ACCELERATING GOOGLE'S PAGERANK

Liz & Steve

Background

- When a search query is entered in Google, the relevant results are returned to the user in an order that Google predetermines.
- This order is determined by each web page's PageRank value.
- Google's system of ranking web pages has made it the most widely used search engine available.
- The PageRank vector is a stochastic vector that gives a numerical value ($0<val<1$) to each web page.
- To compute this vector, Google uses a matrix denoting links between web pages.

Background

Main ideas:

- Web pages with the highest number of inlinks should receive the highest rank.
- The rank of a page P is to be determined by adding the (weighted) ranks of all the pages linking to P.

Background

- Problem: Compute a PageRank vector that contains an meaningful rank of every web page

$$r_k(P_i) = \sum_{Q \varepsilon B_{P_i}} \frac{r_{k-1}(Q)}{|Q|} \qquad v_k = \left[r_k(P_1) \quad r_k(P_2) \quad \cdots \quad r_k(P_n) \right]^T$$

$$v^T{}_k = v^T{}_{k-1} H; \qquad H_{ij} = \begin{cases} \dfrac{1}{\|P_i\|} & \text{if there is a link} \\ 0 & \text{if no link} \end{cases}$$

Power Method

- The PageRank vector is the dominant eigenvector of the matrix H…after modification
- Google currently uses the Power Method to compute this eigenvector. However, H is often not suitable for convergence.
- Power Method: ${v_k}^T = {v_{k-1}}^T H$

$$\text{typically, } H \text{ is } \begin{cases} \text{not stochastic} \\ \text{not irreducible} \end{cases}$$

Creating a usable matrix

$$G = \alpha(H + au^T) + (1-\alpha)eu^T$$

$$\text{where} \quad 0 < \alpha < 1$$

e is a vector of ones and *u* (for the moment) is an arbitrary probabilistic vector.

Using the Power Method

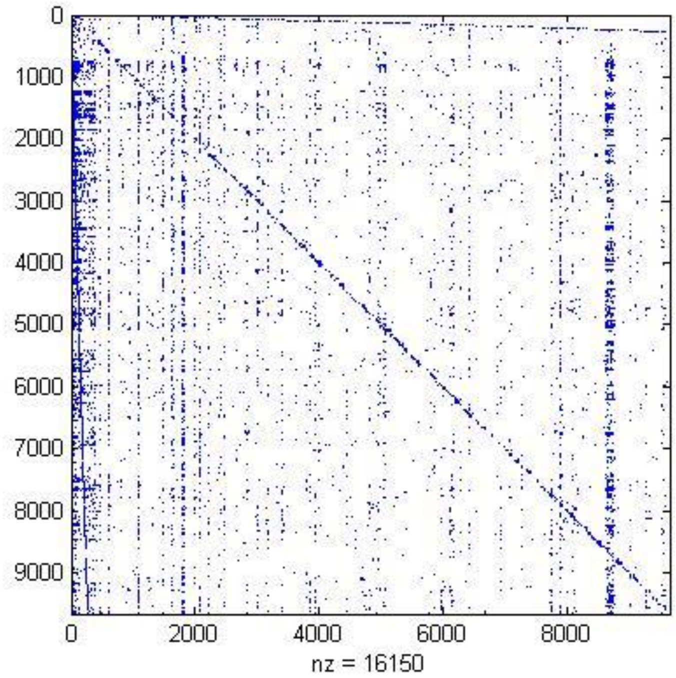
$$\begin{aligned} v_{k+1}^T &= v_k^T G \\ &= \alpha v_k^T H + \alpha v_k^T u a^T + (1-\alpha) u^T \end{aligned}$$

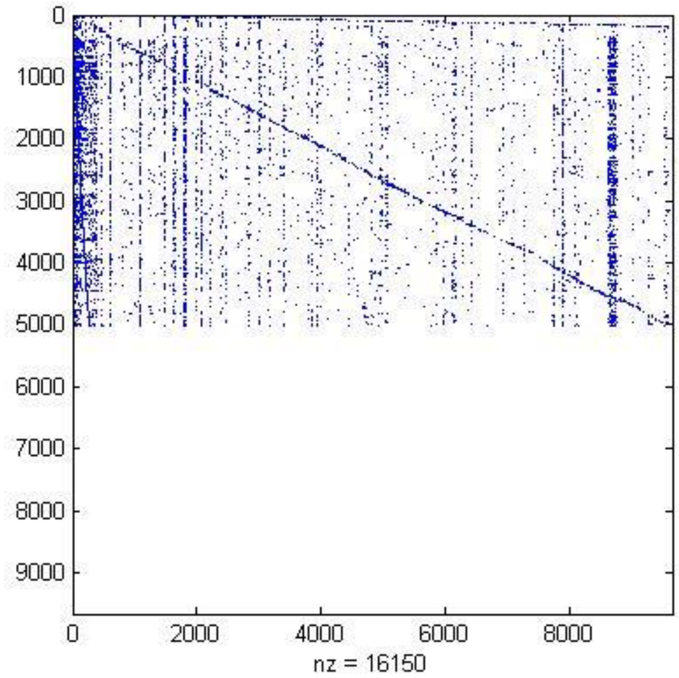
- The rate of convergence is: $\frac{\|\lambda_2\|}{\|\lambda_1\|}$, where λ_1 is the dominant eigenvalue and λ_2 is the aptly named subdominant eigenvalue

Alternative Methods:
Linear Systems

$$v^T = v^T G \Leftrightarrow \begin{cases} x^T (I - \alpha H) = u^T \\ v = x / \|x\| \end{cases}$$

Langville & Meyer's reordering





Alternative Methods: Iterative Aggregation/Disaggregation (IAD)

$$G = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$A = \begin{bmatrix} G_{11} & G_{12}e \\ u_2^T G_{21} & 1 - u_2^T G_{21} e \end{bmatrix} \qquad w = \begin{bmatrix} w_1^T \\ c \end{bmatrix}$$

$$v = \begin{bmatrix} w_1^T \\ c u_2^T \end{bmatrix}$$

IAD Algorithm

- Form the matrix $\tilde{A}$
- Find the stationary vector $\tilde{w}^T = \begin{bmatrix} \tilde{w}_1^T & c \end{bmatrix}$
- $v_k^T = \begin{bmatrix} \tilde{w}_1^T & c\tilde{u}_2 \end{bmatrix}$
- $v_{k+1}^T = v_k^T G$
- If $\left\| v_{k+1}^T - v_k^T \right\| < \varepsilon$, then stop. Otherwise,

  $\tilde{u}_2 = (v_{k+1})_y / \left\| (v_{k+1})_y \right\|_1$

New Ideas:
The Linear System In IAD

$$\begin{bmatrix}\tilde{w}_1^T & c\end{bmatrix}\begin{bmatrix}G_{11} & G_{12}e \\ \tilde{u}_2^T G_{21} & \tilde{u}_2^T G_{22}e\end{bmatrix} = \begin{bmatrix}\tilde{w}_1^T & c\end{bmatrix}$$

$$\tilde{w}_1^T (I - G_{11}) = c\tilde{u}_2^T G_{21}$$

$$\tilde{w}_1^T G_{12}e = c(1 - \tilde{u}_2^T G_{22}e) = c\tilde{u}_2^T G_{21}e$$

New Ideas: Finding c and $\tilde{w}_1$

1. Solve $(I - G_{11})^T \tilde{w}_1 = c G_{21}{}^T \tilde{u}_2$

2. Let $c = \dfrac{\tilde{w}_1^T G_{12} e}{\tilde{u}_2^T G_{21} e}$

3. Continue until $\left\| \tilde{w}_1 - \tilde{w}_1(\text{old}) \right\| < \varepsilon$

Functional Codes

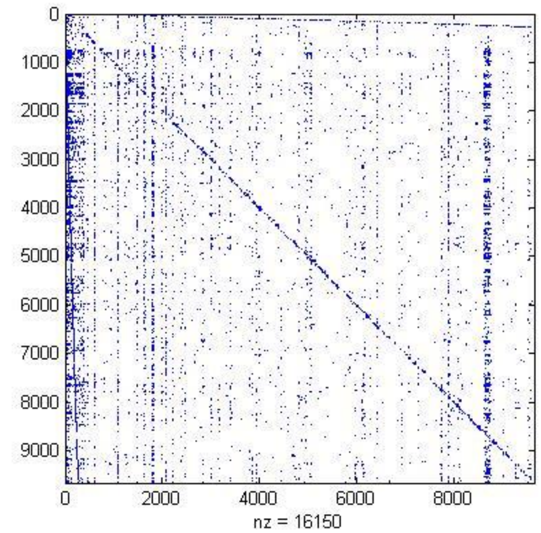
- Power Method
  - We duplicated Google's formulation of the power method in order to have a base time with which to compare our results
- A basic linear solver
  - We used Gauss-Seidel method to solve the very basic linear system: $x^T(I-\alpha H)=u^T$
  - We also experimented with reordering by row degree before solving the aforementioned system.
- Langville & Meyer's Linear System Algorithm
  - Used as another time benchmark against our algorithms

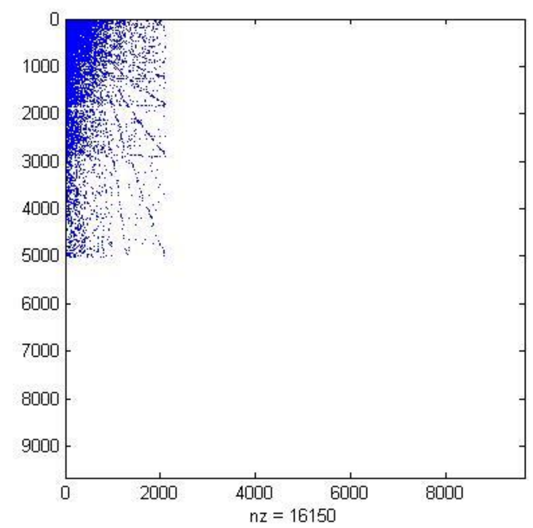
Functional Codes (cont'd)

- IAD - using power method to find w_1
  - We used the power method to find the dominant eigenvector of the aggregated matrix A. The rescaling constant, c, is merely the last entry of the dominant eigenvector
- IAD – using a linear system to find w_1
  - We found the dominant eigenvector as discussed earlier, using some new reorderings

And now… The Winner!

- Power Method with preconditioning
  - Applying a row and column reordering by decreasing degree *almost always* reduces the number of iterations required to converge.

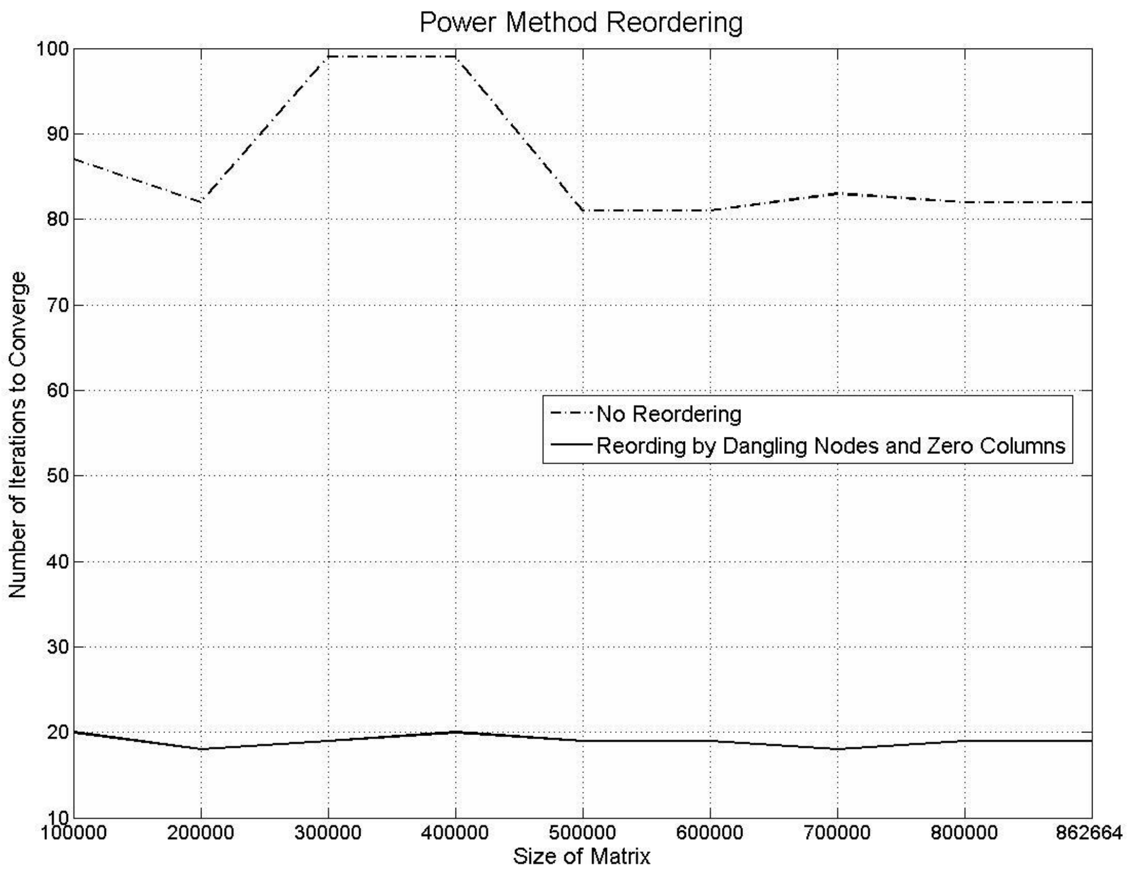




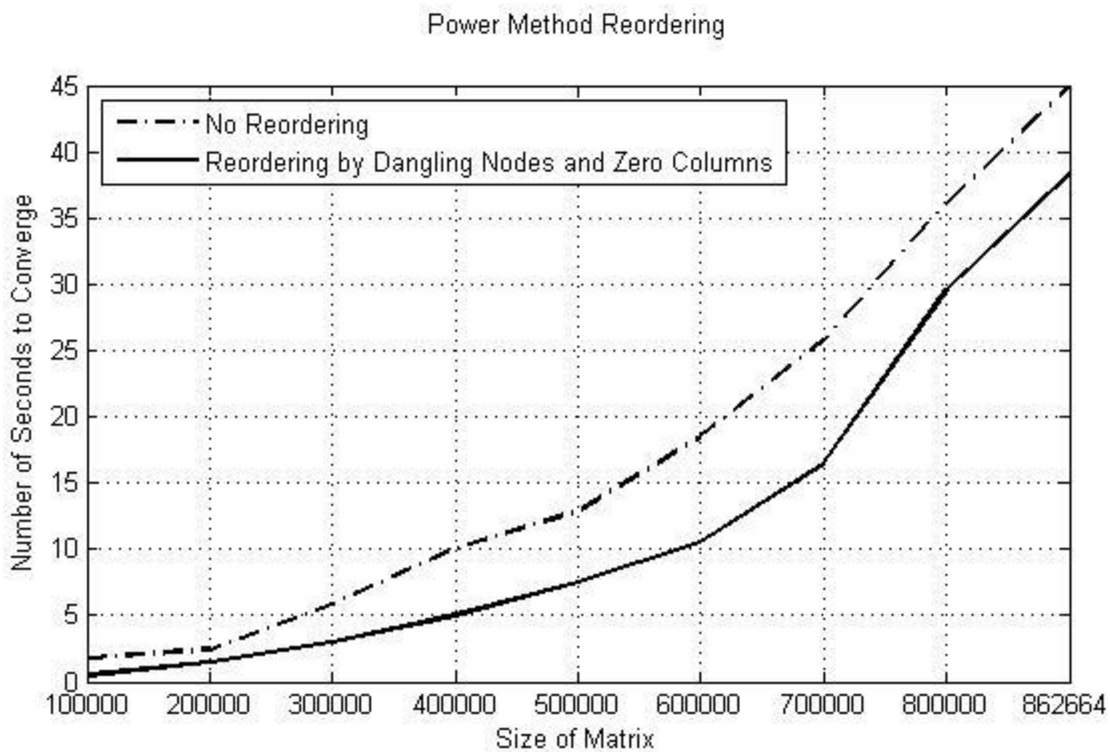
Why this works…

1. The power method converges faster as the magnitude of the subdominant eigenvalue decreases
2. Tugrul Dayar found that partitioning a matrix in such a way that its off-diagonal blocks are close to 0, forces the dominant eigenvalue of the iteration matrix closer to 0. This is somehow related to the subdominant eigenvalue of the coefficient matrix in power method.

Decreased Iterations



Decreased Time



Some Comparisons

	Calif	Stan	CNR	Stan Berk	EU
Sample Size	87	57	66	69	58
Interval Size	100	5000	5000	10000	15000
Mean Time Pwr/Reorder	1.6334	2.2081	2.1136	1.4801	2.2410
STD Time Pwr/Reorder	0.6000	0.3210	0.1634	0.2397	0.2823
Mean Iter Pwr/Reorder	2.0880	4.3903	4.3856	3.7297	4.4752
STD Iter Pwr/Reorder	0.9067	0.7636	0.7732	0.6795	0.6085
Favorable	100.00%	98.25%	100.00%	100.00%	100.00%

Matrix	Code	Time (sec)	GS-I	PM-I	w_1-I	Size G_{11}
California	$I - \alpha H$	0.119873	53			
	$I - \alpha H$ w/ reordering	0.060155	9			
9664 Nodes	Meyer's Algorithm	0.061451	18			
1.73E-4 Sparsity	IAD w/ Power Method	0.125680		13	46	1000
4637 Dang Nodes	IAD w/ system & SOR	0.690981		49	181	1000
	Power Method	0.084233		97		
	Power Method w/ reordering	**0.035091**		**22**		
Stanford	$I - \alpha H$	4.673486	56			
	$I - \alpha H$ w/ reordering	5.290621	55			
281903 Nodes	Meyer's Algorithm	5.584932	56			
2.91E-5 Sparsity	IAD w/ Power Method	4.485200		23	61	1500
172 Dang Nodes	IAD w/ system & SOR	3.684937		23	57	750
	Power Method	4.950945		90		
	Power Method w/ reordering	**2.276534**		**19**		
CNR (2000)	$I - \alpha H$	5.537661	50			
	$I - \alpha H$ w/ reordering	4.942164	24			
325557 Nodes	Meyer's Algorithm	4.380352	23			
3.03E-5 Sparsity	IAD w/ Power Method	5.328511		19	53	10000
78056 Dang Nodes	IAD w/ system & SOR	4.172643		19	48	1000
	Power Method	7.018947		87		
	Power Method w/ reordering	**3.102524**		**19**		
Stanford-Berkley	$I - \alpha H$	9.320476	55			
	$I - \alpha H$ w/ reordering	11.704303	46			
685230 Nodes	Meyer's Algorithm	14.450852	62			
1.62E-5 Sparsity	IAD w/ Power Method	30.842767		90	237	4500
4744 Dang Nodes	IAD w/ system & SOR	24.503765		90	256	1000
	Power Method	8.958482		90		
	Power Method w/ reordering	**7.095738**		**28**		
EU(2005)	$I - \alpha H$	33.467238	50			
	$I - \alpha H$ w/ reordering	53.200779	37			
862664 Nodes	Meyer's Algorithm	45.817442	39			
2.58E-5 Sparsity	IAD w/ Power Method	25.945539		15	41	10000
4744 Dang Nodes	IAD w/ system & SOR	22.499486		16	48	10000
	Power Method	38.617480		82		
	Power Method w/ reordering	**16.727974**		**19**		
IN (2004)	$I - \alpha H$	38.29	52			
	$I - \alpha H$ w/ reordering	31.46	28			
1382908 Nodes	Meyer's Algorithm	29.22	25			
8.85E-6 Sparsity	**IAD w/ Power Method**	**26.69**		**18**	**44**	**10000**
282306 Dang Nodes	IAD w/ system & SOR	23.95		18	46	10000
	Power Method	46.09		88		
	Power Method w/ reordering	31.59		68		
Wikipedia	$I - \alpha H$	44.02	54			
	$I - \alpha H$ w/ reordering	52.95	22			
1634989 Nodes	Meyer's Algorithm	64.01	54			
7.39E-6 Sparsity	IAD w/ Power Method	46.32		19	54	10000
72556 Dang Nodes	IAD w/ system & SOR	119.94		101	400	10000
	Power Method	33.86		59		
	Power Method w/ reordering	**18.89**		**12**		

Future Research

- Test with more advanced numerical algorithms for linear systems (Krylov subspaces methods and preconditioners, i.e. GMRES, BICG, ILU, etc.)
- Test with other reorderings for all methods
- Test with larger matrices (find a supercomputer that works)
- Attempt a theoretical proof of the decrease in the magnitude of the subdominant eigenvalue as result of reorderings.
- Convert codes to low level languages (C++, etc.)
- Decode MATLAB’s spy

Langville & Meyer's Algorithm

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & 0 \end{bmatrix}$$

$$(I - \alpha H)^{-1} = \begin{bmatrix} (I - \alpha H_{11})^{-1} & \alpha (I - \alpha H_{11})^{-1} H_{12} + v_2{}^T \\ 0 & I \end{bmatrix}$$

$$x^T = \begin{bmatrix} v_1{}^T (I - \alpha H_{11})^{-1} & \alpha v_1{}^T (I - \alpha H_{11})^{-1} H_{12} + v_2{}^T \end{bmatrix}$$

$$x_1^T (I - \alpha H_{11}) = v_1{}^T$$

$$x_2{}^T = \alpha x_1^T H_{12} + v_2{}^T$$

Theorem: Perron-Frobenius

- If A_{nxn} is a non-negative irreducible matrix, then
  - $p(A)$ is a positive eigenvalue of A
  - There is a positive eigenvector v associated with $p(A)$
  - $p(A)$ has algebraic and geometric multiplicity 1

The Power Method: Two Assumptions

- The complete set of eigenvectors $v_1 \ldots v_n$ are linearly independent
- For each eigenvector there exists eigenvalues such that $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$