CS6220: DATA MINING TECHNIQUES

Mining Graph/Network Data

Instructor: Yizhou Sun

yzsun@ccs.neu.edu

November 16, 2015

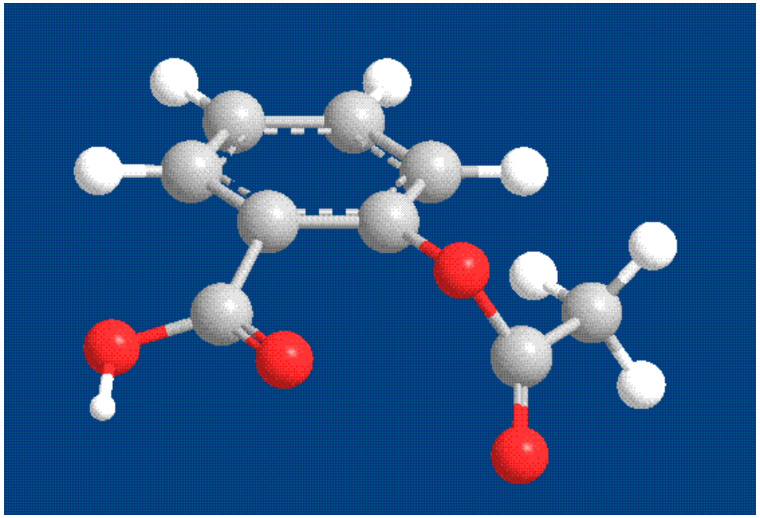
Methods to Learn

	Matrix Data	Text Data	Set Data	Sequence Data	Time Series	Graph & Network	Images
Classification	Decision Tree; Naïve Bayes; Logistic Regression SVM; kNN			HMM		Label Propagation*	Neural Network
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models; kernel k-means*	PLSA				SCAN*; Spectral Clustering*	
Frequent Pattern Mining			Apriori; FP-growth	GSP; PrefixSpan			
Prediction	Linear Regression				Autoregression		
Similarity Search					DTW	**P-PageRank**	
Ranking						**PageRank**	

Mining Graph/Network Data

- Introduction to Graph/Network Data
- PageRank
- Personalized PageRank
- Summary

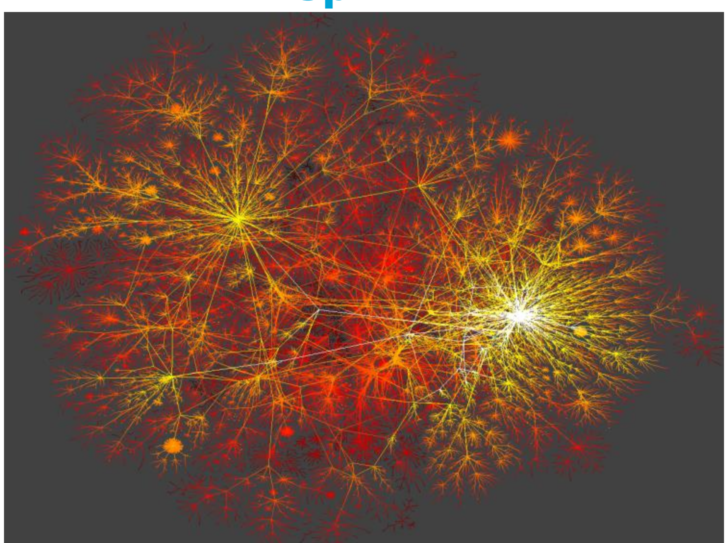
Graph, Graph, Everywhere



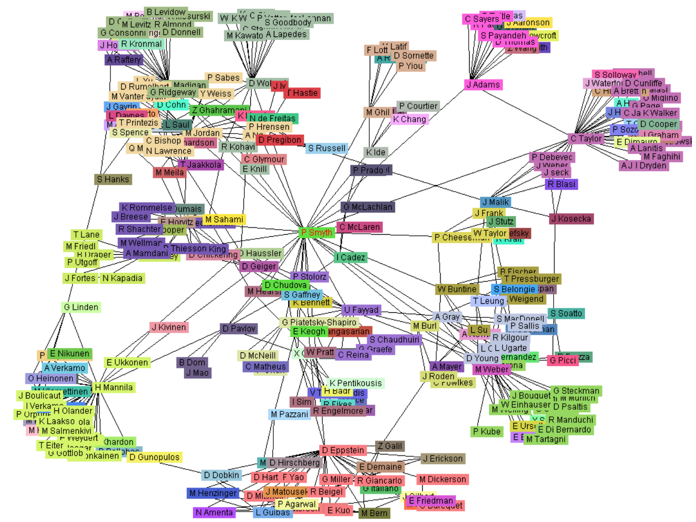
Aspirin

Yeast protein interaction network

from H. Jeong et al Nature 411, 41 (2001)



Internet



Co-author network

Why Graph Mining?

- Graphs are ubiquitous
  - Chemical compounds (Cheminformatics)
  - Protein structures, biological pathways/networks (Bioinformactics)
  - Program control flow, traffic flow, and workflow analysis
  - XML databases, Web, and social network analysis
- Graph is a general model
  - Trees, lattices, sequences, and items are degenerated graphs
- Diversity of graphs
  - Directed vs. undirected, labeled vs. unlabeled (edges & vertices), weighted, with angles & geometry (topological vs. 2-D/3-D)
- Complexity of algorithms: many problems are of high complexity

Representation of a Graph

- $G =< V, E >$
  - $V = \{u_1, \dots, u_n\}$: node set
  - $E \subseteq V \times V$: edge set
- Adjacency matrix
  - $A = \{a_{ij}\}, i, j = 1, \dots, n$
    - $a_{ij} = 1, if \ < u_i, u_j > \in E$
    - $a_{ij} = 0, if \ < u_i, u_j > \notin E$
  - Undirected graph vs. Directed graph
    - $A = A^{\mathrm{T}}\ vs. A \neq A^{\mathrm{T}}$
  - Weighted graph
    - Use W instead of A, where w_{ij} represents the weight of edge $< u_i, u_j >$

Mining Graph/Network Data

- Introduction to Graph/Network Data
- PageRank
- Personalized PageRank
- Summary

The History of PageRank

- PageRank was developed by Larry Page (hence the name *Page*-Rank) and Sergey Brin.

- It is first as part of a research project about a new kind of search engine. That project started in 1995 and led to a functional prototype in 1998.

- Shortly after, Page and Brin founded Google.

Ranking web pages

- Web pages are not equally “important”
  - www.cnn.com vs. a personal webpage
- Inlinks as votes
  - The more inlinks, the more important
- Are all inlinks equal?
  - Recursive question!

Simple recursive formulation

- Each link's vote is proportional to the importance of its source page
- If page P with importance x has n outlinks, each link gets x/n votes
- Page P's own importance is the sum of the votes on its inlinks

Matrix formulation

- Matrix **M** has one row and one column for each web page
- Suppose page j has n outlinks
  - If j -> i, then $M_{ij}=1/n$
  - Else $M_{ij}=0$
- **M** is a column stochastic matrix
  - Columns sum to 1
- Suppose **r** is a vector with one entry per web page
  - r_i is the importance score of page i
  - Call it the rank vector
  - $|\mathbf{r}| = 1$

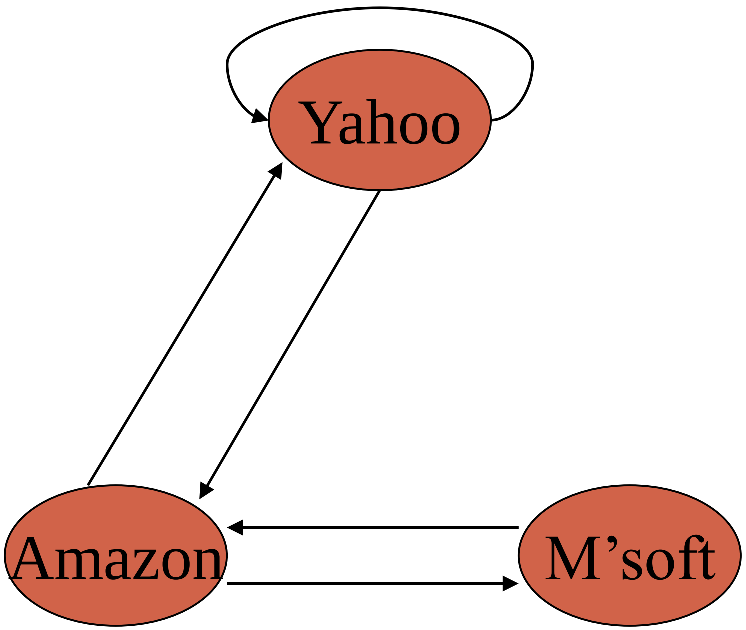
Eigenvector formulation

- The flow equations can be written

$$\mathbf{r} = \mathbf{M}\mathbf{r}$$

- So the rank vector is an eigenvector of the stochastic web matrix
  - In fact, its first or principal eigenvector, with corresponding eigenvalue 1

Example



$$y = y/2 + a/2$$

$$a = y/2 + m$$

$$m = a/2$$

$$\begin{array}{c|ccc|} & y & a & m \\ \hline y & 1/2 & 1/2 & 0 \\ a & 1/2 & 0 & 1 \\ m & 0 & 1/2 & 0 \\ \hline \end{array}$$

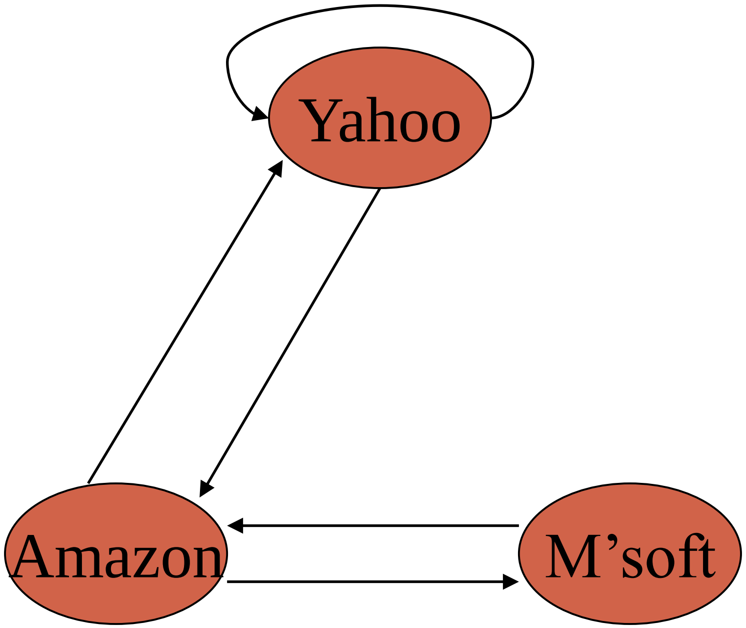
r = Mr

$$\boxed{\begin{matrix} y \\ a \\ m \end{matrix}} = \boxed{\begin{matrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{matrix}} \boxed{\begin{matrix} y \\ a \\ m \end{matrix}}$$

Power Iteration method

- Simple iterative scheme (aka relaxation)
- Suppose there are N web pages
- Initialize: $\mathbf{r}^0 = [1/N,\ldots.,1/N]^T$
- Iterate: $\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$
- Stop when $|\mathbf{r}^{k+1} - \mathbf{r}^k|_1 < \varepsilon$
  - $|\mathbf{x}|_1 = \sum_{1\leq i\leq N}|x_i|$ is the L_1 norm
  - Can use any other vector norm e.g., Euclidean

Power Iteration Example



$$
\begin{array}{c|ccc|}
 & y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1 \\
m & 0 & 1/2 & 0 \\
\hline
\end{array}
$$

$$
\begin{array}{l}
y \\ a \\ m
\end{array}
=
\begin{array}{c}
1/3 \\ 1/3 \\ 1/3
\end{array}
\quad
\begin{array}{c}
1/3 \\ 1/2 \\ 1/6
\end{array}
\quad
\begin{array}{c}
5/12 \\ 1/3 \\ 1/4
\end{array}
\quad
\begin{array}{c}
3/8 \\ 11/24 \\ 1/6
\end{array}
\quad
\cdots
\quad
\begin{array}{c}
2/5 \\ 2/5 \\ 1/5
\end{array}
$$

$$
\boldsymbol{r}_0 \quad \boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{r}_3 \quad \ldots \quad \boldsymbol{r}^*
$$

Random Walk Interpretation

- Imagine a random web surfer
  - At any time t, surfer is on some page P
  - At time $t+1$, the surfer follows an outlink from P uniformly at random
  - Ends up on some page Q linked from P
  - Process repeats indefinitely
- Let $\mathbf{p}(t)$ be a vector whose i^{th} component is the probability that the surfer is at page i at time t
  - $\mathbf{p}(t)$ is a probability distribution on pages

The stationary distribution

- Where is the surfer at time t+1?
  - Follows a link uniformly at random
  - $\mathbf{p}(t+1) = \mathbf{M}\mathbf{p}(t)$
- Suppose the random walk reaches a state such that $\mathbf{p}(t+1) = \mathbf{M}\mathbf{p}(t) = \mathbf{p}(t)$
  - Then $\mathbf{p}(t)$ is called a stationary distribution for the random walk
- Our rank vector **r** satisfies $\mathbf{r} = \mathbf{M}\mathbf{r}$
  - So it is a stationary distribution for the random surfer

Existence and Uniqueness

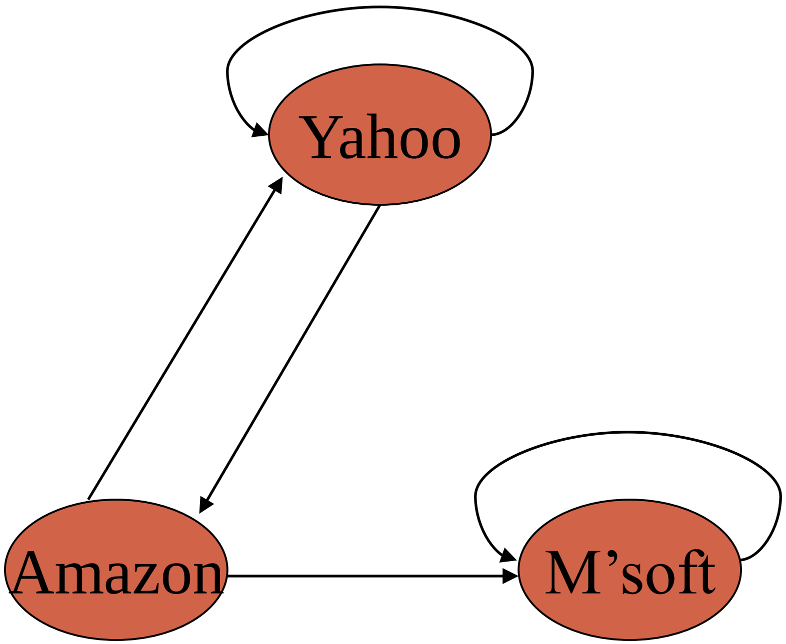
A central result from the theory of random walks (aka Markov processes):

For graphs that satisfy certain conditions, the stationary distribution is unique and eventually will be reached no matter what the initial probability distribution at time $t = 0$.

Spider traps

- A group of pages is a spider trap if there are no links from within the group to outside the group
  - Random surfer gets trapped
- Spider traps violate the conditions needed for the random walk theorem

Microsoft becomes a spider trap



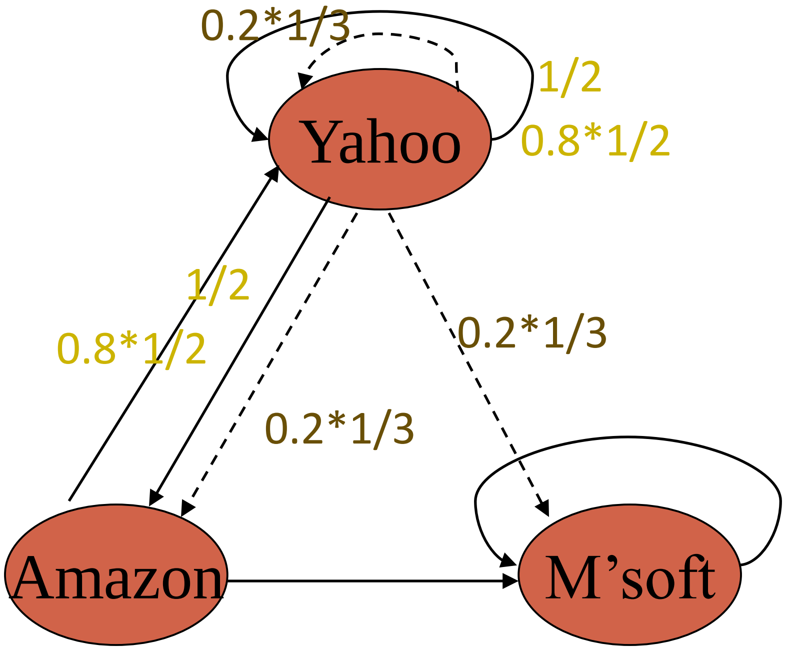
$$
\begin{array}{c|ccc|}
 & y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 0 \\
m & 0 & 1/2 & 1 \\
\hline
\end{array}
$$

$$
\begin{array}{l}
y \\ a \\ m
\end{array}
=
\begin{array}{c}
1/3 \\ 1/3 \\ 1/3
\end{array}
\quad
\begin{array}{c}
1/3 \\ 1/6 \\ 1/2
\end{array}
\quad
\begin{array}{c}
1/4 \\ 1/6 \\ 7/12
\end{array}
\quad
\begin{array}{c}
5/24 \\ 1/8 \\ 2/3
\end{array}
\quad \ldots \quad
\begin{array}{c}
0 \\ 0 \\ 1
\end{array}
$$

Random teleports

- The Google solution for spider traps
- At each time step, the random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1-\beta$, jump to some page uniformly at random
  - Common values for β are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps

Random teleports ($\beta = 0.8$)

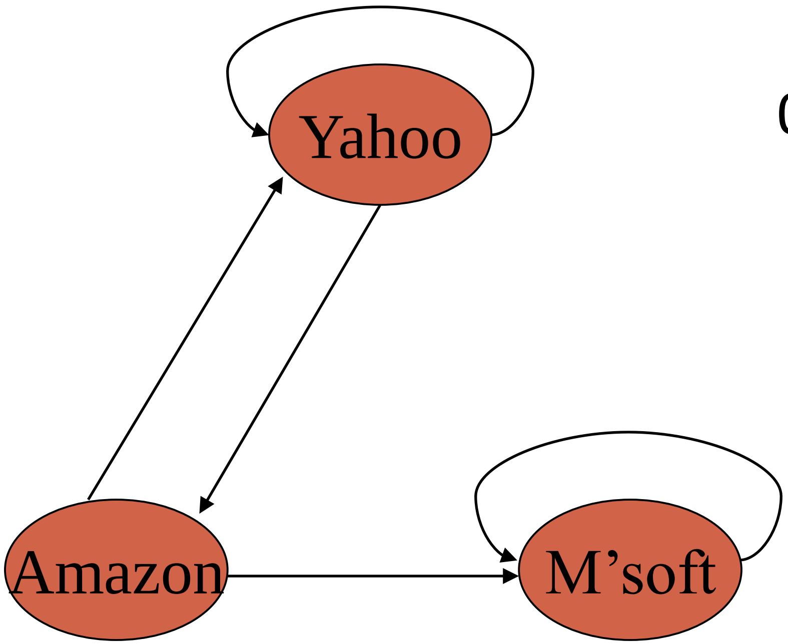


$$\begin{array}{cc} & y \\ y & \\ a & \\ m & \end{array} \boxed{\begin{array}{c} 1/2 \\ 1/2 \\ 0 \end{array}} \qquad \begin{array}{c} y \\ 0.8* \boxed{\begin{array}{c} 1/2 \\ 1/2 \\ 0 \end{array}} \end{array} \quad \begin{array}{c} y \\ + 0.2* \boxed{\begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array}} \end{array}$$

$$0.8 \boxed{\begin{array}{ccc} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{array}} \quad + 0.2 \boxed{\begin{array}{ccc} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{array}}$$

$$\begin{array}{c} y \\ a \\ m \end{array} \boxed{\begin{array}{ccc} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{array}}$$

Random teleports ($\beta = 0.8$)



$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$\begin{array}{c} y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{bmatrix} 0.333 \\ 0.333 \\ 0.333 \end{bmatrix} \begin{bmatrix} 0.333 \\ 0.200 \\ 0.467 \end{bmatrix} \begin{bmatrix} 0.280 \\ 0.200 \\ 0.520 \end{bmatrix} \begin{bmatrix} 0.259 \\ 0.179 \\ 0.563 \end{bmatrix} \dots \begin{bmatrix} 7/33 \\ 5/33 \\ 21/33 \end{bmatrix}$$

Matrix formulation

- Suppose there are N pages
  - Consider a page j, with set of outlinks O(j)
  - We have $M_{ij} = 1/|O(j)|$ when j->i and $M_{ij} = 0$ otherwise
  - The random teleport is equivalent to
    - adding a teleport link from j to every other page with probability $(1-\beta)/N$
    - reducing the probability of following each outlink from $1/|O(j)|$ to $\beta/|O(j)|$
    - Equivalent: tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

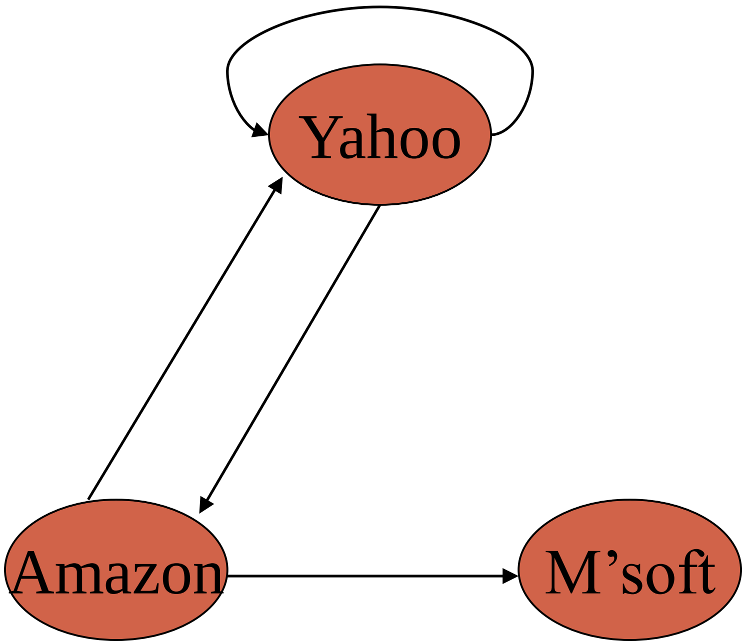
PageRank

- Construct the N-by-N matrix **A** as follows
  - $A_{ij} = \beta M_{ij} + (1-\beta)/N$
- Verify that **A** is a stochastic matrix
- The page rank vector **r** is the principal eigenvector of this matrix
  - satisfying $\mathbf{r} = \mathbf{A}\mathbf{r}$
- Equivalently, **r** is the stationary distribution of the random walk with teleports

Dead ends

- Pages with no outlinks are "dead ends" for the random surfer
  - Nowhere to go on next step

Microsoft becomes a dead end



$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$\begin{array}{c} y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 1/15 \end{bmatrix}$$

Non-stochastic!

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array} \quad \begin{array}{c} 1/3 \\ 0.2 \\ 0.2 \end{array} \quad \ldots \quad \begin{array}{c} 0 \\ 0 \\ 0 \end{array}$$

Dealing with dead-ends

- Teleport
  - Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly
- Prune and propagate
  - Preprocess the graph to eliminate dead-ends
  - Might require multiple passes
  - Compute page rank on reduced graph
  - Approximate values for deadends by propagating values from reduced graph

Computing PageRank

- Key step is matrix-vector multiplication
  - $\mathbf{r}^{\text{new}} = \mathbf{A}\mathbf{r}^{\text{old}}$
- Easy if we have enough main memory to hold $\mathbf{A}$, $\mathbf{r}^{\text{old}}$, $\mathbf{r}^{\text{new}}$
- Say N = 1 billion pages
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix A has N^2 entries
    - 10^{18} is a large number!

Rearranging the equation

$\mathbf{r} = \mathbf{A}\mathbf{r}$, where

$A_{ij} = \beta M_{ij} + (1-\beta)/N$

$r_i = \sum_{1\leq j\leq N} A_{ij}\, r_j$

$r_i = \sum_{1\leq j\leq N} [\beta M_{ij} + (1-\beta)/N]\, r_j$

$\quad = \beta \sum_{1\leq j\leq N} M_{ij}\, r_j + (1-\beta)/N \sum_{1\leq j\leq N} r_j$

$\quad = \beta \sum_{1\leq j\leq N} M_{ij}\, r_j + (1-\beta)/N$, since $|\mathbf{r}| = 1$

$\mathbf{r} = \beta\mathbf{M}\mathbf{r} + [(1-\beta)/N]_N$

where $[x]_N$ is an N-vector with all entries x

Sparse matrix formulation

- We can rearrange the page rank equation:
  - $\mathbf{r} = \beta\mathbf{M}\mathbf{r} + [(1-\beta)/N]_N$
  - $[(1-\beta)/N]_N$ is an N-vector with all entries $(1-\beta)/N$
- **M** is a sparse matrix!
  - 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute $\mathbf{r}^{new} = \beta\mathbf{M}\mathbf{r}^{old}$
  - Add a constant value $(1-\beta)/N$ to each entry in $\mathbf{r}^{new}$

Sparse matrix encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - say 10N, or 4*10*1 billion = 40GB
  - still won't fit in memory, but will fit on disk

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Basic Algorithm

- Assume we have enough RAM to fit $\mathbf{r}^{new}$, plus some working memory
  - Store $\mathbf{r}^{old}$ and matrix $\mathbf{M}$ on disk

Basic Algorithm:

- Initialize: $\mathbf{r}^{old} = [1/N]_N$
- Iterate:
  - Update: Perform a sequential scan of $\mathbf{M}$ and $\mathbf{r}^{old}$ to update $\mathbf{r}^{new}$
  - Write out $\mathbf{r}^{new}$ to disk as $\mathbf{r}^{old}$ for next iteration
  - Every few iterations, compute $|\mathbf{r}^{new}-\mathbf{r}^{old}|$ and stop if it is below threshold
    - Need to read in both vectors into memory

Mining Graph/Network Data

- Introduction to Graph/Network Data
- PageRank
- Personalized PageRank
- Summary

Personalized PageRank

- Query-dependent Ranking
  - For a query webpage q, which webpages are most important to q?
  - The relative important webpages to different queries would be different

Calculation of P-PageRank

- Recall PageRank calculation:
  - $\mathbf{r} = \beta\mathbf{M}\mathbf{r} + [(1-\beta)/N]_N$ or
  - $\mathbf{r} = \beta\mathbf{M}\mathbf{r} + (1-\beta)\, r_0$, where $r_0 = \begin{pmatrix} 1/N \\ 1/N \\ \dots \\ 1/N \end{pmatrix}$
- For P-PageRank
  - Replace r_0 with $r_0 = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix}$

qth webpage

Mining Graph/Network Data

- Introduction to Graph/Network Data
- PageRank
- Personalized PageRank
- Summary

Summary

- Ranking on Graph / Network
  - PageRank
  - Personalized PageRank