Solution Methods
for the
PageRank Problem

Amy Langville
Carl Meyer

Department of Mathematics
North Carolina State University
Raleigh, NC

College of Charleston 2/27/04

Outline

- Introduction to Google and PageRank
- Markov Chain methods for computation of PageRank
- Linear System methods for computation of PageRank

Google

Indexing

- Must index key terms on each page
  Robots crawl the web — software does indexing

- Inverted file structure (like book index: terms $\longrightarrow$ to pages)

$$\begin{aligned} Term_1 &\rightarrow P_i,\ P_j, \ldots \\ Term_2 &\rightarrow P_k,\ P_l, \ldots \\ &\ \vdots \end{aligned}$$

Ranking

- Determine a "PageRank" for each page $P_i, P_j, P_k, P_l, \ldots$
  Query independent — Based only on link structure

- Query matching
  $Q = Term_1, Term_2, \ldots$ produces $P_i, P_j, P_k, P_l, \ldots$

- Return $P_i, P_j, P_k, P_l, \ldots$ to user in order of PageRank

Google's PageRank Idea

(Sergey Brin & Lawrence Page 1998)

- Rankings are not query dependent

  Depend only on link structure

  Off-line calculations

- Your page P has some rank $r(P)$

- Adjust $r(P)$ higher or lower depending on ranks of pages that point to P

- Importance is not number of in-links or out-links

  One link to P from Yahoo! is important

  Many links to P from me is not

- Yahoo! points many places — value of link to P is diluted

PageRank

The Definition

$$r(P) = \sum_{P \in \mathcal{B}_P} \frac{r(P)}{|P|}$$

$\mathcal{B}_P$ = {all pages pointing to P}

$|P|$ = number of out links from P

Successive Refinement

Start with $r_0(P_i) = 1/n$ for all pages $P_1, P_2, \ldots, P_n$

Iteratively refine rankings for each page

$$r_1(P_i) = \sum_{P \in \mathcal{B}_{P_i}} \frac{r_0(P)}{|P|}$$

$$r_2(P_i) = \sum_{P \in \mathcal{B}_{P_i}} \frac{r_1(P)}{|P|}$$

$$\ddots$$

$$r_{j+1}(P_i) = \sum_{P \in \mathcal{B}_{P_i}} \frac{r_j(P)}{|P|}$$

In Matrix Notation

After Step j

$$\boldsymbol{\pi}_j^T = \left[r_j(P_1),\, r_j(P_2),\, \cdots,\, r_j(P_n)\right]$$

$$\boldsymbol{\pi}_{j+1}^T = \boldsymbol{\pi}_j^T \mathbf{P} \quad \text{where} \quad p_{ij} = \begin{cases} 1/|P_i| & \text{if } i \rightarrow j \\ 0 & \text{otherwise} \end{cases}$$

$$\text{PageRank} = \lim_{j\rightarrow\infty} \boldsymbol{\pi}_j^T = \boldsymbol{\pi}^T$$

(provided limit exists)

It’s Almost a Markov Chain

P has row sums = 1 for ND nodes, row sums = 0 for D nodes

In Matrix Notation

After Step j

$$\boldsymbol{\pi}_j^T = \left[r_j(P_1),\ r_j(P_2),\ \cdots,\ r_j(P_n)\right]$$

$$\boldsymbol{\pi}_{j+1}^T = \boldsymbol{\pi}_j^T \mathbf{P} \quad \text{where} \quad p_{ij} = \begin{cases} 1/|P_i| & \text{if } i \rightarrow j \\ 0 & \text{otherwise} \end{cases}$$

$$\text{PageRank} = \lim_{j\rightarrow\infty} \boldsymbol{\pi}_j^T = \boldsymbol{\pi}^T$$

(provided limit exists)

It's Almost a Markov Chain

P has row sums = 1 for ND nodes, row sums = 0 for D nodes

Stochasticity Fix: $\bar{\mathbf{P}} = \mathbf{P} + \mathbf{a}\mathbf{v}^T$.

(a_i=1 for $i \in D$, 0, o.w.)

In Matrix Notation

After Step j

$$\boldsymbol{\pi}_j^T = \left[r_j(P_1),\ r_j(P_2),\ \cdots,\ r_j(P_n) \right]$$

$$\boldsymbol{\pi}_{j+1}^T = \boldsymbol{\pi}_j^T \mathbf{P} \quad \text{where} \quad p_{ij} = \begin{cases} 1/|P_i| & \text{if } i \rightarrow j \\ 0 & \text{otherwise} \end{cases}$$

$$\text{PageRank} = \lim_{j\rightarrow\infty} \boldsymbol{\pi}_j^T = \boldsymbol{\pi}^T$$

(provided limit exists)

It's Almost a Markov Chain

P has row sums = 1 for ND nodes, row sums = 0 for D nodes

Stochasticity Fix: $\bar{\mathbf{P}} = \mathbf{P} + \mathbf{a}\mathbf{v}^T$. ($a_i$=1 for $i \in D$, 0, o.w.)

Each $\boldsymbol{\pi}_j^T$ is a probability distribution vector $\left(\sum_i r_j(P_i) = 1 \right)$

$\boldsymbol{\pi}_{j+1}^T = \boldsymbol{\pi}_j^T \bar{\mathbf{P}}$ is random walk on the graph defined by links

$\boldsymbol{\pi}^T = \lim_{j\rightarrow\infty} \boldsymbol{\pi}_j^T$ = stationary probability distribution

Random Surfer

Web Surfer Randomly Clicks On Links

(Back button not a link)

Long-run proportion of time on page P_i is $\boldsymbol{\pi}_i$

Problems

Dead end page (nothing to click on) ($\boldsymbol{\pi}^T$ not well defined)

Could get trapped into a cycle $(P_i \longrightarrow P_j \longrightarrow P_i)$ (No convergence)

Convergence

Markov chain must be irreducible and aperiodic

Bored Surfer Enters Random URL

Irreducibility Fix: $\bar{\bar{\mathbf{P}}} = \alpha\bar{\mathbf{P}} + (1-\alpha)\mathbf{E}$ $\quad e_{ij} = 1/n \quad \alpha \approx .85$

$$\bar{\bar{\mathbf{P}}} = \alpha\mathbf{P} + \alpha\,\mathbf{a}\,\mathbf{v}^T + (1-\alpha)\mathbf{E}$$

Different $\mathbf{E} = \mathbf{e}\mathbf{v}^T$ and α allow customization & speedup,

yet rank-one update maintained; $\bar{\bar{\mathbf{P}}} = \alpha\mathbf{P} + (\alpha\,\mathbf{a} + (1-\alpha)\,\mathbf{e})\mathbf{v}^T$

Computing π^T

A Big Problem

Solve $\pi^T = \pi^T \bar{\bar{\mathbf{P}}}$ (stationary distribution vector)

$\pi^T(\mathbf{I} - \bar{\bar{\mathbf{P}}}) = 0$ (too big for direct solves)

THE WORLD'S LARGEST MATRIX COMPUTATION

Google's PageRank is an eigenvector of a matrix of order 2.7 billion.

BY CLEVE MOLER

One of the reasons why Google is such an effective search engine is the PageRank™ algorithm, developed by Google's founders, Larry Page and Sergey Brin, when they were graduate students at Stanford University. PageRank is determined entirely by the link structure of the Web. It is recomputed about once a month and does not involve any of the actual content of Web pages or of any individual query. Then, for any particular query, Google finds the pages on the Web that match that query and lists those pages in the order of their PageRank.

Imagine surfing the Web, going from page to page by randomly choosing an outgoing link from one page to get to the next. This can lead to dead ends at pages with no outgoing links, or cycles around cliques of interconnected pages. So, a certain fraction of the time, simply choose a random page from anywhere on the Web. This theoretical random walk of the Web is a *Markov chain* or *Markov process*. The limiting probability that a dedicated random surfer visits any particular page is its PageRank. A page has high rank if it has links to and from other pages with high rank.

Let W be the set of Web pages that can reached by following a chain of hyperlinks starting from a page at Google and let n be the number of pages in W. The set W actually varies with time, but in May 2002, n was about 2.7 billion. Let G be the n-by-n connectivity matrix of

It tells us that the largest eigenvalue of A is equal to one and that the corresponding eigenvector, which satisfies the equation

$$x = Ax,$$

exists and is unique to within a scaling factor. When this scaling factor is chosen so that

$$\Sigma_i x_i = 1$$

then x is the state vector of the Markov chain. The elements of x are Google's PageRank.

If the matrix were small enough to fit in MATLAB, one way to compute the eigenvector x would be to start with a good approximate solution, such as the PageRanks from the previous month, and simply repeat the assignment statement

```
x = Ax
```

until successive vectors agree to within specified tolerance. This is known as the power method and is about the only possible approach for very large n. I'm not sure how Google actually computes PageRank, but one step of the power method would require one pass over a database of Web pages, updating weighted reference counts generated by the hyperlinks between pages.

PART 2

Markov chain methods

for the

Computation of PageRank

Computing π^T

A Big Problem

Solve $\boldsymbol{\pi}^T = \boldsymbol{\pi}^T \bar{\bar{\mathbf{P}}}$ (stationary distribution vector)

$\boldsymbol{\pi}^T(\mathbf{I} - \bar{\bar{\mathbf{P}}}) = \mathbf{0}$ (too big for direct solves)

Start with $\boldsymbol{\pi}_0^T = \mathbf{e}/n$ and iterate $\boldsymbol{\pi}_{j+1}^T = \boldsymbol{\pi}_j^T \bar{\bar{\mathbf{P}}}$ (power method)

Power Method to compute PageRank

```
$\pi_0^T = \mathbf{e}^T/n$
until convergence, do
    $\pi_{j+1}^T = \pi_j^T \bar{\bar{\mathbf{P}}}$        (dense computation)
end
```

Power Method to compute PageRank

```
$\pi_0^T = \mathbf{e}^T/n$
until convergence, do
    X  $\pi_{j+1}^T = \pi_j^T \bar{\bar{\mathbf{P}}}$    (dense computation)
    ●  $\pi_{j+1}^T = \alpha\, \pi_j^T \bar{\mathbf{P}} + (1-\alpha)\, \pi_j^T \mathbf{e}\, \mathbf{v}^T$    (sparser computation)
end
```

Power Method to compute PageRank

```
$\pi_0^T = \mathbf{e}^T/n$
until convergence, do
    X  $\pi_{j+1}^T = \pi_j^T \,\bar{\bar{\mathbf{P}}}$    (dense computation)
    X  $\pi_{j+1}^T = \alpha\, \pi_j^T \,\bar{\mathbf{P}} + (1-\alpha)\, \pi_j^T\, \mathbf{e}\, \mathbf{v}^T$    (sparser computation)
    •  $\pi_{j+1}^T = \alpha\, \pi_j^T \,\mathbf{P} + (\alpha\, \pi_j^T\, \mathbf{a} + (1-\alpha))\, \mathbf{v}^T$    (even less computation)
end
```

- **P** is very, very sparse with about 3-10 nonzeros per row.
- $\Rightarrow$ one vector-matrix mult. is $O(nnz(\mathbf{P})) \approx O(n)$.

Convergence

Can prove $\lambda_2(\bar{\bar{\mathbf{P}}}) = \boldsymbol{\alpha}$

($\Rightarrow$ asymptotic rate of convergence of PageRank method is rate at which $\alpha^k \longrightarrow \mathbf{0}$)

Google

– uses $\alpha = .85$ (5/6, 1/6 interpretation)

– report 50-100 iterations til convergence

– still takes days to converge

Enhancements to the PR power method

- Kamvar et al. Extrapolation
- Kamvar et al. Adaptive PageRank
- Kamvar et al. BlockRank
- Lee et al. Lumpability of Dangling Nodes
- Langville/Meyer: Updating PageRank
- Ipsen/Kirkland: more theory for Langville/Meyer

Kamvar et al. Extrapolation

$$\boldsymbol{\pi}_{k+1}^T = \boldsymbol{\pi}_k^T\mathbf{P} = \boldsymbol{\pi}_k^T\mathbf{e}\boldsymbol{\pi}^T + \lambda_2^k\boldsymbol{\pi}_k^T\mathbf{x}_2\mathbf{y}_2^T + \lambda_3^k\boldsymbol{\pi}_k^T\mathbf{x}_3\mathbf{y}_3^T + \cdots + \lambda_n^k\boldsymbol{\pi}_k^T\mathbf{x}_n\mathbf{y}_n^T$$

Idea: - asympt. rate of conv. $\Rightarrow$ must wait for $\lambda_2^k \rightarrow 0$

- can do better by subtracting off components in $\mathbf{y}_2^T$ direction

(Aitken δ^2 extrapolation)

- next step = annihilate additional components

(Kamvar et al. quadratic extrapolation)

Results: - extrapolation reduces time til convergence for large convergence tolerance $\tau = 10^{-2}$–10^{-3} by about 20%. Speed improvements slow down as τ gets smaller.

Kamvar et al. Adaptive PageRank

Observation

– In the power method, some pages converge to their PR values faster than others.

Idea

– lock states that have already converged to their PR values. Do not involve them in subsequent computations.

(early conv. determined by rel. diff. in successive iterates, $|\pi_i^{(k+1)} - \pi_i^{(k)}| / |\pi_i^{(k)}| < \mathbf{10^{-3}}$)

Results

– speeds computation of PageRank by 15%.

– adaptive PR converges experimentally, but no proof it will converge in theory, or converge to correct PRs.

Kamvar et al. BlockRank

Observation

– Web has domain structure. Lots of links within domain/host, fewer outside domain. This community structure affects convergence of PR algorithm.

Idea

– use aggregation to
  (1) compute local PRs for within each host,
  (2) weight local PRs by importance of corresponding host,
  (3) begin standard PR algorithm with weighted aggregate of local PRs as starting vector.

Results

– speeds computation of PageRank by factor of 2 on some data.

– Also: personalized PR (changing $\mathbf{v}^T$) becomes feasible. $\mathbf{v}^T$ is $1 \times k$ vector where k is # of hosts. Incorporates personalization in aggregation step, where it is computationally efficient.

Lee et al. Lumpability of D nodes

Observation

– rows of $\bar{\mathbf{P}}$ corresponding to D nodes are identical ($\mathbf{v}^T$).
$\Rightarrow$ use theory of lumpable Markov chains.

Idea

– do most of work only on ND part of $\bar{\mathbf{P}}$ by using two aggregation matrices.

Results

– if D nodes make up 4/5 of webpages, then lumpable PageRank takes only 1/5 time of standard PageRank.

– Also: other acceleration methods can be applied to small ND system.

Langville/Meyer Updating

Motivation

- Updating PR is huge problem. Currently done monthly, but web changes hourly.
- Chien et al. use aggregation to focus on pages whose PR is most likely to change.

Idea

- Use iterative aggregation to extend Chien idea.
- Focus on bad states, aggregate good, fast-converging states into one superstate.
- $\Rightarrow$ only work on much smaller aggregated chain.

Results

- speedup by factor of 5-10 on some datasets.

Issue

- Partitioning into good and bad states is hard, and IAD is very sensitive to partition.

Ipsen/Kirkland Updating Theory

Motivation

– L/M prove updating method converges at rate $(\lambda_2(\mathbf{S}_2))^k \to \mathbf{0}$.

– Ipsen/Kirkand wonder: can $\lambda_2(\mathbf{S}_2) > \alpha$?

Results

– $\lambda_2(\mathbf{S}_2) \leq \alpha$ for all partitions.

– $\lambda_2(\mathbf{S}_2) < \alpha$ under two trivial assumptions on **P**.

(**P** is reducible, and at least one page in each essential class does not self-link)

PART 3

Linear System methods

for the

Computation of PageRank

Linear System Formulation

For $\bar{\bar{\mathbf{P}}}$

$$\pi^T(\mathbf{I} - \bar{\bar{\mathbf{P}}}) = 0^T \qquad \text{and} \qquad \pi^T\mathbf{e} = 1.$$

For $\bar{\mathbf{P}}$

$$\pi^T(\mathbf{I} - \alpha\bar{\mathbf{P}}) = (1-\alpha)\mathbf{v}^T \qquad \text{and} \qquad \pi^T\mathbf{e} = 1.$$

For $\mathbf{P}$

$$\pi^T(\mathbf{I} - \alpha\mathbf{P}) = \mathbf{v}^T \qquad \text{and} \qquad \pi^T\mathbf{e} = 1.$$

(**P** is very sparse, 3-10 nonzeros per row)

Properties of $(\mathbf{I} - \alpha\,\mathbf{P})$:

1. $(\mathbf{I} - \alpha\mathbf{P})$ is nonsingular.
2. $(\mathbf{I} - \alpha\mathbf{P})$ is an **M**-matrix.
3. The row sums of $(\mathbf{I} - \alpha\mathbf{P})$ are either $1 - \alpha$ for ND nodes
   or 1 for D nodes.
4. $\|\mathbf{I} - \alpha\mathbf{P}\|_\infty = 1 + \alpha$.
5. Since $(\mathbf{I} - \alpha\mathbf{P})$ is an **M**-matrix, $(\mathbf{I} - \alpha\mathbf{P})^{-1} \geq \mathbf{0}$.
6. The row sums of $(\mathbf{I} - \alpha\mathbf{P})^{-1}$ are equal to 1 for the D nodes
   and less than or equal to $1/(1 - \alpha)$ for the ND nodes.
7. The condition number $\kappa_\infty(\mathbf{I} - \alpha\mathbf{P}) \leq (1 + \alpha)/(1 - \alpha)$.
8. The row of $(\mathbf{I} - \alpha\mathbf{P})^{-1}$ corresponding to D node i is $\mathbf{e}_i^T$.

ND-D Reordering

$$\mathbf{P} = \begin{array}{c|cc} & ND & D \\ \hline ND & \mathbf{P}_{11} & \mathbf{P}_{12} \\ D & 0 & 0 \end{array}.$$

$$(\mathbf{I} - \alpha\mathbf{P}) = \begin{bmatrix} \mathbf{I} - \alpha\mathbf{P}_{11} & -\alpha\mathbf{P}_{12} \\ 0 & \mathbf{I} \end{bmatrix}.$$

$$(\mathbf{I} - \alpha\mathbf{P})^{-1} = \begin{bmatrix} (\mathbf{I} - \alpha\mathbf{P}_{11})^{-1} & \alpha(\mathbf{I} - \alpha\mathbf{P}_{11})^{-1}\mathbf{P}_{12} \\ 0 & \mathbf{I} \end{bmatrix}.$$

Algorithm 1: ND-D Reordering

Solve $\pi^T(\mathbf{I} - \alpha \mathbf{P}) = \mathbf{v}^T$ and $\pi^T \mathbf{e} = 1$.

Algorithm 1:

1. Solve for π_1^T in $\pi_1^T(\mathbf{I} - \alpha \mathbf{P}_{11}) = \mathbf{v}_1^T$.
2. Compute $\pi_2^T = \alpha \pi_1^T \mathbf{P}_{12} + \mathbf{v}_2^T$.
3. Normalize $\pi^T = [\pi_1^T \;\; \pi_2^T] / \| [\pi_1^T \;\; \pi_2^T] \|_1$.

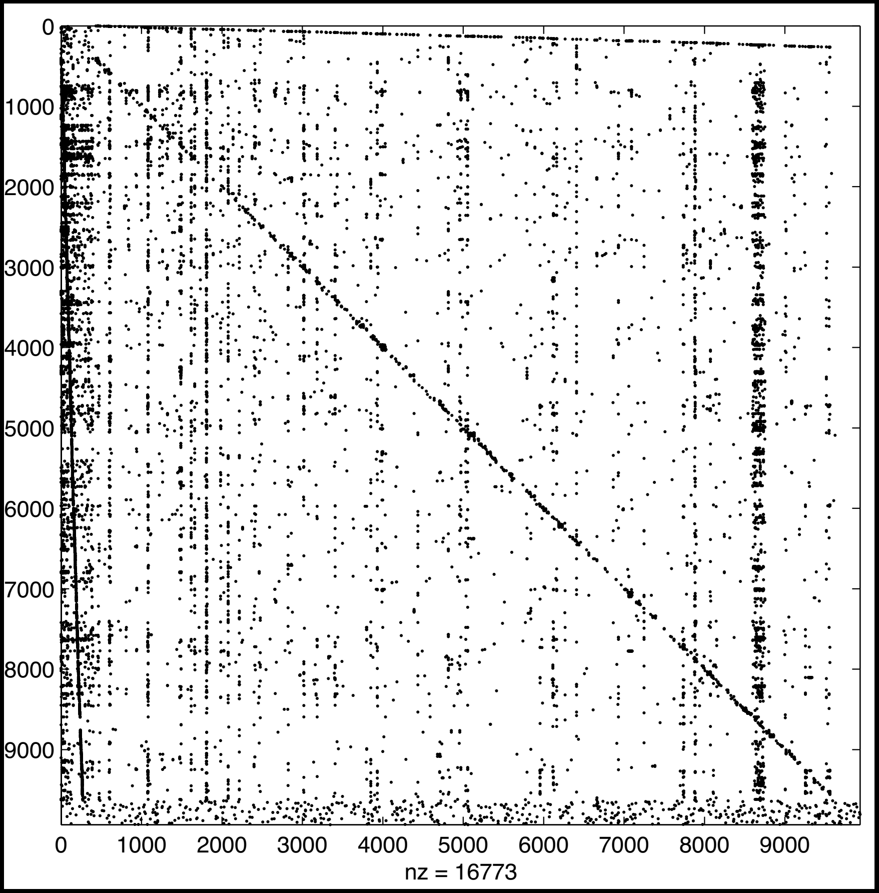
Pro: one small system solve, plus forward substitution.

Analog: Lee et al. lumpable D node Markov formulation.

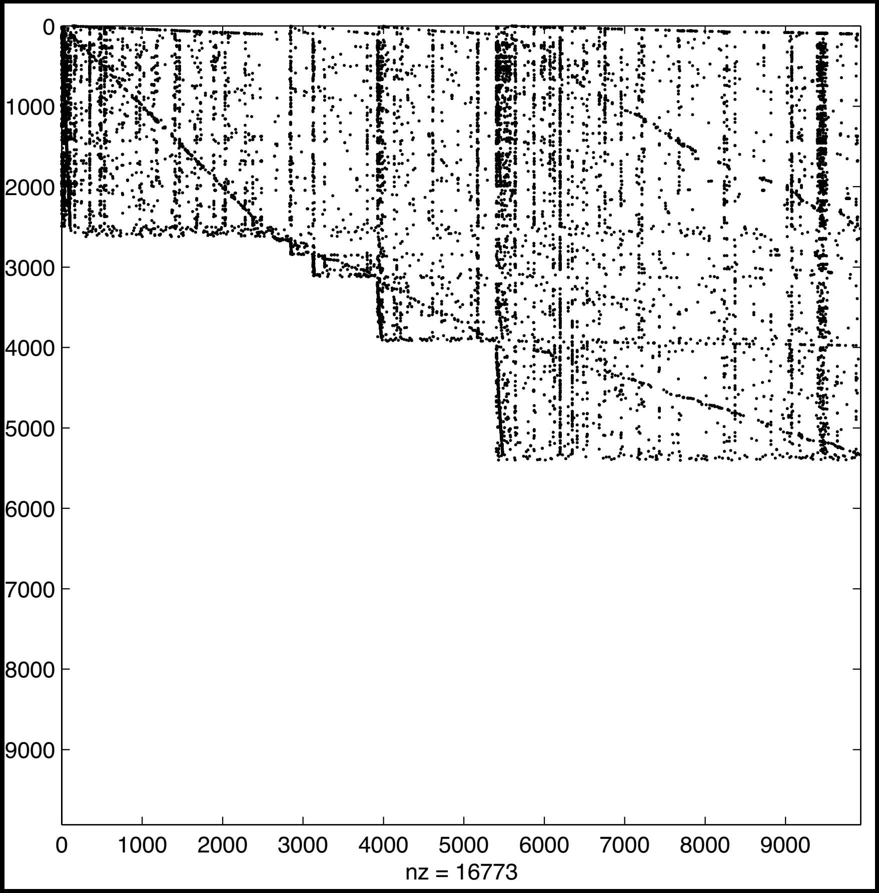
Extension of ND-D Reordering

- Continue locating 0 rows in submatrices of $(\mathbf{I} - \alpha\mathbf{P})$ until no 0 rows remain. Amounts to a reordering of indices.

Before Reordering



After Reordering



Algorithm 2: Recursive ND-D Reordering

Solve $\boldsymbol{\pi}^T(\mathbf{I} - \alpha\mathbf{P}) = \mathbf{v}^T$ and $\boldsymbol{\pi}^T\mathbf{e} = 1$.

Algorithm 2:

1. Reorder the states of the original Markov chain, so that the reordered matrix has the 0 block structure. $O(nnz(\mathbf{P})) \approx 1$ power iter.
2. Solve for $\boldsymbol{\pi}_1^T$ in $\boldsymbol{\pi}_1^T(\mathbf{I} - \alpha\mathbf{P}_{11}) = \mathbf{v}_1^T$. Jacobi method with rate of conv. $\leq \alpha$
3. Compute $\boldsymbol{\pi}_2^T = \alpha\boldsymbol{\pi}_1^T\mathbf{P}_{12} + \mathbf{v}_2^T$.
4. Compute $\boldsymbol{\pi}_3^T = \alpha\boldsymbol{\pi}_1^T\mathbf{P}_{13} + \alpha\boldsymbol{\pi}_2^T\mathbf{P}_{23} + \mathbf{v}_3^T$.
5. Compute $\boldsymbol{\pi}_b^T = \alpha\boldsymbol{\pi}_1^T\mathbf{P}_{1b} + \alpha\boldsymbol{\pi}_2^T\mathbf{P}_{2b} + \cdots + \alpha\boldsymbol{\pi}_{b-1}^T\mathbf{P}_{b-1,b} + \mathbf{v}_b^T$. $O(nnz(\mathbf{P}))$
6. Normalize $\boldsymbol{\pi}^T = [\boldsymbol{\pi}_1^T \;\; \boldsymbol{\pi}_2^T \;\; \cdots \;\; \boldsymbol{\pi}_b^T] / \|[\boldsymbol{\pi}_1^T \;\; \boldsymbol{\pi}_2^T \;\; \cdots \;\; \boldsymbol{\pi}_b^T]\|_1$.

Pro: even smaller system solve, plus forward substitution.

Speedup: by factor of $nnz(\mathbf{P})/nnz(\mathbf{P}_{11})$ (estimated)

Results of Reordered PageRank

		EPA.dat	CA.dat	NCS.dat	ND.dat	SU450k.dat
PR	$Time$	3.80	9.63	13.17	177.16	237.37
	$Iter.$	159	176	162	166	164
	$n(\mathbf{P})$	5,042	9,664	10,000	325,729	451,237
	$nz(\mathbf{P})$	9,563	16,873	101,118	1,497,134	1,082,604
$RePR$	$Time$	.59	1.22	7.65	130.54	52.84
	$Iter.$	155	169	160	170	145
	b	10	9	5	18	12
	$n(\mathbf{P}_{11})$	704	2,622	7,136	325,729	84,861
	$nz(\mathbf{P}_{11})$	1,330	5,238	79,230	1,191,761	267,566
$Speed$	$Est.$	7.2	6.4	1.3	1.3	4.0
Up	$Act.$	6.4	7.9	1.7	1.4	4.5

- can do no worse than original PR power method
- Speedup is dataset-dependent

Conclusions

- there are many solution methods for the PageRank problem.
- nearly all methods have stayed in the Markov chain realm.
- yet, solution methods in the linear system realm may provide powerful alternatives.
- many of these methods can be combined to achieve even greater speedups.
- We are moving closer to lofty goal of computing real-time personalized PageRank.