

CS247: ADVANCED DATA MINING

Graph and Network: Random Walk

Instructor: Yizhou Sun


yzsun@cs.ucla.edu

May 4, 2020

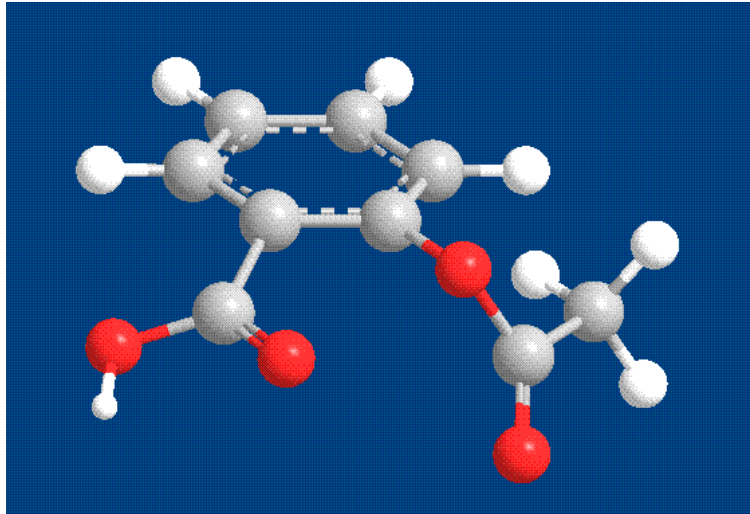
Methods to Learn

	Vector Data	Text Data	Graph & Network	Recommender Systems
Classification	Naïve Bayes; Logistic Regression; NN		Label Propagation	
Clustering	K-means; kernel k-means; Mixture Models	PLSA; LDA	Spectral Clustering	Matrix Factorization
Prediction	NN			Collaborative Filtering; Factorization machine; Hybrid CF; Recommendation with graph regularization
Ranking			PageRank	
Similarity Search			P-PageRank	
Representation Learning		Word embedding	Network embedding	Deep collaborative learning

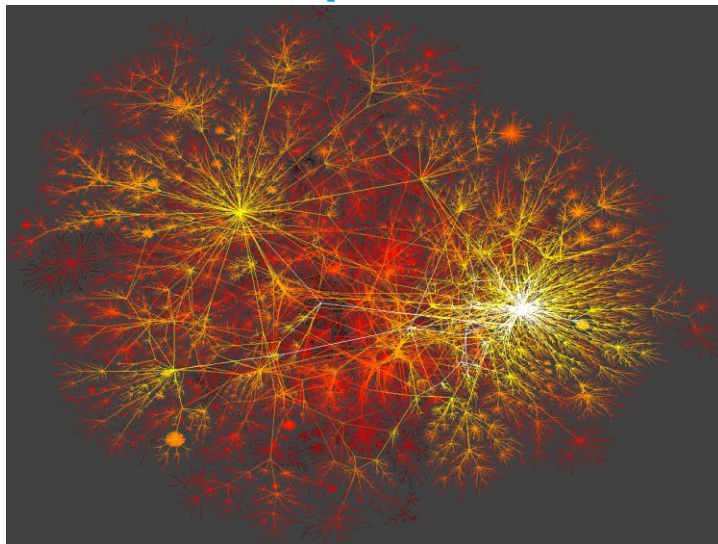
Random Walk on Graphs

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

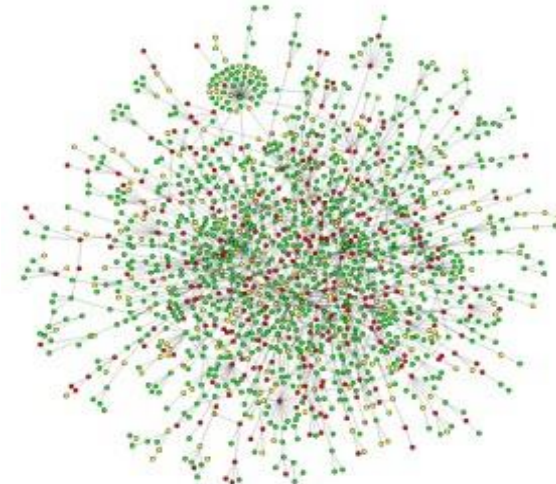
Graph, Graph, Everywhere



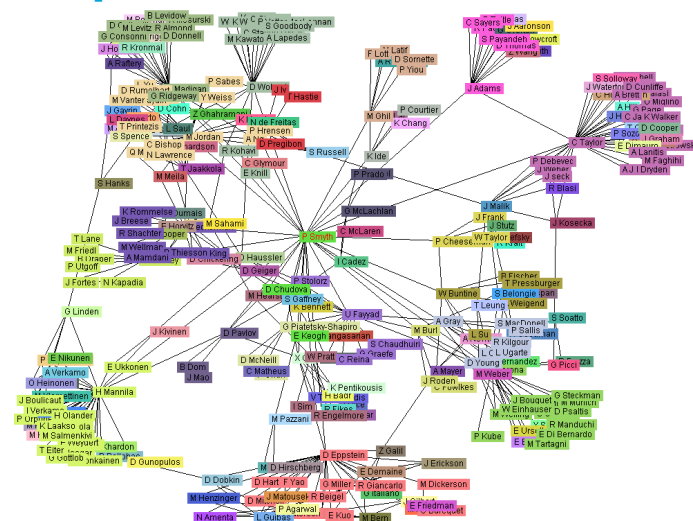
Aspirin



Internet



Yeast protein interaction network



Co-author network

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

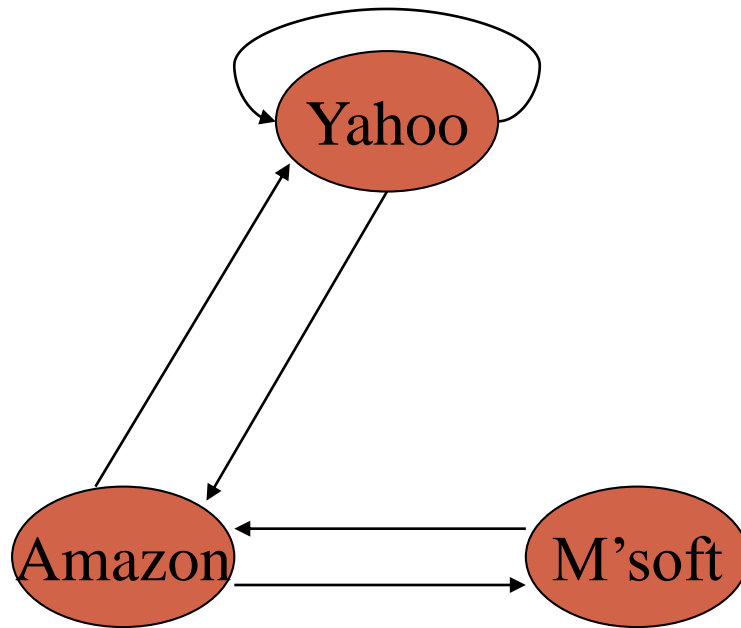
Why Graph Mining?

- Graphs are ubiquitous
 - Chemical compounds (Cheminformatics)
 - Protein structures, biological pathways/networks (Bioinformatics)
 - 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 vs. unweighted, homogeneous vs. heterogeneous
- Complexity of algorithms: many problems are of high complexity

Representation of a Graph

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


Example



	y	a	m
y	1	1	0
a	1	0	1
m	0	1	0

Adjacency matrix A

Random Walk on Graphs

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

The History of PageRank

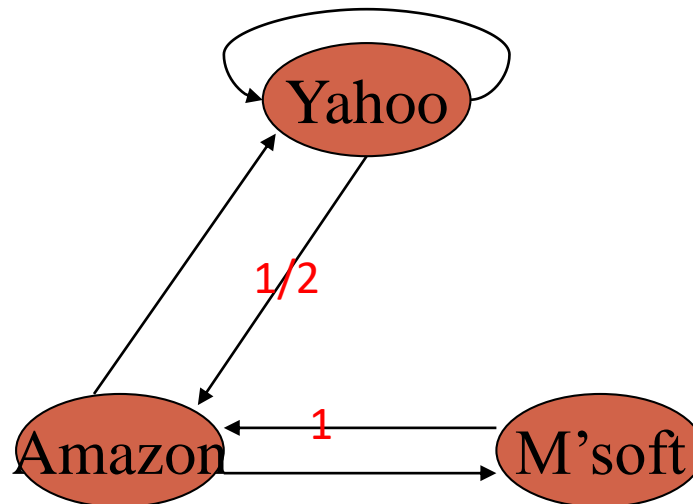
- PageRank was developed by Larry Page (hence the name *Page-Rank*) and Sergey Brin.
- It is first as part of a research project about a new kind of search engine. That project started in 1995 and led to a functional prototype in 1998.
- Shortly after, Page and Brin founded Google.

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?
 - Higher ranked inlink should play a more important role
 - 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

- N by N Matrix **M**
- Suppose page j has n outlinks

	y	a	m	
y	1	1	0	$\frac{1}{2}$ $\frac{1}{2}$ 0
a	1	0	1	$\frac{1}{2}$ 0 1
m	0	1	0	0 $\frac{1}{2}$ 0



- If $j \rightarrow i$, then $M_{ij} = 1/n$; Else $M_{ij} = 0$
- Or $M = (D^{-1}A)^T$, where **D** is a diagonal matrix and $D_{ii} = \sum_j A_{ij}$
- **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**
 - $|\mathbf{r}| = 1$ (i.e., $r_1 + r_2 + \dots + r_N = 1$)

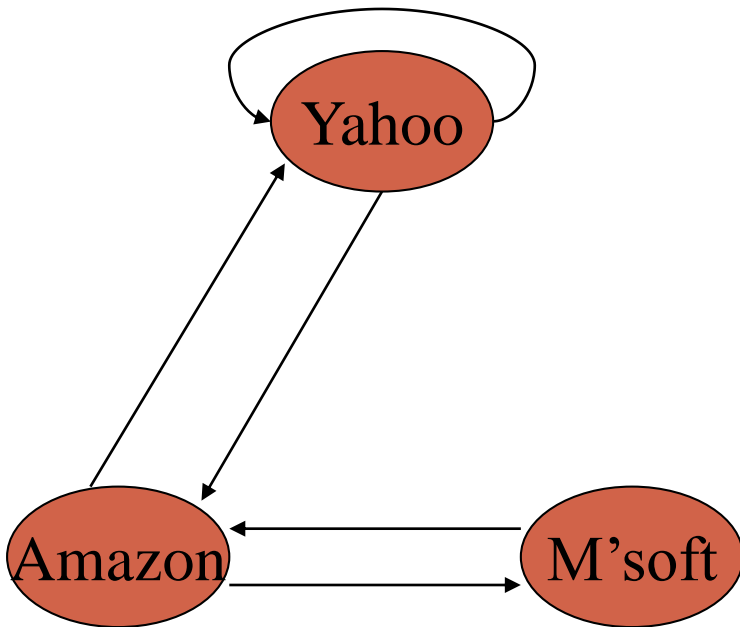
Eigenvector formulation

- The flow equations can be written

$$r = Mr$$

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

Example



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

$$a = y/2 + m$$

$$m = a/2$$

	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

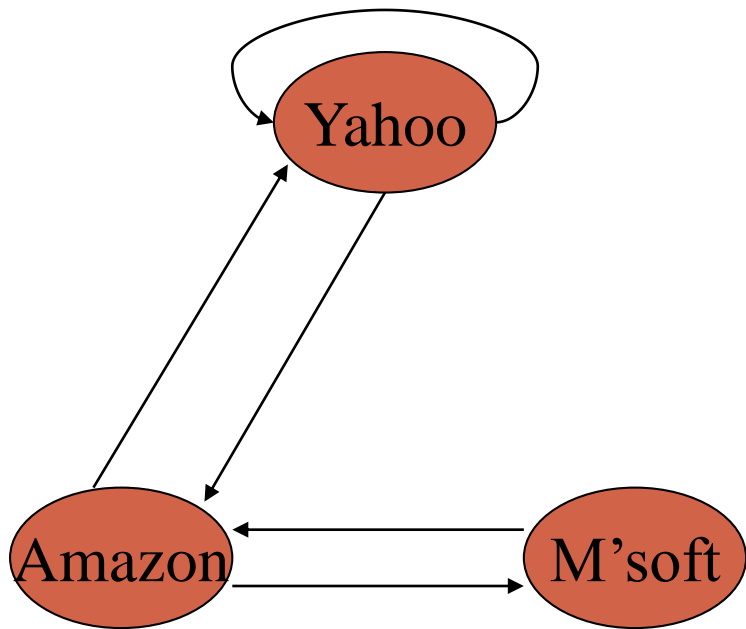
$$\mathbf{r} = \mathbf{M} * \mathbf{r}$$

y	1/2	1/2	0	y
a	1/2	0	1	a
m	0	1/2	0	m

Power Iteration method

- Simple iterative scheme
- Suppose there are N web pages
 - Initialize: $\mathbf{r}^0 = [1/N, \dots, 1/N]^T$
 - 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



	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

y	=	1/3	1/3	5/12	3/8		2/5
a		1/3	1/2	1/3	11/24	...	2/5
m		1/3	1/6	1/4	1/6		1/5
		r_0	r_1	r_2	r_3	...	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 \mathbf{r} satisfies $\mathbf{r} = \mathbf{M}\mathbf{r}$
 - So it is a stationary distribution for the random surfer

Existence and Uniqueness

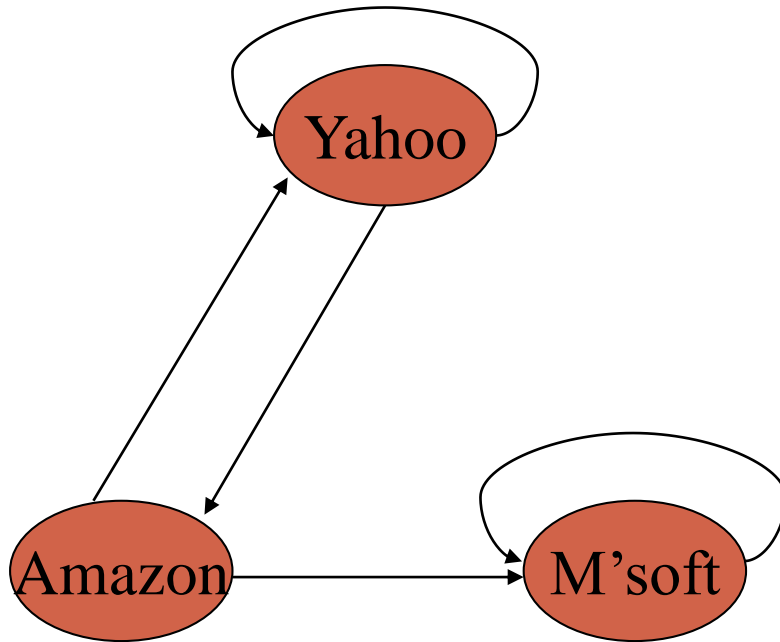
A central result from the theory of random walks (aka Markov processes):

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!

Microsoft becomes a spider trap



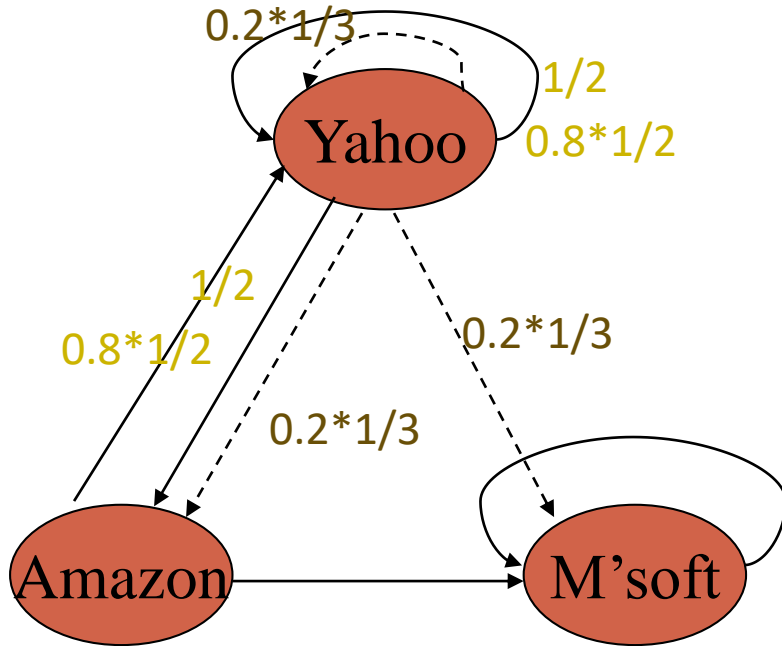
	y	a	m
y	1/2	1/2	0
a	1/2	0	0
m	0	1/2	1

y	=	1/3	1/3	1/4	5/24		0
a		1/3	1/6	1/6	1/8	...	0
m		1/3	1/2	7/12	2/3		1

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 ($\beta = 0.8$)



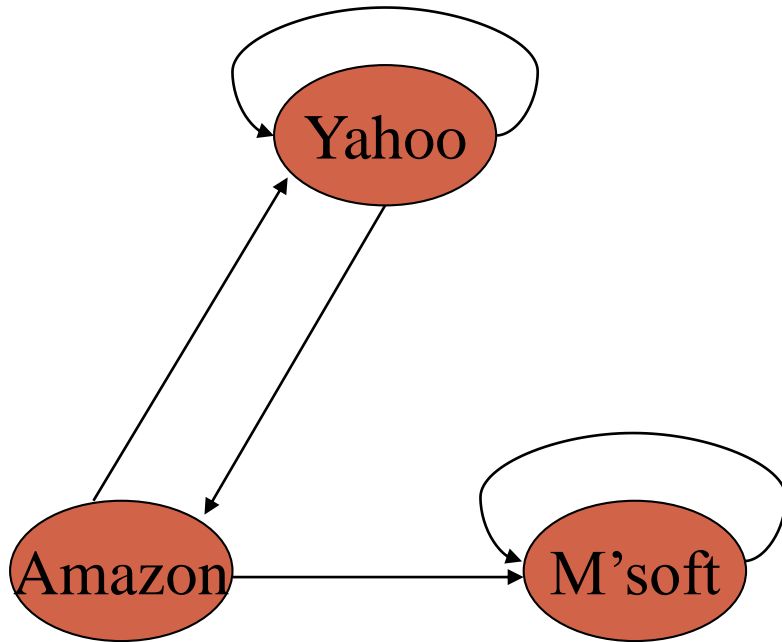
$$\begin{array}{c} y \\ a \\ m \end{array} \begin{array}{c} y \\ 1/2 \\ 1/2 \\ 0 \end{array} \quad 0.8 * \begin{array}{c} y \\ 1/2 \\ 1/2 \\ 0 \end{array} \quad + \quad 0.2 * \begin{array}{c} y \\ 1/3 \\ 1/3 \\ 1/3 \end{array}$$

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

-----> : teleport links from "Yahoo"

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

Random teleports ($\beta = 0.8$)



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

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

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

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 $j \rightarrow i$ 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

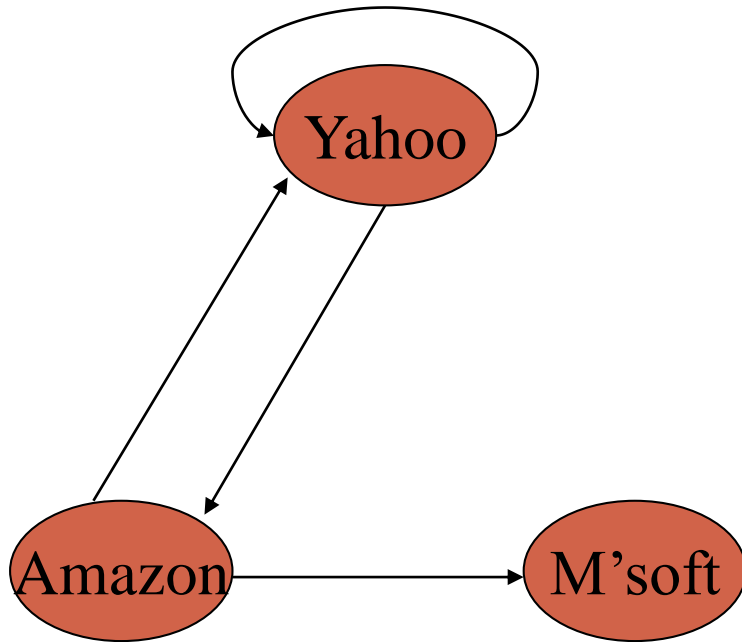
PageRank

- Construct the N-by-N matrix **B** as follows
 - $B_{ij} = \beta M_{ij} + (1-\beta)/N$
- Verify that **B** is a stochastic matrix
- The **page rank vector** **r** is the principal eigenvector of this matrix
 - satisfying $\mathbf{r} = \mathbf{B}\mathbf{r}$
- 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



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

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

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

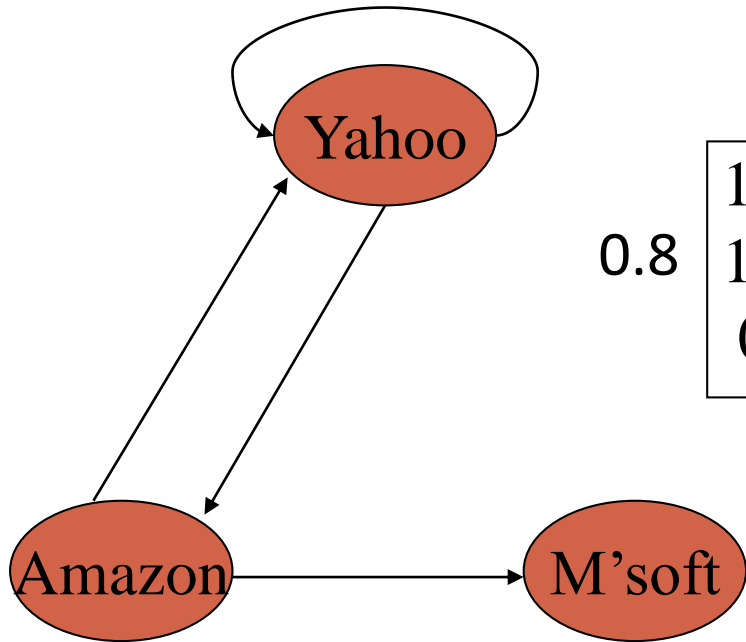
$$\begin{matrix} y \\ a \\ m \end{matrix} = \begin{matrix} 1/3 & 1/3 \\ 1/3 & 0.2 \\ 1/3 & 0.2 \end{matrix} \begin{matrix} 0 \\ \dots & 0 \\ 0 \end{matrix}$$

↓
Non-stochastic!

Dealing with dead-ends

- Method 1: Teleport
 - Follow random teleport links with probability 1.0 from dead-ends
 - Adjust matrix accordingly
- Method 2: Prune and propagate
 - Preprocess the graph to eliminate dead-ends
 - Might require multiple passes
 - Compute pagerank on reduced graph
 - Approximate values for deadends by propagating values from reduced graph

Dealing dead end: teleport



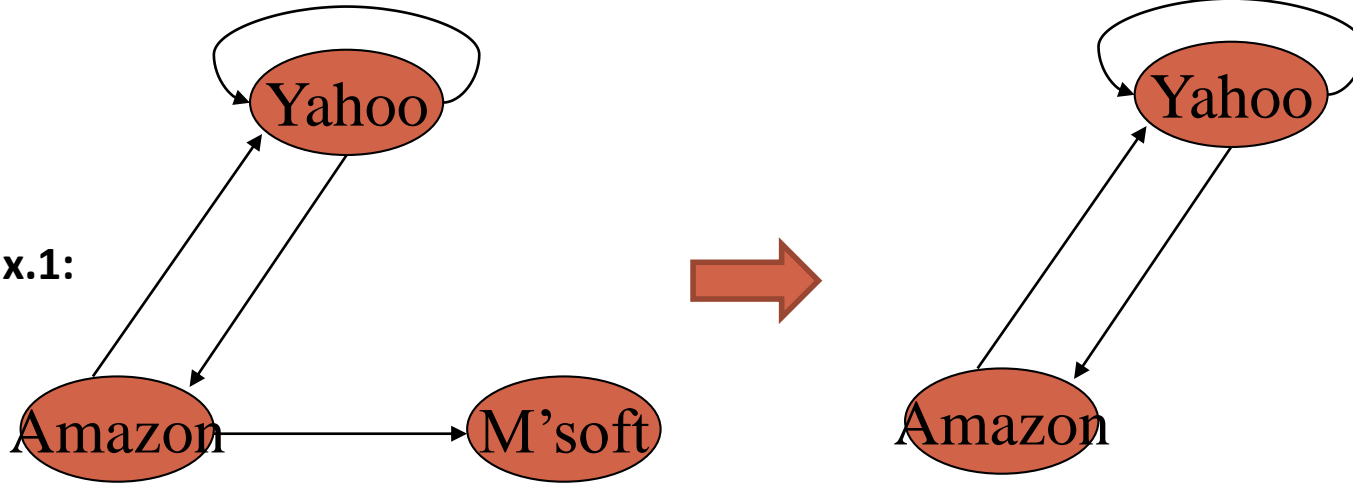
$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 0.2*1/3 & 0.2*1/3 & 1*1/3 \\ 0.2*1/3 & 0.2*1/3 & 1*1/3 \\ 0.2*1/3 & 0.2*1/3 & 1*1/3 \end{bmatrix}$$

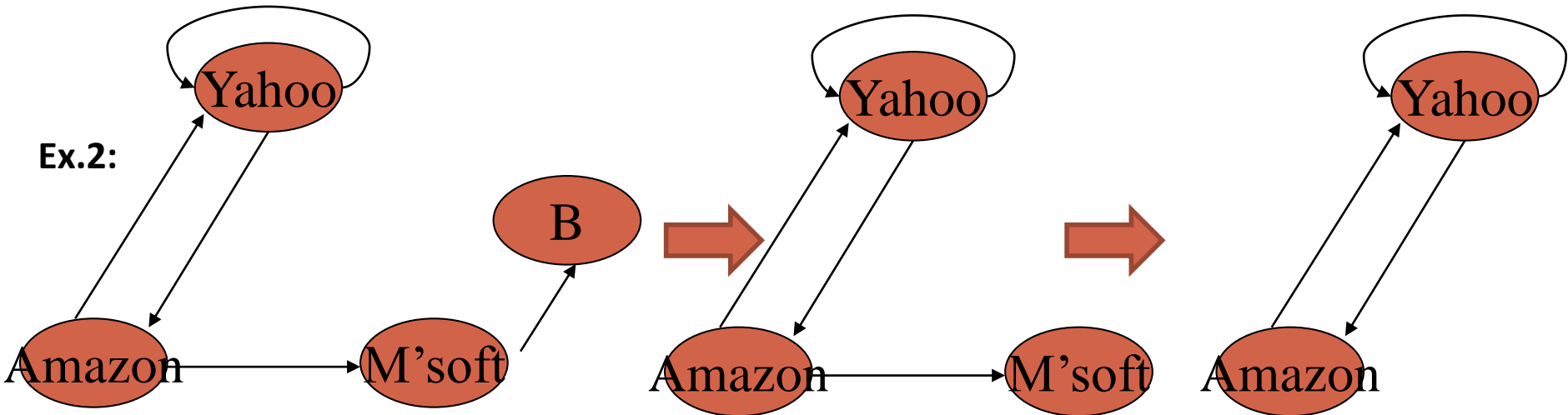
$$\begin{matrix} y \\ a \\ m \end{matrix} \begin{bmatrix} 7/15 & 7/15 & 1/3 \\ 7/15 & 1/15 & 1/3 \\ 1/15 & 7/15 & 1/3 \end{bmatrix}$$

Dealing dead end: reduce graph

Ex.1:



Ex.2:



Computing PageRank

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

Rearranging the equation

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

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

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

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

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

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

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

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 $(1-\beta)/N$
- \mathbf{M} is a sparse matrix!
 - 10 links per node, approx $10N$ entries
- So in each iteration, we need to:
 - Compute $\mathbf{r}^{\text{new}} = \beta \mathbf{M} \mathbf{r}^{\text{old}}$
 - 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: $\mathbf{r}^{\text{old}} = [1/N]_N$
- 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 $|\mathbf{r}^{\text{new}} - \mathbf{r}^{\text{old}}|$ and stop if it is below threshold
 - Need to read in both vectors into memory

Random Walk on Graphs

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

Personalized PageRank

- Query-dependent Ranking
 - For a query webpage u , which webpages are most important to u ?
 - We need a measure $s(u,v)$
 - The relative important webpages to different queries would be different

Calculation of P-PageRank

- Recall PageRank calculation:

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


- $\mathbf{r} = \beta \mathbf{M} \mathbf{r} + (1-\beta) \mathbf{r}_0$, where $\mathbf{r}_0 = \begin{pmatrix} 1/N \\ 1/N \\ \dots \\ 1/N \end{pmatrix}$

- For P-PageRank, $s(u,v) = \mathbf{r}_u(v)$, where $\mathbf{r}_u = \beta \mathbf{M} \mathbf{r}_u + (1-\beta) \mathbf{r}_0$

by replacing \mathbf{r}_0 with $\mathbf{r}_0 = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix}$ ← uth webpage

- Teleport to webpage u

Random Walk on Graphs

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

Summary

- Random walk on graph
 - PageRank for ranking
 - P-PageRank for similarity search

Quiz

- Prove $r' = Mr$ is still sum up to one, where M is the column stochastic matrix, and r is a non-negative vector summing up to 1