications

Action:

$f(x) = e^x$: $\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}$; $\mathbf{x}(0) = \mathbf{x}_0$
solution: $\mathbf{x}(t) = \exp\{t\mathbf{A}\}\mathbf{x}_0$

$f(x) = x^{1/p}$: $\mathbf{P}(t)$ transition matrix for Markov process
$\mathbf{P}(1)$ describes process over a year; $\mathbf{P}^{1/12}$ for a month

Functions of Matrices: applications

Action:

$f(x) = e^x$: $\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}$; $\mathbf{x}(0) = \mathbf{x}_0$
solution: $\mathbf{x}(t) = \exp\{t\mathbf{A}\}\mathbf{x}_0$

$f(x) = x^{1/p}$: $\mathbf{P}(t)$ transition matrix for Markov process
$\mathbf{P}(1)$ describes process over a year; $\mathbf{P}^{1/12}$ for a month

Diffusion:

$f(x) = (1 - \alpha x)^{-1}$: the **resolvent** yields the PageRank diffusion:
$f(\mathbf{P})\mathbf{e}_i$ interpreted as nodes' importance to node i.

$f(x) = e^{tx}$: $e^{t\mathbf{P}}\mathbf{e}_i$, the **heat kernel** diffusion, offers
an alternative ranking of nodes' importance

Parameters of $f(\mathbf{A})\mathbf{b}$

$\mathbf{A}$:

- Original motivation: $\mathbf{A} =$ a normalized version of an adjacency matrix from a social network; the Laplacian or random-walk matrix. Sparse, small diameter, stochastic, degree distribution follows power-law
- Generalized: any nonnegative $\mathbf{A}$ with $\|\mathbf{A}\|_1 \leq 1$.

$\mathbf{b}$:

- Originally $\mathbf{b} = \mathbf{e}_i$, i.e. compute a column $f(\mathbf{A})\mathbf{e}_i$
- Generalized: $\mathbf{b}$ can be any sparse, stochastic vector

$f(\cdot)$:

- Originally $f(x) = e^x, (1 - \alpha x)^{-1}$
- Generalized: can be any function decaying "fast enough"

Columns of the Matrix Exponential

$\exp\{\ \mathbf{A}\ \}$ used for link-prediction, node centrality, and clustering. Why?

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

- $(\mathbf{A}^k)_{ij}$ gives the number of length-k walks from i to j, so...
- Large entries of $\exp\{\mathbf{A}\}$ denote "important" nodes / links
- Used for link-prediction, node ranking, clustering

Columns of the Matrix Exponential

exp{ **A** } used for link-prediction, node centrality, and clustering. Why?

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

- $(\mathbf{A}^k)_{ij}$ gives the number of length-k walks from i to j, so...
- Large entries of $\exp\{\mathbf{A}\}$ denote "important" nodes / links
- Used for link-prediction, node ranking, clustering
- $\exp\{\mathbf{A}\}$ is common, but other $f(\mathbf{A})$ can be used:
- PageRank can be defined from the resolvent:

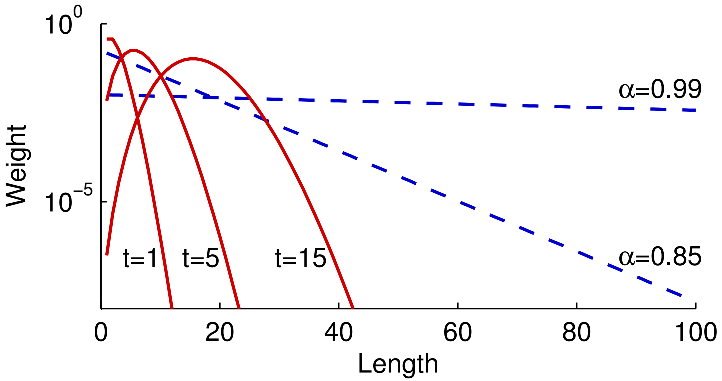
$$(\mathbf{I} - \alpha\mathbf{A})^{-1} = \sum_{k=0}^{\infty} \alpha^k \mathbf{A}^k$$

$\rightarrow$ replace $\frac{1}{k!}$ with other coefficients?

$f(\mathbf{A})$ as weighted sum of walks

For $f(\mathbf{A}) = e^{t\mathbf{A}}$ and $f(\mathbf{A}) = (1 - \alpha\mathbf{A})^{-1}$, how are walks weighted?

$$f(\mathbf{A})\mathbf{b} = \left(f_0\mathbf{I} + f_1\mathbf{A} + f_2\mathbf{A}^2 + f_3\mathbf{A}^3 + \cdots\right)\mathbf{b}$$

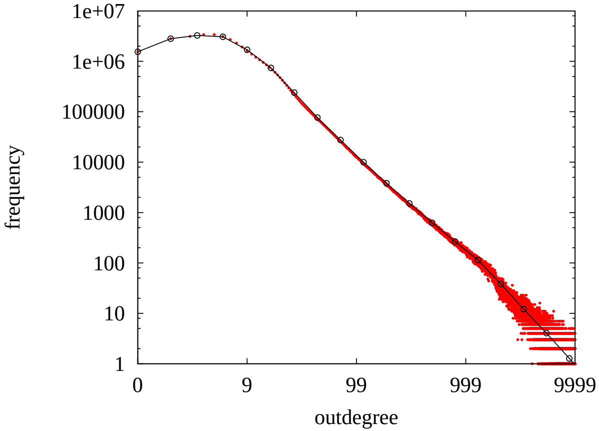


Big Graphs from Social Networks

We've seen the computation (f); what does the domain of inputs look like?

- Social networks like Twitter, YouTube, Friendster, Livejournal
- Large: $n = 10^6, 10^7, 10^9+$
- Sparse: $|E| = O(n)$, often $\leq 50n$
- Difficulty: "small world" property: diameter ≈ 4 (!)
- Helpful: Power-law degree distribution (picture)

Power-law degree distribution

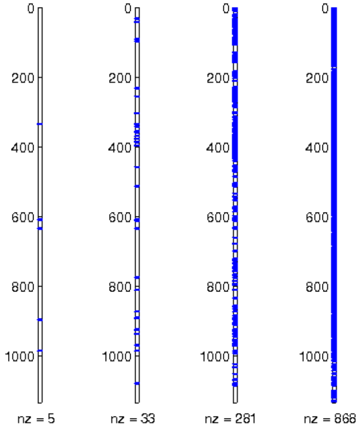


[Laboratory for Web Algorithms, http://law.di.unimi.it/index.php]

Difficulties with current methods: Sidje, TOMS 1998; Al-Mohy and Higham, SISC 2011

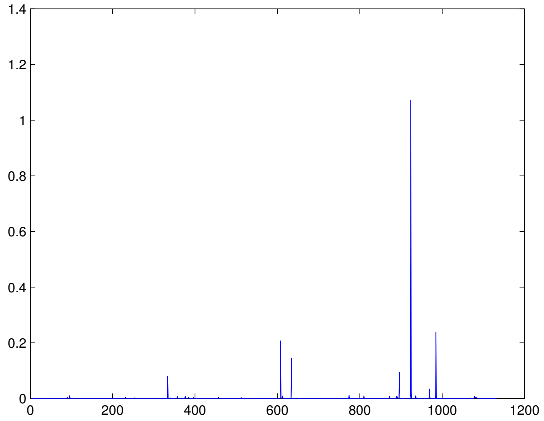
- Leading methods for $f(\mathbf{A})\mathbf{b}$ use Krylov or Taylor methods: "basically" repeated mat-vecs
- "Small world" property: graph diameter $\leq 4 \Rightarrow$ repeated mat-vecs fill in rapidly (see picture)
- Not designed specifically for sparse networks.

Fill-in from repeated matvecs



Vectors $\mathbf{P}^k \mathbf{e}_i$ for $k = 1, 2, 3, 4$. $n = 1133$

$f(\mathbf{P})\mathbf{e}_i$ is a localized vector



x-axis: vector index, y-axis: magnitude of entry
the column of $\exp\{\mathbf{P}\}$ produced by previous slide's matvecs

Local Method

New method: avoid mat-vecs! $\rightarrow$ 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.

Main Result 1

Theorem 1:[action of f on **b**]
Given nonnegative **A** satisfying $\|\mathbf{A}\|_1 \leq 1$, with power-law degree distribution and max degree d; and sparse stochastic **b**; our method computes $\mathbf{x} \approx f(\mathbf{A})\mathbf{b}$ such that

$$\|f(\mathbf{A})\mathbf{b} - \mathbf{x}\|_1 < \varepsilon \text{ in work } (\varepsilon) = O\left((1/\varepsilon)^{C_f} \log(1/\varepsilon) d^2 \log(d)^2\right),$$

"work" "scales as" $d^2\log(d)^2$ in the graph size

for any function f that decays "fast enough". The constant C_f depends on how quickly the Taylor coefficients of f decay.

Main Result 1

Theorem 1:[action of f on $\mathbf{b}$]
Given nonnegative $\mathbf{A}$ satisfying $\|\mathbf{A}\|_1 \leq 1$, with power-law degree distribution and max degree d; and sparse stochastic $\mathbf{b}$; our method computes $\mathbf{x} \approx f(\mathbf{A})\mathbf{b}$ such that

$$\|f(\mathbf{A})\mathbf{b} - \mathbf{x}\|_1 < \varepsilon \text{ in work } (\varepsilon) = O\left((1/\varepsilon)^{C_f} \log(1/\varepsilon) d^2 \log(d)^2\right),$$

"work" "scales as" $d^2 \log(d)^2$ in the graph size

for any function f that decays "fast enough". The constant C_f depends on how quickly the Taylor coefficients of f decay.

For $f(x) = (1 - \alpha x)^{-1}$, $C_f = \frac{1}{1-\alpha}$ (Note: $\alpha \in (0, 1)$).
For $f(x) = e^x$, $C_f = \frac{3}{2}$
For $f(x) = x^{1/p}$, $C_f = \frac{3p}{5p-1}$ (Note: $p \in (0, 1)$).

Main Result 2

Theorem 2:[diffusion of f across a graph]
Given column stochastic $\mathbf{A}$ and $\mathbf{b}$, $\tilde{\mathbf{x}} \approx \tilde{f}(t\mathbf{A})\mathbf{b}$ can be computed such that

$$\|\tilde{f}(\mathbf{P})\mathbf{b} - \tilde{\mathbf{x}}\|_\infty < \varepsilon \text{ in work } (\varepsilon) = O\left(\frac{2f(t)}{\varepsilon}\right),$$

(Remark: the 'tilde' denotes a degree-normalized version for the diffusion: $\mathbf{D}^{-1}\exp\{t\mathbf{P}\}\mathbf{b}$, for example. We normalize by degrees to adjust for the influence of the stationary distribution of $\mathbf{P}$.)

Corollary: $f(\mathbf{A})\mathbf{b}$ is a local vector.

Proof: Because sublinear work is done, $f(\mathbf{A})\mathbf{b}$ cannot have $O(n)$ nonzeros.

Overview

Outline of Nexpokit method (our second method, hk-relax, is related)

1. Express $f(\mathbf{A})\mathbf{b}$ via a Taylor polynomial
2. Form large linear system out of Taylor terms
3. Use sparse solver to approximate each term's largest entries
4. Combine approximated terms into a solution

In terms of Taylor terms

Taylor polynomial:

$$f(\mathbf{A})\mathbf{b} \approx \left(f_0\mathbf{I} + f_1\mathbf{A} + f_2\mathbf{A}^2 + f_3\mathbf{A}^3 + \cdots + f_N\mathbf{A}^N \right) \mathbf{b}$$

Compute terms recursively: $\mathbf{v}_k = f_k\mathbf{A}^k\mathbf{e}_i = \frac{f_k}{f_{k-1}}\mathbf{A}\left(f_{k-1}\mathbf{A}^{k-1}\right)\mathbf{e}_i$

$$\mathbf{v}_k = \frac{f_k}{f_{k-1}}\mathbf{A}\mathbf{v}_{k-1}$$

Then $f(\mathbf{A})\mathbf{b} \approx \mathbf{v}_0 + \mathbf{v}_1 + \cdots + \mathbf{v}_{N-1} + \mathbf{v}_N$
(But we want to avoid computing $\mathbf{v}_j$ in full...)

Forming a linear system

So we convert the Taylor polynomial into a linear system. For simplicity's sake, we use the example of $\exp\{\mathbf{A}\}\mathbf{e}_i$ here.

Forming a linear system

So we convert the Taylor polynomial into a linear system. For simplicity's sake, we use the example of $\exp\{\mathbf{A}\}\mathbf{e}_i$ here.

$$\begin{bmatrix} \mathbf{I} & & & & \\ -\mathbf{A}/1 & \mathbf{I} & & & \\ & -\mathbf{A}/2 & \ddots & & \\ & & \ddots & \mathbf{I} & \\ & & & -\mathbf{A}/N & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_N \end{bmatrix} = \begin{bmatrix} \mathbf{e}_i \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where we use the identity $\mathbf{v}_k = \frac{1}{k}\mathbf{A}\mathbf{v}_{k-1}$ (which comes from $\mathbf{v}_k = \frac{f_k}{f_{k-1}}\mathbf{A}\mathbf{v}_{k-1}$, since $f_k = \frac{1}{k!}$, so $f_k/f_{k-1} = \frac{(k-1)!}{k!} = \frac{1}{k}$).
Then $\exp\{\mathbf{A}\}\mathbf{e}_i \approx \mathbf{v}_0 + \mathbf{v}_1 + \cdots + \mathbf{v}_{N-1} + \mathbf{v}_N$

Sparse solver: Gauss Southwell

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

1. Set $\mathbf{x}^{(0)} = 0$, $\mathbf{r}^{(0)} = \mathbf{b}$, then iterate:
2. At step k, relax maximal entry of $\mathbf{r}^{(k)}$ (denoted $m^{(k)}$), add to $\mathbf{x}^{(k)}$;

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + m^{(k)} \cdot \mathbf{e}_i$$

3. Add corresponding column of $\mathbf{M}$ to residual:

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - m^{(k)} \cdot \mathbf{M}(:, i)$$

NEXPOKIT

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

$$\begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_N \end{bmatrix} = \begin{bmatrix} \mathbf{e}_i \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{I} & & & & \\ -\mathbf{A}/1 & \mathbf{I} & & & \\ & -\mathbf{A}/2 & \ddots & & \\ & & \ddots & \mathbf{I} & \\ & & & -\mathbf{A}/N & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_N \end{bmatrix}$$

The update can be simplified to a block-wise update:

$$\mathbf{r}^{(k+1)} = (\mathbf{r}^{(k)} - m^{(k)} \cdot \mathbf{e}_j \otimes \mathbf{e}_i) + \frac{m^{(k)}}{j+1} \cdot \mathbf{A}(:, i) \qquad (1)$$

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 $\mathbf{v}_j$ not stored separately, stored as one solution vector $\mathbf{x} = \sum \mathbf{v}_j$.

Outline of proof

Initial residual is $\mathbf{r} = \mathbf{e}_i$, has $\|\mathbf{r}^{(0)}\|_1 = 1$, and it decreases at each step. We show that

1. decay of $\|\mathbf{r}^{(k)}\|_1$ depends on its max value $m^{(k)}$
2. max value $m^{(k)}$ is bounded below by average value of $\mathbf{r}$
3. average value of $\mathbf{r}$ depends on # nonzeros in $\mathbf{r}$
4. growth of #nnz($\mathbf{r}$) depends on degree distribution
5. Power-law degree distribution implies $\#\mathrm{nnz}(\mathbf{r})$ grows slowly, so
6. $\|\mathbf{r}\|_1 \to 0$ at a certain minimum speed!

Decay of $\|\mathbf{r}\|_1$

Residual $\mathbf{r} = [\mathbf{r}_0; \mathbf{r}_1; \cdots ; \mathbf{r}_N]$ has index and block section: $r(i,j)$. For our special linear system, the GS residual reduces to: during step k, do

(1) delete $r(i,j)^{(k)}$ in $\mathbf{r}$ and add it to $\mathbf{x}_i$, our approximation;
(2) add scaled column, $\frac{m^{(k)}}{j}\mathbf{A}(:,i)$, to section j of the residual.

Taking the 1-norm of (1) shows

$$\|\mathbf{r}^{(k+1)}\|_1 \leq \|\mathbf{r}^{(k)}\|_1 - m^{(k)}(1 - \tfrac{1}{j})$$

Note the $(1 - \frac{1}{j})$ factor appears because we're looking specifically at e^x. For the resolvent, $f(x) = (1 - \alpha x)^{-1}$, this factor would be $(1 - \alpha)$ instead.

Number of nonzeros

Largest entry, $m^{(k)} = r(i,j)$ is bounded below by average value of the residual,

$$m^{(k)} = r(i,j) > \|\mathbf{r}\|_1 / (\#\text{ non zeros in } \mathbf{r})$$

But we can bound $\mathrm{nnz}(\mathbf{r}) := (\#\text{ of nonzeros in } \mathbf{r})$ based on the degree of the column of **A** that we add to the residual each step.

Number of nonzeros

Largest entry, $m^{(k)} = r(i,j)$ is bounded below by average value of the residual,

$$m^{(k)} = r(i,j) > \|\mathbf{r}\|_1/(\# \text{ non zeros in } \mathbf{r})$$

But we can bound $\mathrm{nnz}(\mathbf{r}) := (\#$ of nonzeros in $\mathbf{r})$ based on the degree of the column of **A** that we add to the residual each step.

Each iteration we can add no more nonzeros to **r** than the largest degree among all unvisited nodes.

Usually the best we can say is that this is upper bounded by $d := d_{\text{max}} * (\# iterations)$, because it's possible every node has max degree.

But with the power-law assumption ...

Number of nonzeros

Largest entry, $m^{(k)} = r(i,j)$ is bounded below by average value of the residual,

$$m^{(k)} = r(i,j) > \|\mathbf{r}\|_1 / (\#\text{ non zeros in } \mathbf{r})$$

But we can bound $\mathrm{nnz}(\mathbf{r}) := (\#\text{ of nonzeros in } \mathbf{r})$ based on the degree of the column of **A** that we add to the residual each step.

Each iteration we can add no more nonzeros to **r** than the largest degree among all unvisited nodes.

Usually the best we can say is that this is upper bounded by $d := d_{\text{max}} * (\#\mathit{iterations})$, because it's possible every node has max degree.

But with the power-law assumption ...

Power-law degree distribution

With power-law assumption, we know that the t^{th} largest degree, d_t, is bounded by $d_t \leq Cd \cdot t^{-\beta}$ for some β near 1 and some constant C.

After k iterations, $\mathrm{nnz}(\mathbf{r})$ is bounded by the sum of the degrees of the new vertices visited in those k iterations. By step k, this is at most $\mathrm{nnz}(\mathbf{r}) \leq \sum_{t=1}^{k} d_t$, so

$$\mathrm{nnz}(\mathbf{r}) \leq \sum_{t=1}^{k} d_t \leq \sum_{t=1}^{k} Cd \cdot t^{-1}$$

In fact, after the first d iterations, d_t is just a small constant, c. Then this sum grows no faster than $\sum_{t=1}^{k} d \cdot t^{-1} \leq d\log(d) + c \cdot t$. So $\mathrm{nnz}(\mathbf{r})$grows like $t \cdot c$ for $c \approx 1$ instead of $t \cdot d$ (!).

Convergence

We had

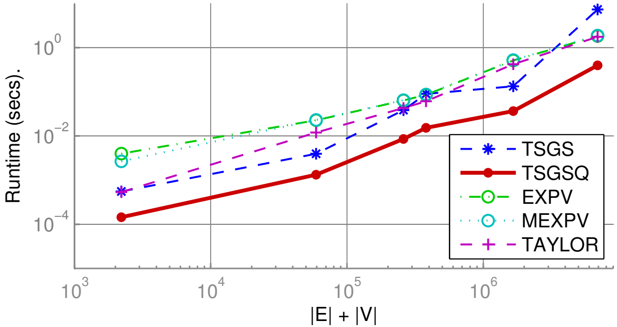
$$\|\mathbf{r}^{(k+1)}\|_1 \leq \|\mathbf{r}^{(k)}\|_1 - m^{(k)}(1 - \tfrac{1}{j})$$

The power-law assumption allows the bound $-m^{(k)} \leq -\frac{\|\mathbf{r}^{(k)}\|_1}{C_2 + c \cdot k}$.

$$\begin{aligned}
\|\mathbf{r}^{(k+1)}\|_1 &\leq \|\mathbf{r}^{(k)}\|_1 \left(1 - \frac{2/3}{C_2 + c \cdot k}\right) \\
&\leq \|\mathbf{r}^{(k)}\|_1 \exp\{-\tfrac{2}{3}\tfrac{1}{C_2 + c \cdot k}\} \\
&\leq \|\mathbf{r}^{(0)}\|_1 \exp\{-\tfrac{2}{3}\sum_{t=0}^{k} \tfrac{1}{C_2 + c \cdot t}\} \\
&\leq \exp\{-\tfrac{2}{3}\log(k + C)\} \\
\|\mathbf{r}^{(k+1)}\|_1 &\leq (k + C)^{-2/3}
\end{aligned}$$

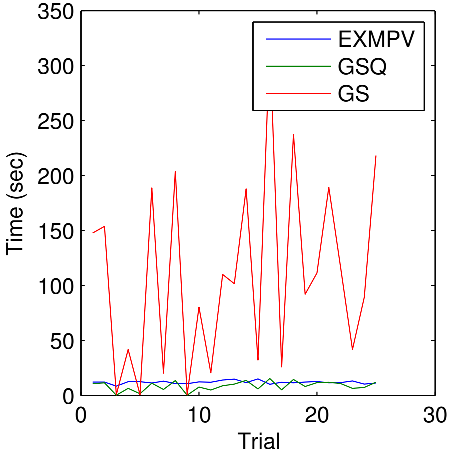
(See the paper cited at the end for a precise completion of the proof).

Runtime v. Graph Size



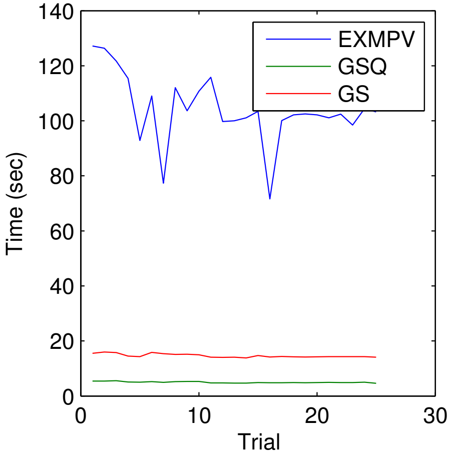
“GSQ” is a version of our Gauss-Southwell method that stores the residual vector in a queue instead of a heap.

Runtime on larger networks



For `ljournal-2008`, $n = 5,363,260$, ave degree $= 14.7$.

Runtime on larger networks



For `webbase-2001`, $n = 118,142,155$, ave degree $= 8.6$.

Code and Further Details

Code available at

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

For details and references, see our paper at

`http://arxiv.org/abs/1310.3423`