Introduction to

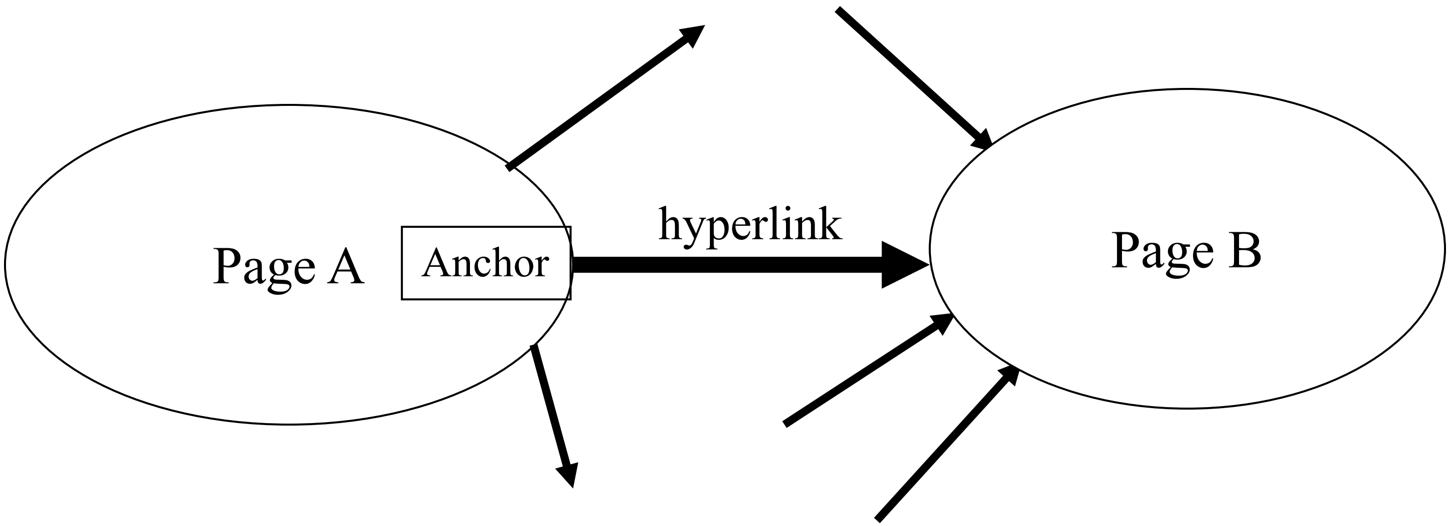
Information Retrieval

Lecture 18: Link analysis

Today's lecture

- Anchor text
- Link analysis for ranking
  - Pagerank and variants

The Web as a Directed Graph



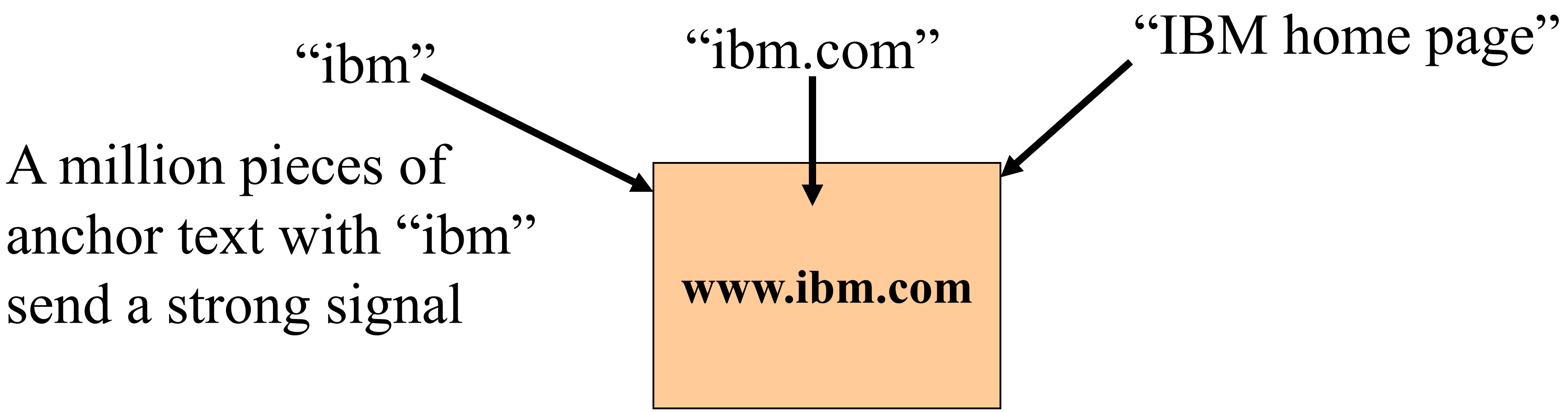
Assumption 1: A hyperlink between pages denotes author perceived relevance (quality signal)

Assumption 2: The text in the anchor of the hyperlink describes the target page (textual context)

Anchor Text

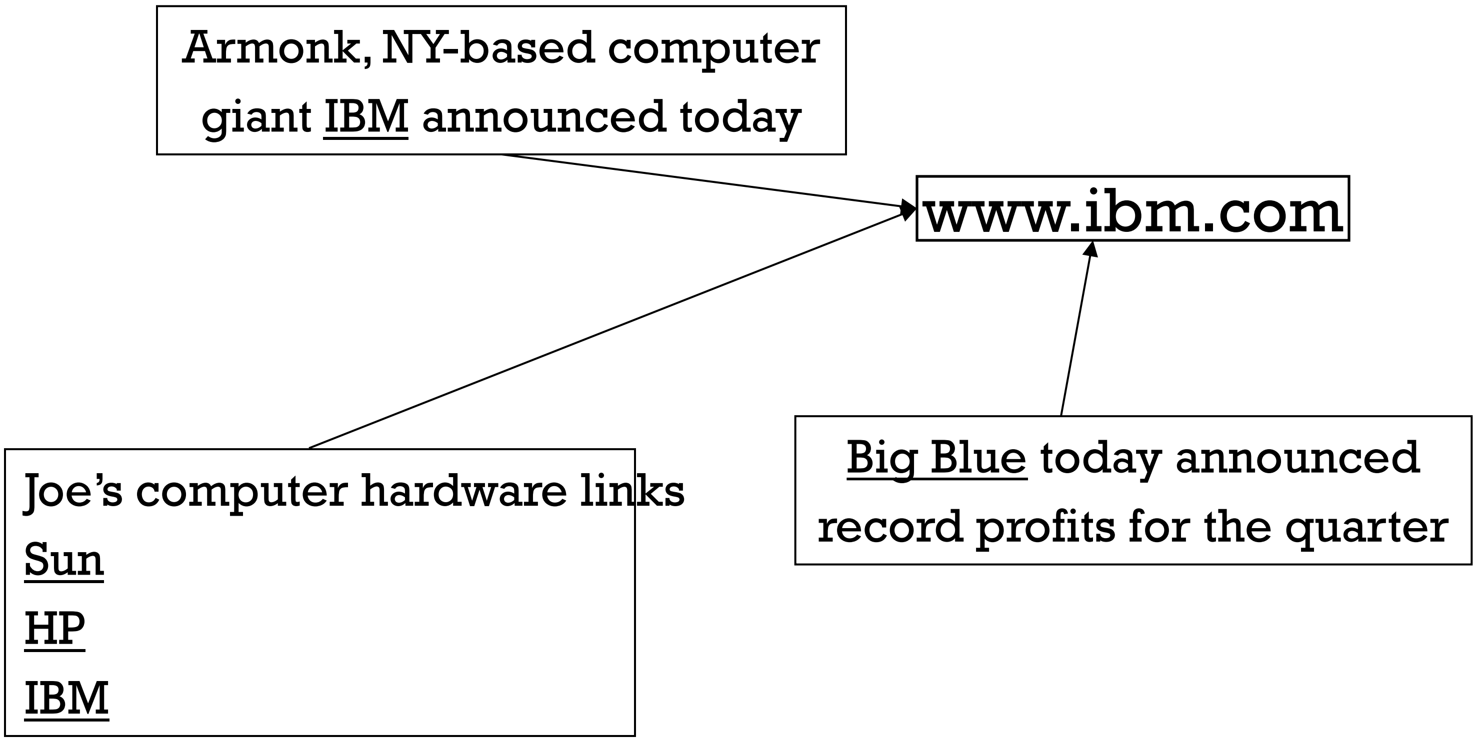
WWW Worm - McBryan [Mcbr94]

- For ***ibm*** how to distinguish between:
  - IBM's home page (mostly graphical)
  - IBM's copyright page (high term freq. for 'ibm')
  - Rival's spam page (arbitrarily high term freq.)



Indexing anchor text

- When indexing a document *D*, include anchor text from links pointing to *D*.



Indexing anchor text

- Can sometimes have unexpected side effects - *e.g.,* ***evil empire***.
- Can score anchor text with weight depending on the authority of the anchor page's website
  - E.g., if we were to assume that content from cnn.com or yahoo.com is authoritative, then trust the anchor text from them

Anchor Text

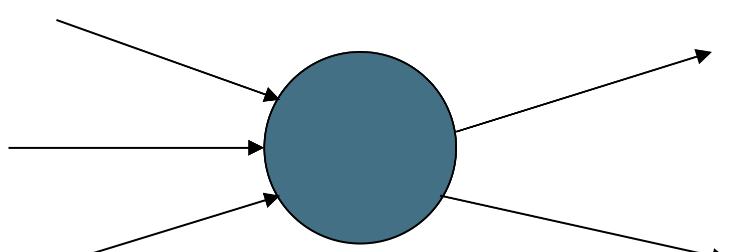
- Other applications
  - Weighting/filtering links in the graph
  - Generating page descriptions from anchor text

Citation Analysis

- Citation frequency
- Co-citation coupling frequency
  - Cocitations with a given author measures “impact”
  - Cocitation analysis
- Bibliographic coupling frequency
  - Articles that co-cite the same articles are related
- Citation indexing
  - Who is this author cited by? (Garfield 1972)
- Pagerank preview: Pinsker and Narin ’60s

Query-independent ordering

- First generation: using link counts as simple measures of popularity.
- Two basic suggestions:
  - Undirected popularity:
    - Each page gets a score = the number of in-links plus the number of out-links (3+2=5).
  - Directed popularity:
    - Score of a page = number of its in-links (3).



Query processing

- First retrieve all pages meeting the text query (say ***venture capital***).
- Order these by their link popularity (either variant on the previous slide).
- More nuanced – use link counts as a measure of static goodness (Lecture 7), combined with text match score

Spamming simple popularity

- *Exercise*: How do you spam each of the following heuristics so your page gets a high score?

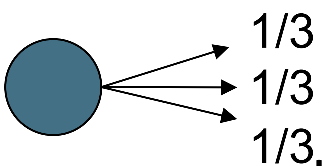
1. Each page gets a static score = the number of in-links plus the number of out-links.
2. Static score of a page = number of its in-links.

Ideas of Pagerank

- Inlinks as votes
  - www.stanford.edu has 23,400 inlinks
  - www.joe-schmoe.com has 1 inlink
- Web pages are not equally "important"
  - www.joe-schmoe.com → p1
  - vs. www.stanford.edu → p2
- Are all inlinks equal?
  - Recursive question!

Pagerank scoring

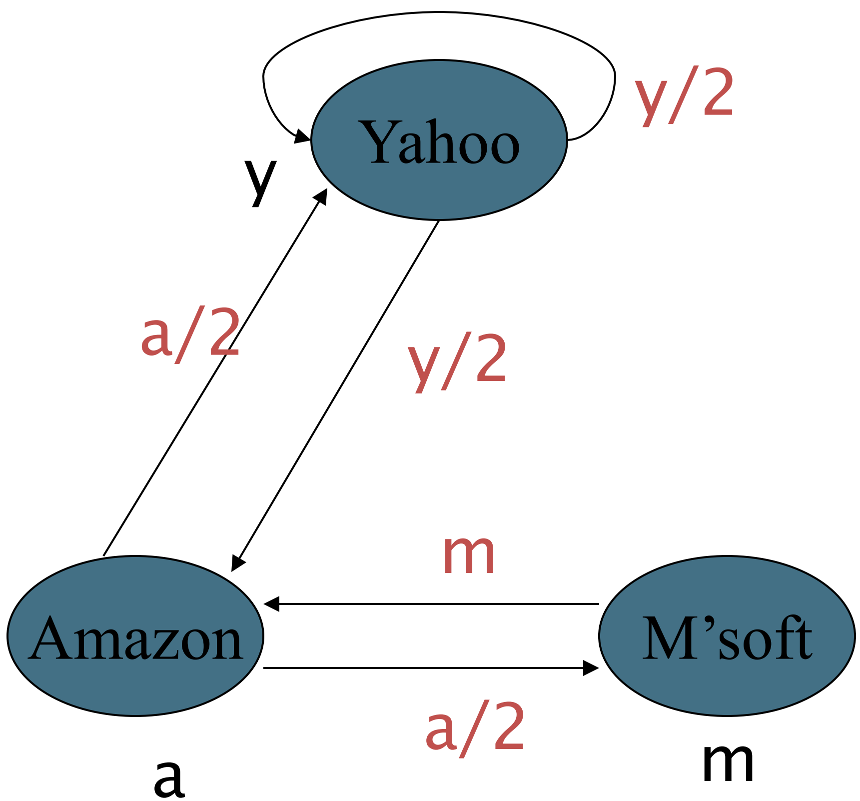
- Imagine a browser doing a *random walk* on web pages:
  - Start at a random page



  - At each step, go out of the current page along one of the links on that page, equiprobably
- "In the steady state" each page has a long-term visit rate - use this as the page's score.

Example – the Simple "Flow" Model

The web in 1839



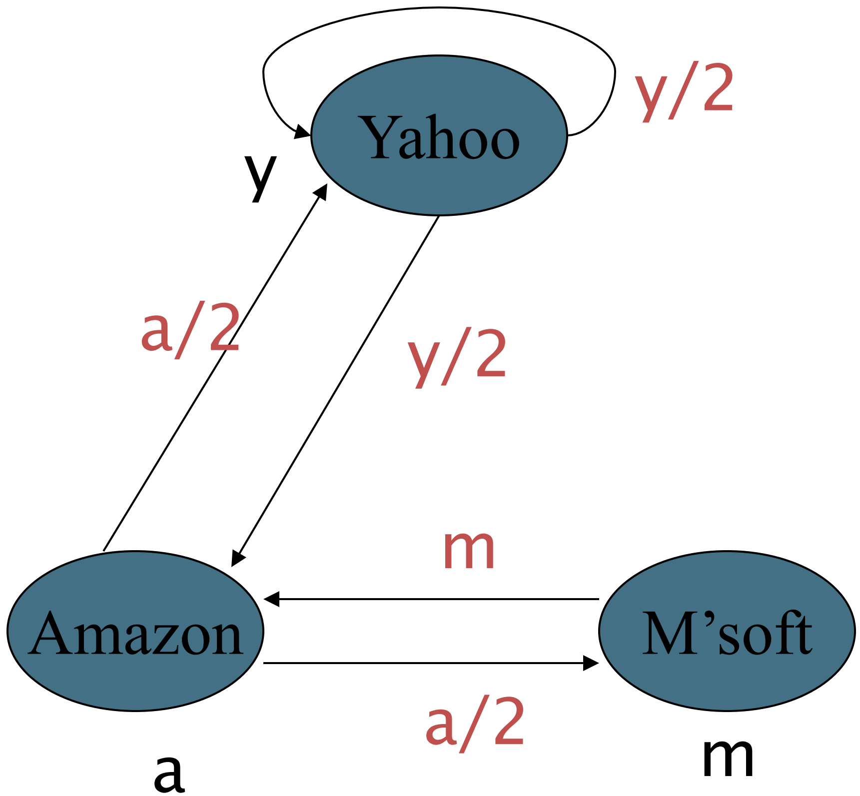
$$y = y/2 + a/2$$
$$a = y/2 + m$$
$$m = a/2$$

Solving the flow equations

- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint forces uniqueness
  - $y+a+m = 1$
  - $y = 2/5$, $a = 2/5$, $m = 1/5$
- Gaussian elimination method works for small examples, but we need a better method for large graphs

Example – the Simple "Flow" Model

The web in 1839



$$y_{new} = y_{old}/2 + a_{old}/2$$
$$a_{new} = y_{old}/2 + m_{old}$$
$$m_{new} = a_{old}/2$$

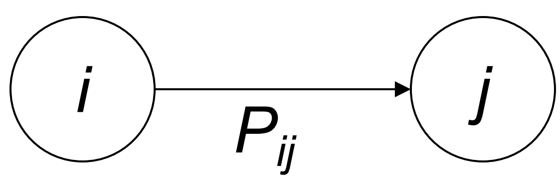
$$y = 1/3 \quad 1/3 \quad 5/12 \ldots 2/5$$
$$a = 1/3 \quad 1/2 \quad 1/3 \quad \ldots 2/5$$
$$m = 1/3 \quad 1/6 \quad 1/4 \quad \ldots 1/5$$

Matrix-based characterization of the computation is simpler and more useful for the general case.

Markov chains

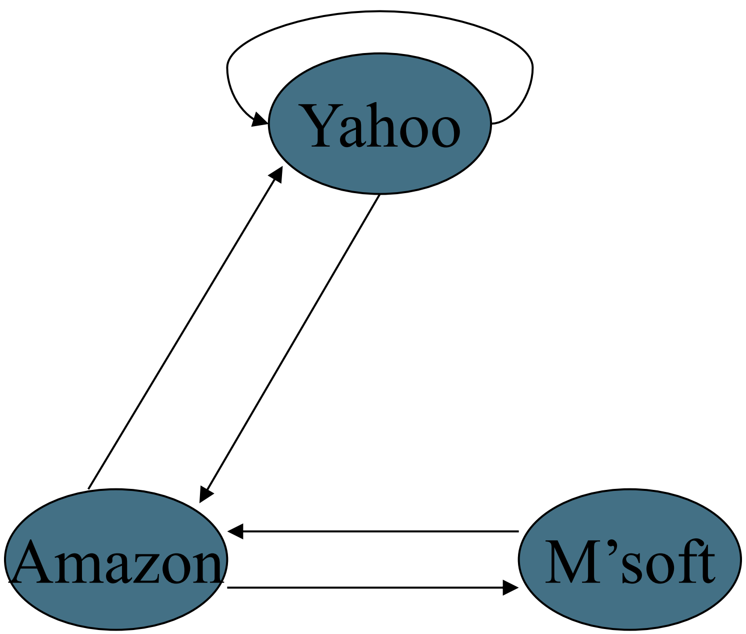
- A Markov chain consists of n states, plus an $n \times n$ transition probability matrix **P**.
- At each step, we are in exactly one of the states.
- For $1 \leq i,j \leq n$, the matrix entry P_{ij} tells us the probability of j being the next state, given we are currently in state i.

$P_{ii}>0$ is OK.



Markov chains

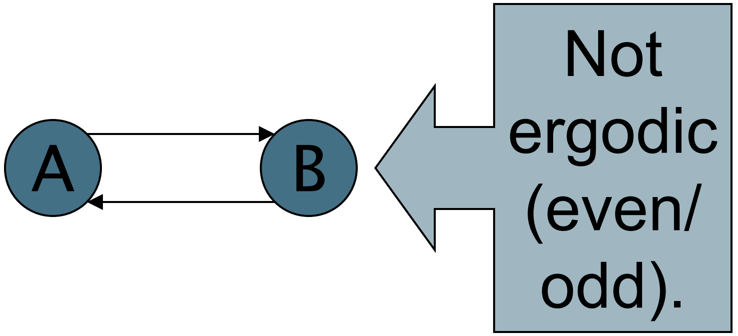
- Clearly, for all i, $\sum_{j=1}^{n} P_{ij} = 1.$
- Markov chains are abstractions of random walks.



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

Ergodic Markov chains

- A Markov chain is <u>ergodic</u> if
  - you have a path from any state to any other
  - For any start state, after a finite transient time T_0, <u>the probability of being in any state at a fixed time $T>T_0$ is nonzero.</u>



Ergodic Markov chains

- For any ergodic Markov chain, there is a unique long-term visit rate for each state.
  - *Steady-state probability distribution*.
- Over a long time-period, we visit each state in proportion to this rate.
- It doesn't matter where we start.

Probability vectors

- A probability (row) vector **x** = $(x_1, \ldots x_n)$ tells us where the walk is at any point.
- E.g., (000…1…000) means we're in state *i*.

  1 *i* *n*

More generally, the vector **x** = $(x_1, \ldots x_n)$ means the walk is in state *i* with probability x_i.

$$\sum_{i=1}^{n} x_i = 1.$$

Change in probability vector

- If the probability vector is $\mathbf{x} = (x_1, \ldots x_n)$ at this step, what is it at the next step?
- Recall that row *i* of the transition prob. Matrix **P** tells us where we go next from state *i*.
- So from **x**, our next state is distributed as **xP**.

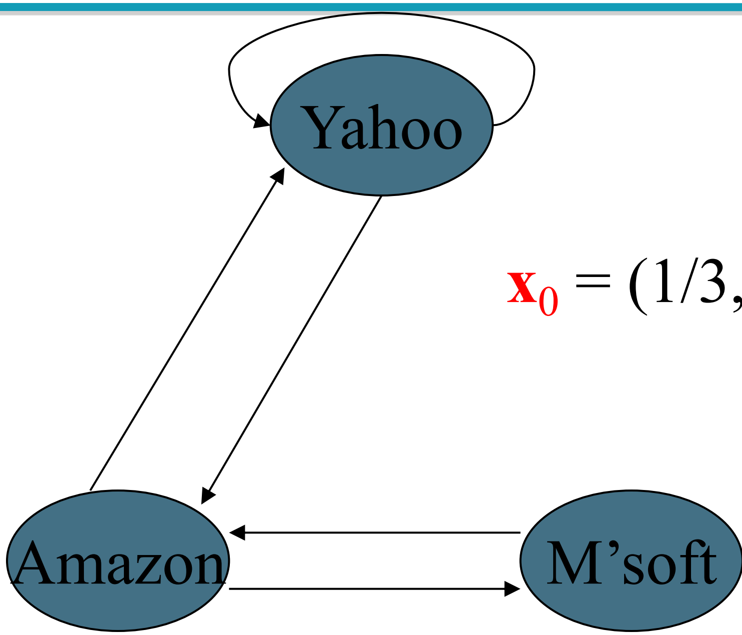
How do we compute this vector?

- Let **a** = $(a_1, \ldots a_n)$ denote the row vector of steady-state probabilities.
- If our current position is described by **a**, then the next step is distributed as **aP**.
- But **a** is the steady state, so **a**=**aP**.
- Solving this matrix equation gives us **a**.
  - So **a** is the (left) eigenvector for **P**.
  - (Corresponds to the "principal" eigenvector of **P** with the largest eigenvalue.)
  - Transition probability matrices always have largest eigenvalue 1.

One way of computing **a**

- Recall, regardless of where we start, we eventually reach the steady state **a**.
- Start with any distribution (say **x**=(*1/n*, *1/n*, …, *1/n*)).
- After one step, we're at **xP**;
- after two steps at $\mathbf{xP}^2$, then $\mathbf{xP}^3$ and so on.
- "Eventually" means for "large" k, $\mathbf{xP}^k = \mathbf{a}$.
- Algorithm: multiply **x** by increasing powers of **P** until the product looks stable.

Power Iteration Example



$\mathbf{x}_0 = (1/3,\ 1/3,\ 1/3)$

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

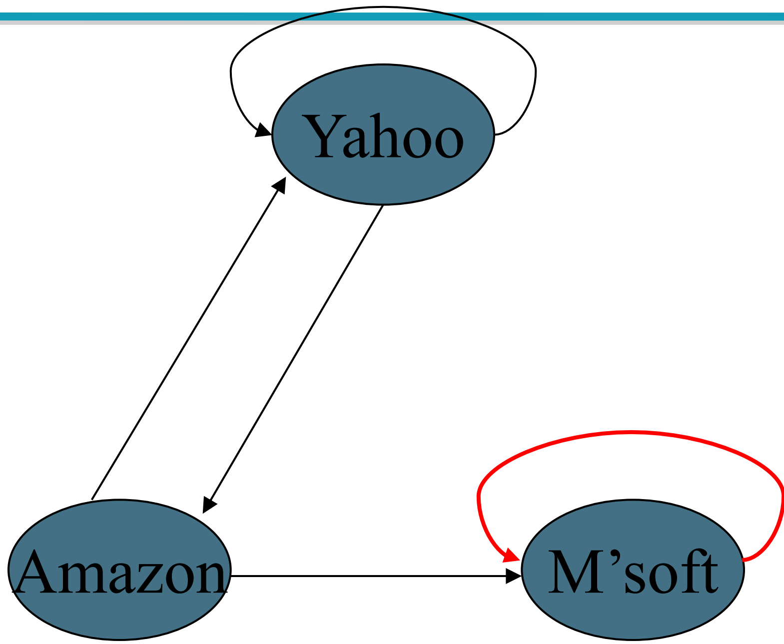
$$\begin{aligned} y_{new} &= y_{old}/2 + a_{old}/2 \\ a_{new} &= y_{old}/2 + m_{old} \\ m_{new} &= a_{old}/2 \end{aligned}$$

		$\mathbf{x}_0$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$		$\mathbf{x}_t$
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

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



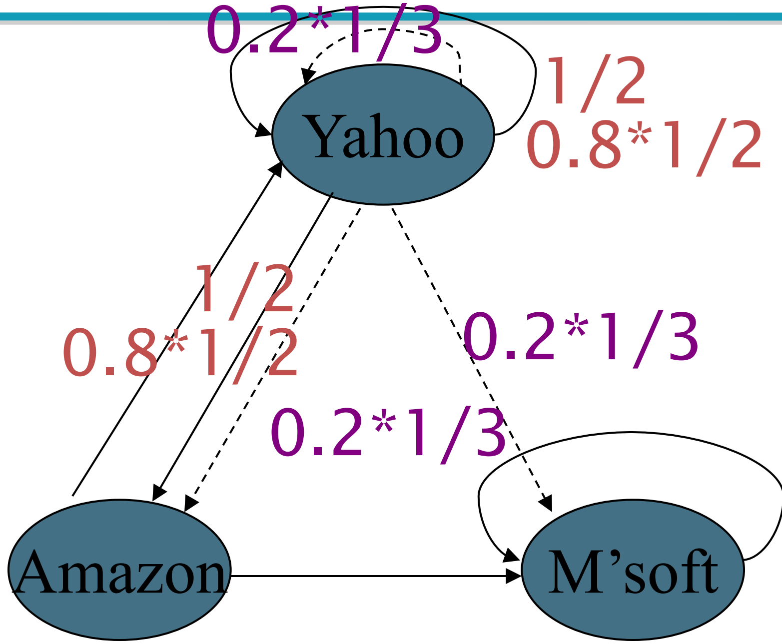
$$
\begin{array}{c|ccc|}
 & y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1/2 \\
m & 0 & 0 & 1 \\
\hline
\end{array}
$$

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

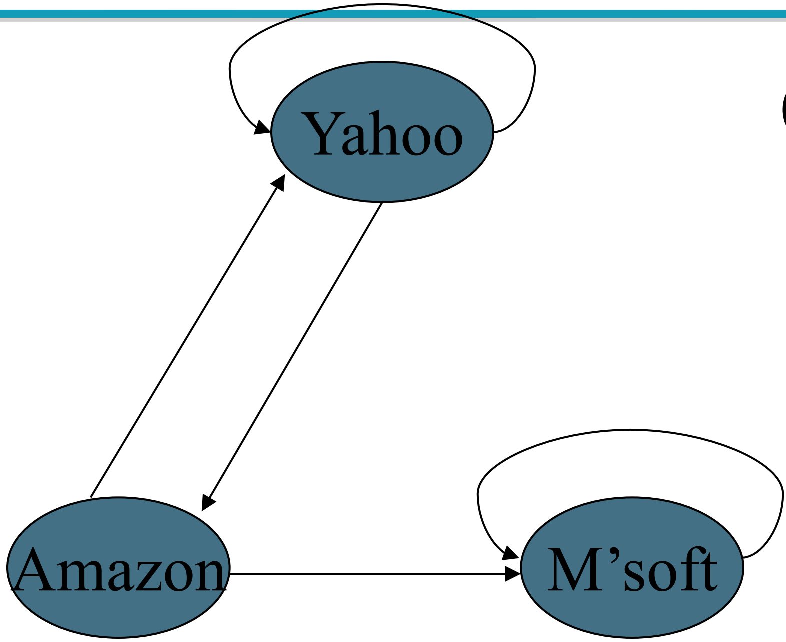
Random teleports ($\beta = 0.8$)



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

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

Random teleports ($\beta = 0.8$)



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

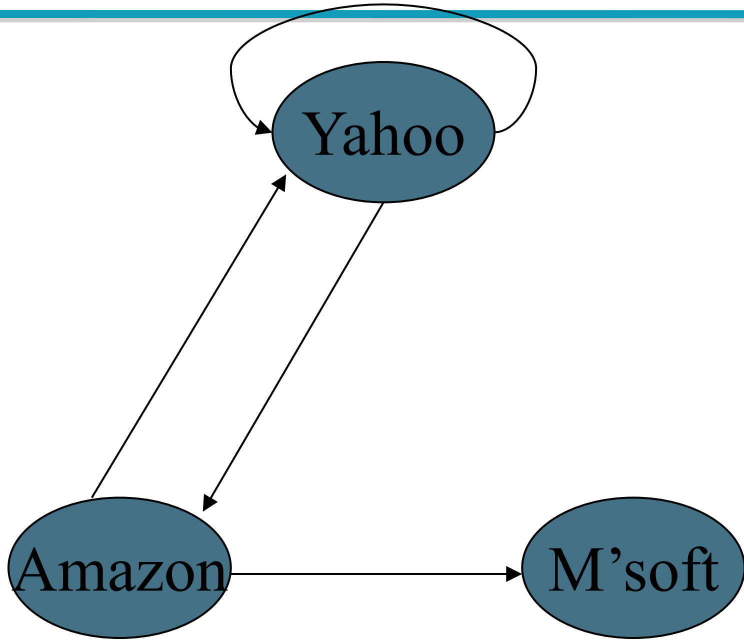
$$\begin{array}{c} y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 7/15 \\ 1/15 & 1/15 & 13/15 \end{bmatrix}$$

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 0.33 \\ 0.33 \\ 0.33 \end{array} \quad \begin{array}{c} 0.33 \\ 0.20 \\ 0.47 \end{array} \quad \begin{array}{c} 0.28 \\ 0.20 \\ 0.52 \end{array} \quad \begin{array}{c} 0.26 \\ 0.18 \\ 0.56 \end{array} \quad \ldots \quad \begin{array}{c} 7/33 \\ 5/33 \\ 7/11 \end{array}$$

Dead ends

- Pages with no outlinks are “dead ends” for the random surfer
  - Nowhere to go on next step
- Especially common for Web Search Engines
  - URLs that have not yet been crawled

Microsoft becomes a dead end



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

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

Non-stochastic!

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 0.33 \\ 0.33 \\ 0.33 \end{array} \quad \begin{array}{c} 0.33 \\ 0.20 \\ 0.20 \end{array} \quad \begin{array}{c} 0.262 \\ 0.182 \\ 0.129 \end{array} \quad \begin{array}{c} 0.216 \\ 0.143 \\ 0.111 \end{array} \quad \cdots \quad \begin{array}{c} 0 \\ 0 \\ 0 \end{array}$$

Dealing with dead-ends

- Teleport
  - Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly
    - How?
- (Suggested by Google) 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

Q: Why approximate values and why errors are insignificant?

Pagerank summary

- Preprocessing:
  - Given graph of links, build matrix **P**.
  - From it compute **a**.
    - **a** is the principle eigen vector of a matrix $\tilde{\mathbf{P}}$

$$\tilde{\mathbf{P}} = (1-\beta)\mathbf{P} + \beta\mathbf{T}, \qquad \mathbf{T}_{i,j} = \frac{1}{n}$$

  - The entry a_i is a number between 0 and 1: the pagerank of page *i*.
- Query processing:
  - Retrieve pages meeting query.
  - Rank them by their pagerank.
  - Order is query-*independent*.

The reality

- Pagerank is used in google, but is hardly the full story of ranking
  - Many sophisticated features are used
  - Some address specific query classes
  - Machine learned ranking (Lecture 15) heavily used
- Pagerank still very useful for things like crawl policy

Pagerank: Issues and Variants

- How realistic is the random surfer model?
  - (Does it matter?)
  - What if we modeled the back button?
  - Surfer behavior sharply skewed towards short paths
  - Search engines, bookmarks & directories make jumps non-random.
- Biased Surfer Models
  - Weight edge traversal probabilities based on match with topic/query (non-uniform edge selection)
  - Bias jumps to pages on topic (e.g., based on personal bookmarks & categories of interest)

Topic Specific Pagerank

- Goal – pagerank values that depend on query *topic*
- Conceptually, we use a random surfer who teleports, with say 10% probability, using the following rule:
  - Selects a topic (say, one of the 16 top level ODP categories) based on a query & user -specific distribution over the categories
  - Teleport to a page uniformly at random within the chosen topic
- Sounds hard to implement: can't compute PageRank at query time!

Topic Specific Pagerank

- **Offline**:Compute pagerank for *individual* topics
  - Query independent as before
  - Each page has multiple pagerank scores – one for each ODP category, with teleportation only to that category
- **Online**: Query context classified into (distribution of weights over) topics
  - Generate a dynamic pagerank score for each page – weighted sum of topic-specific pageranks

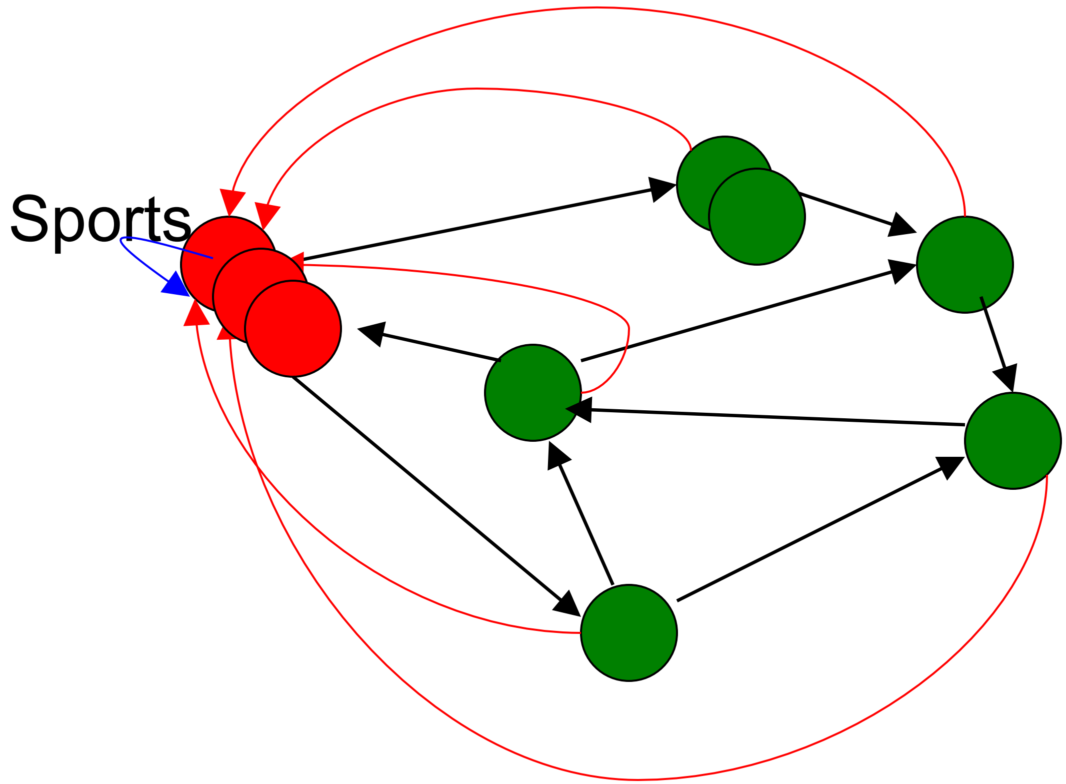
Influencing PageRank ("Personalization")

- Input:
  - Web graph W
  - Influence vector **v** over topics

    v : (page $\rightarrow$ degree of influence)
- Output:
  - Rank vector **r**: (page $\rightarrow$ page importance wrt **v**)
- **r** = PR(W , **v**)

Vector has one component for each topic

Non-uniform Teleportation



Teleport with 10% probability to a Sports page

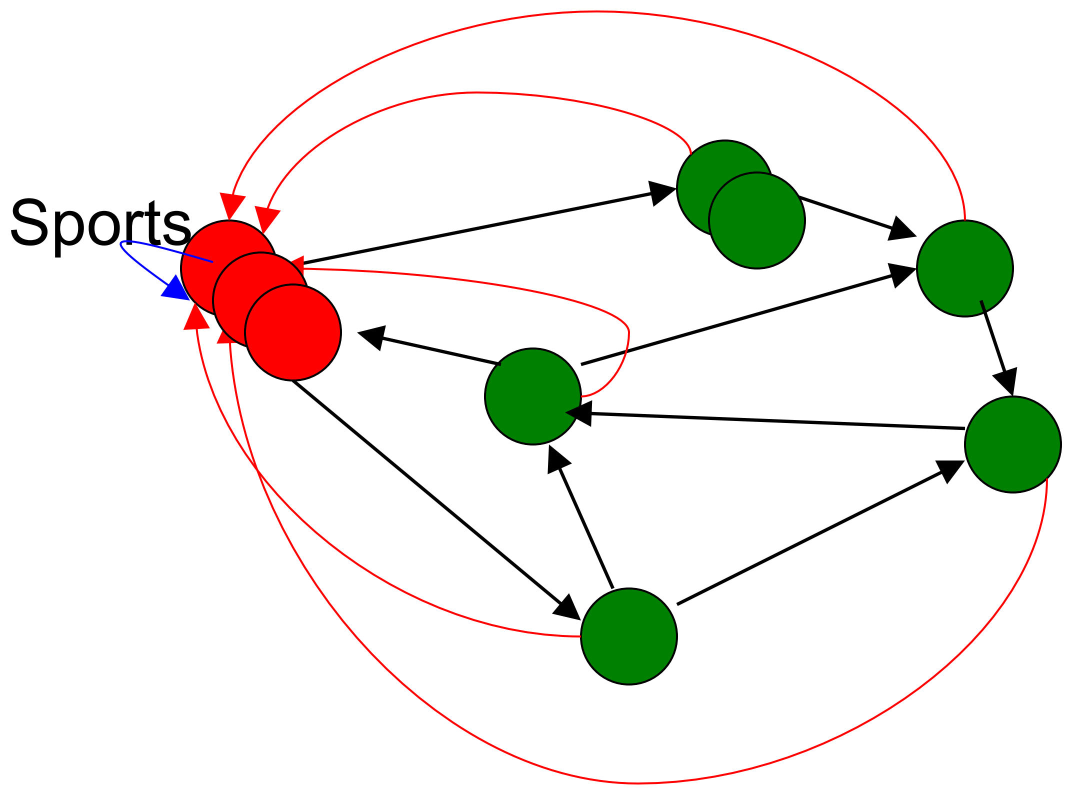
Interpretation of Composite Score

- Given a set of personalization vectors $\{\mathbf{v}_j\}$

$$\sum_j [w_j \cdot PR(W, \mathbf{v}_j)] = PR(W, \sum_j [w_j \cdot \mathbf{v}_j])$$

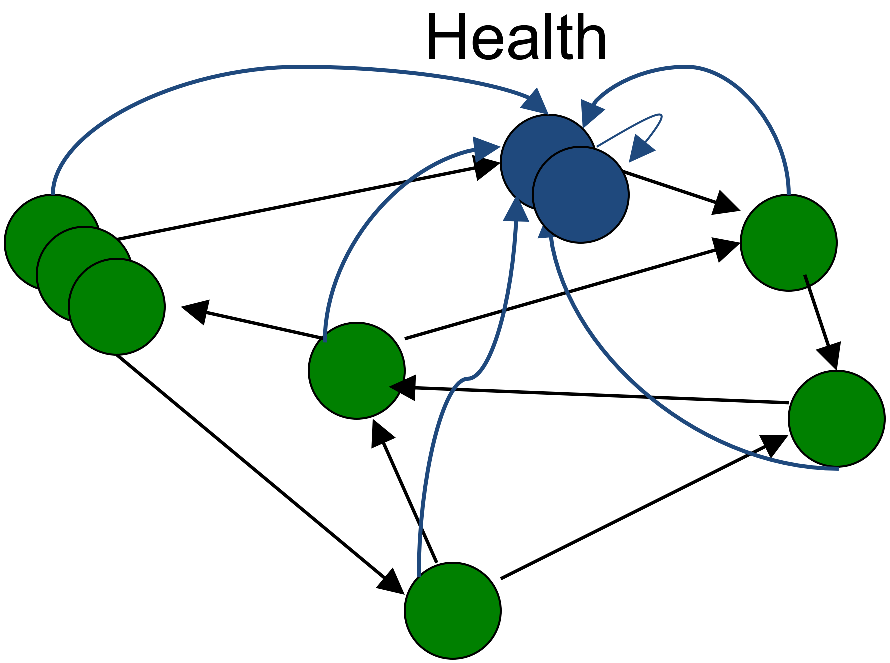
Given a user's preferences over topics, express as a combination of the "basis" vectors $\mathbf{v}_j$

Interpretation



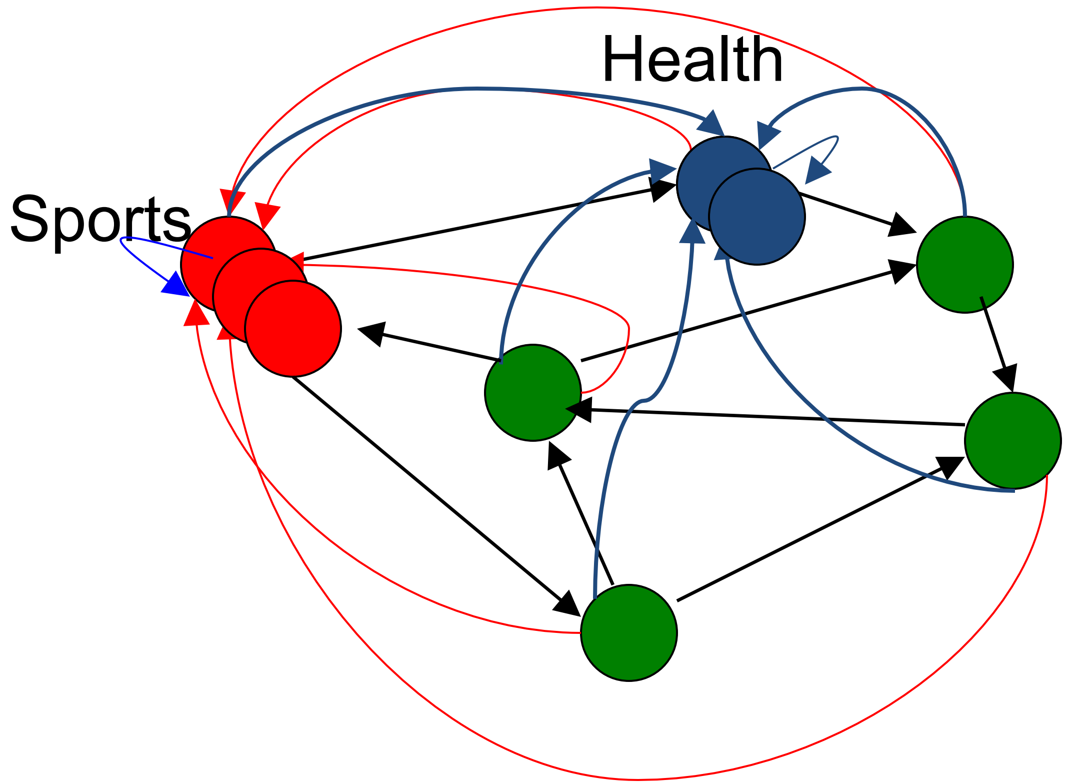
10% Sports teleportation

Interpretation



10% Health teleportation

Interpretation



pr = (0.9 PR_{sports} + 0.1 PR_{health}) gives you:
9% sports teleportation, 1% health teleportation

Resources

- IIR Chap 21
- http://www2004.org/proceedings/docs/1p309.pdf
- http://www2004.org/proceedings/docs/1p595.pdf
- http://www2003.org/cdrom/papers/refereed/p270/kamvar-270-xhtml/index.html
- http://www2003.org/cdrom/papers/refereed/p641/xhtml/p641-mccurley.html