Webgraph structure and PageRank

CS345a: Data Mining

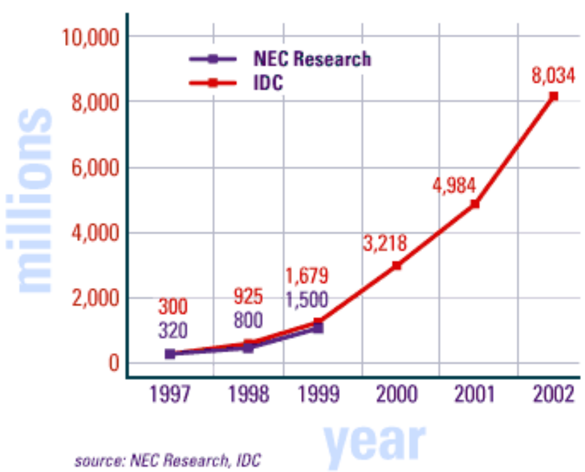
Jure Leskovec and Anand Rajaraman
Stanford University

Two More Datasets Available

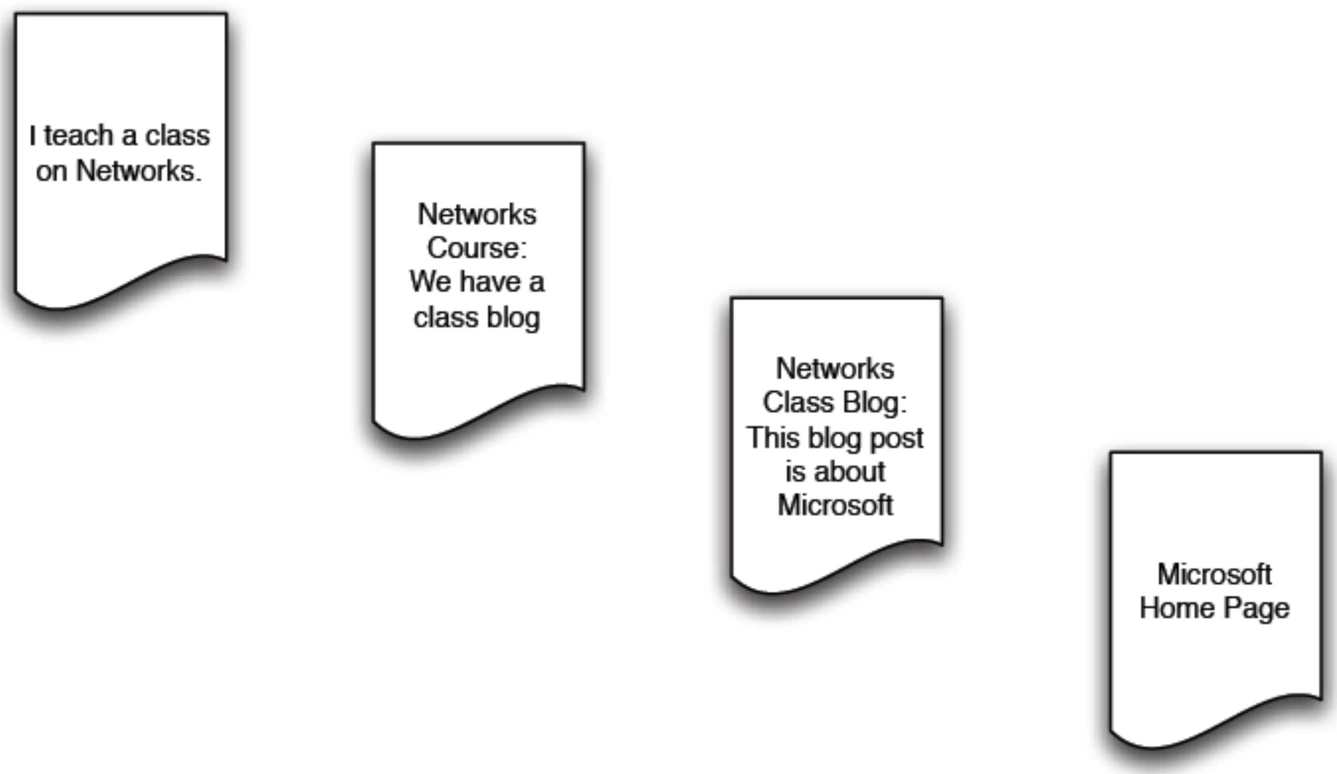
- TheFind.com
  - Large set of products (~6GB compressed)
  - For each product
    - Attributes
    - Related products
- Craigslist
  - About 3 weeks of data (~7.5GB compressed)
    - Text of posts, plus category metadata
    - e.g., match buyers and sellers

How big is the Web?

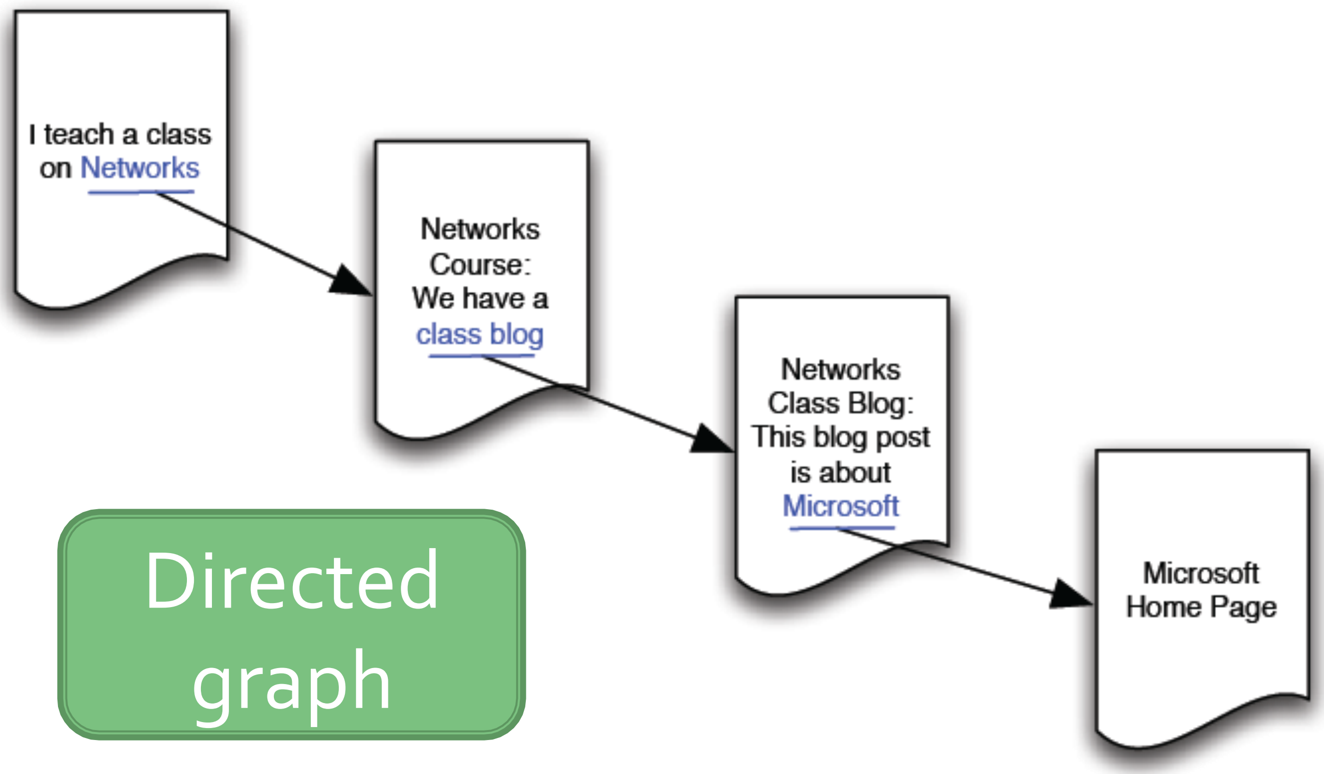
- How big is the Web?
  - Technically, infinite
  - Much duplication (30-40%)
  - Best estimate of "unique" static HTML pages comes from search engine claims
    - Google = 8 billion(?), Yahoo = 20 billion
- What is the structure of the Web? How is it organized?



Web as a Graph



Web as a Graph



Directed graph

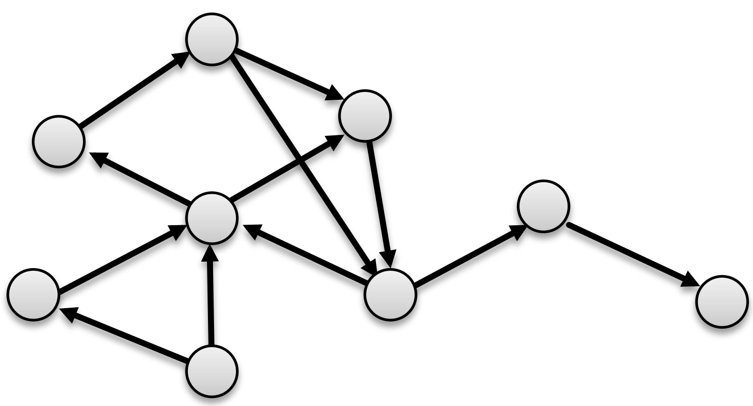
- In early days of the Web links were navigational
- Today many links are transactional

Directed graphs

- Two types of directed graphs:
  - DAG – directed acyclic graph:
    - Has no cycles: if u can reach v, then v can not reach u
  - Strongly connected:
    - Any node can reach any node via a directed path
- Any directed graph can be expressed in terms of these two types

Strongly connected component

- Strongly connected component (SCC) is a set of nodes S so that:
  - Every pair of nodes in S can reach each other
  - There is no larger set containing S with this property



Graph structure of the Web

- Take a large snapshot of the web and try to understand how it's SCCs "fit" as a DAG.

- Computational issues:
  - Say want to find SCC containing specific node v?
  - Observation:
    - Out(v) … nodes that can be reachable from v (BFS out)
    - SCC containing v:

      $= \mathrm{Out}(v, G) \cap \mathrm{In}(v, G)$

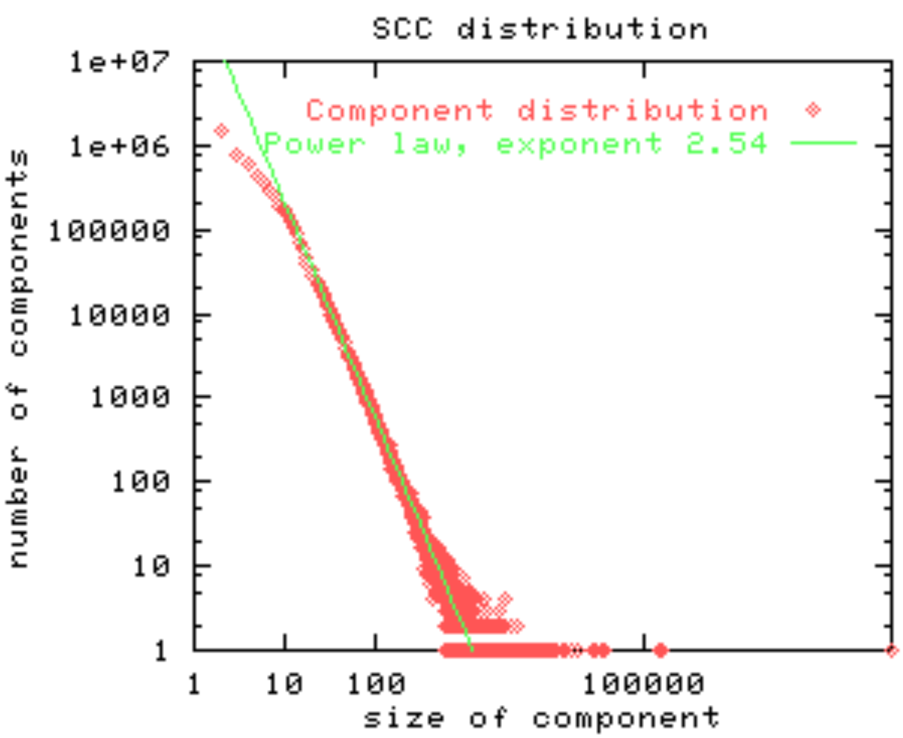
      $= \mathrm{Out}(v, G) \cap \mathrm{Out}(v, \overline{G})$

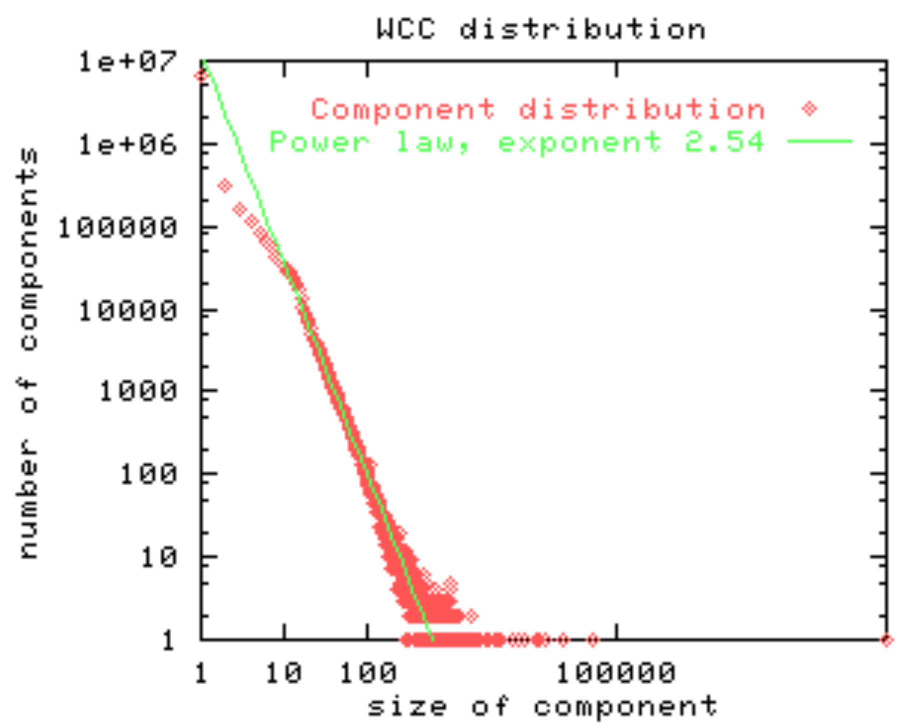
  where $\overline{G}$ is G with directions of all edge flipped

[Broder et al., 'oo]

Graph structure of the Web

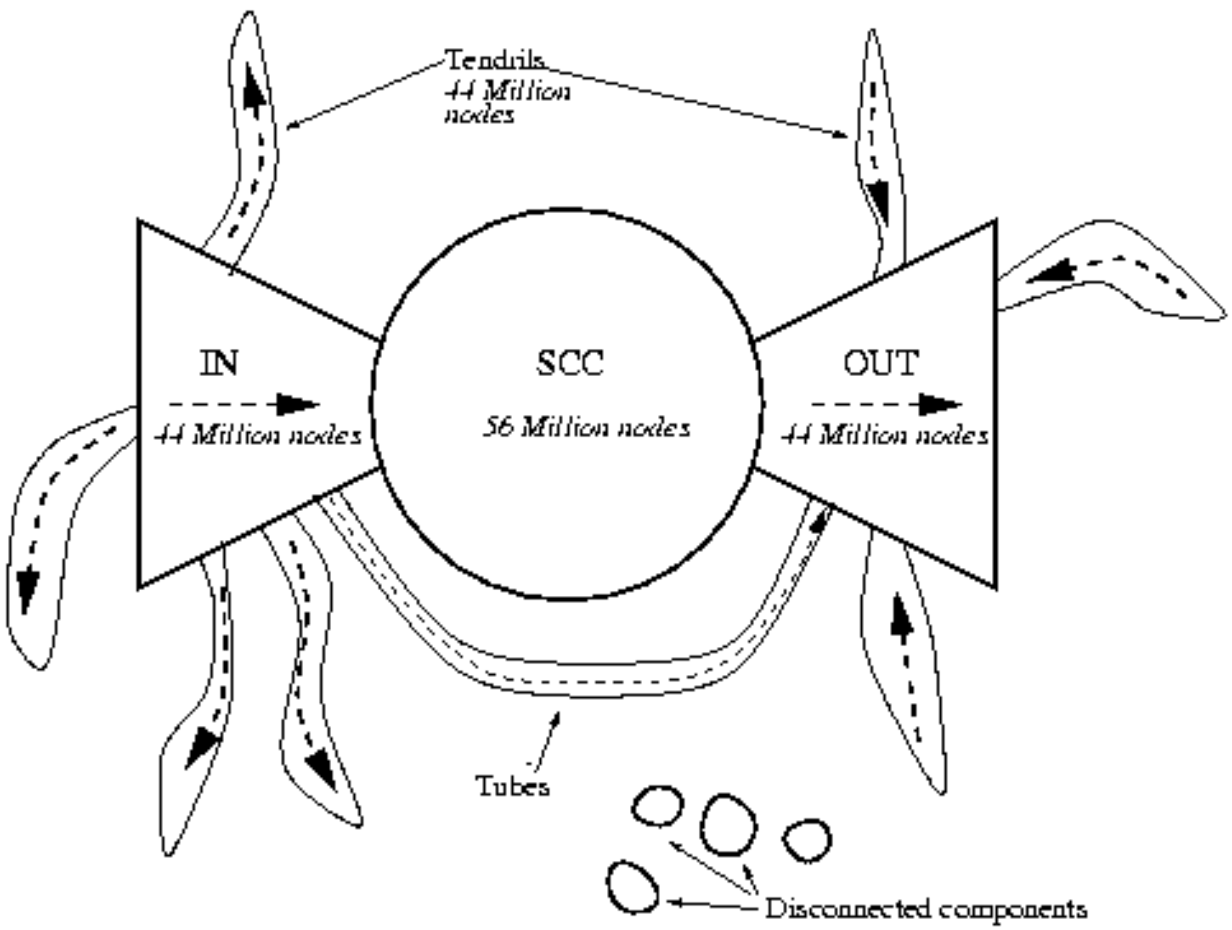
- There is a giant SCC
- Broder et al., 2000:
  - Giant weakly connected component: 90% of the nodes





[Broder et al., 'oo]

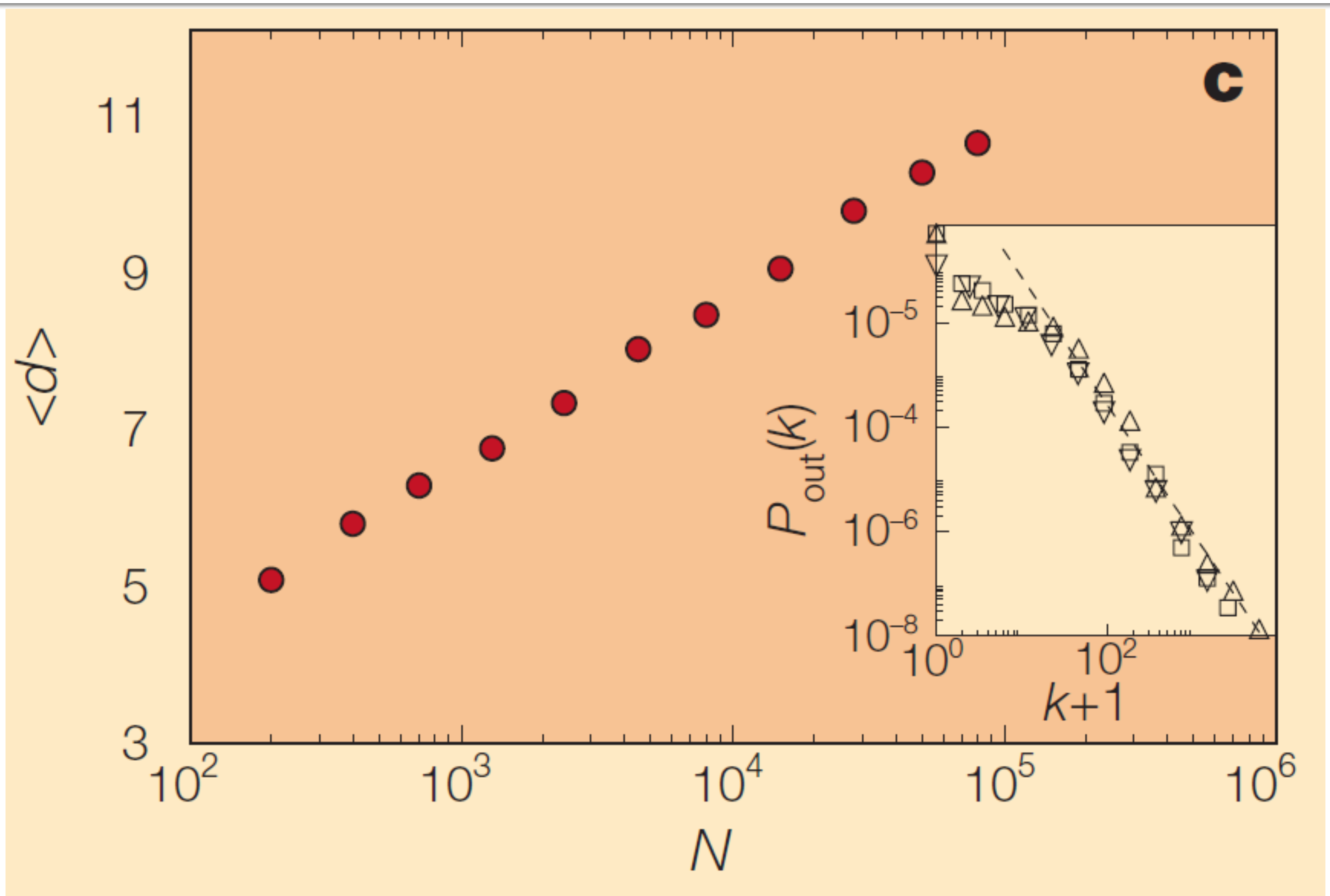
Bow-tie structure of the Web



- 250 million webpages, 1.5 billion links [Altavista]

[Albert et al., '99]

Diameter of the Web



- Diameter (average directed shortest path length) is 19 (in 1999)

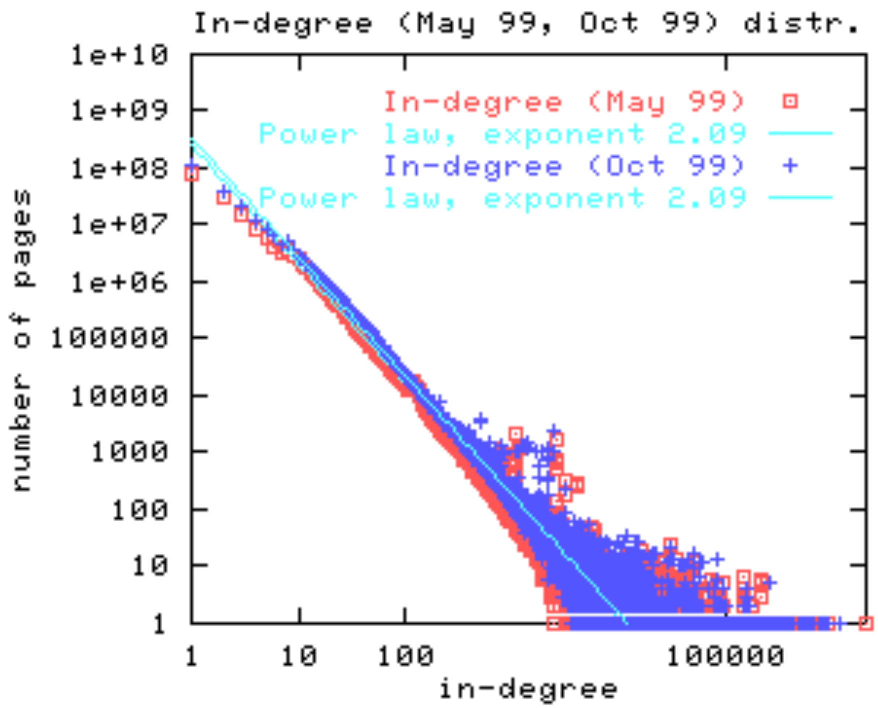
[Broder et al., ‘00]

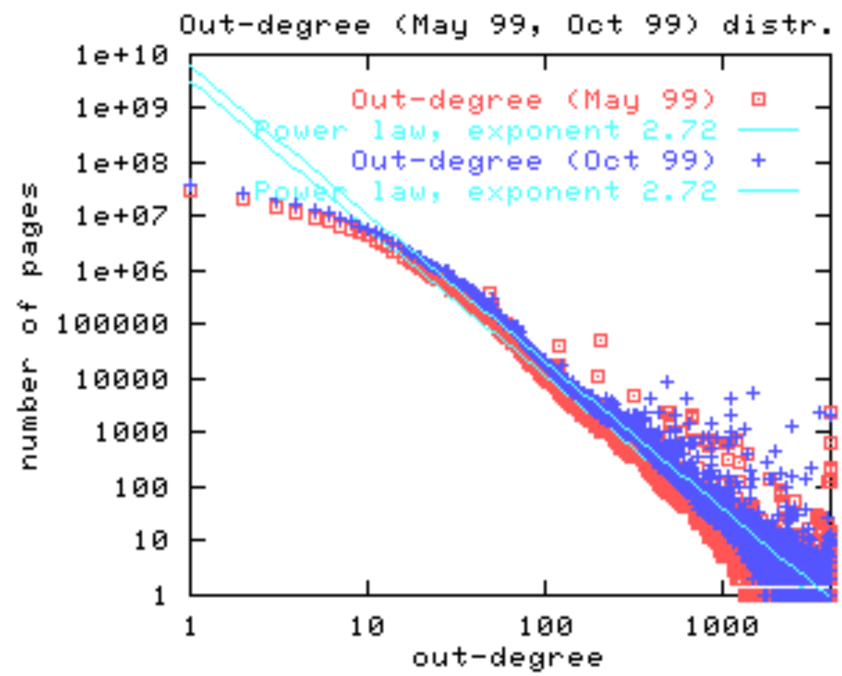
Diameter of the Web

- Average distance:
  75% of time there is no directed path from start to finish page
  - Follow in-links (directed): 16.12
  - Follow out-links (directed): 16.18
  - Undirected: 6.83
- Diameter of SCC (directed):
  - At least 28

[Broder et al., ‘00]

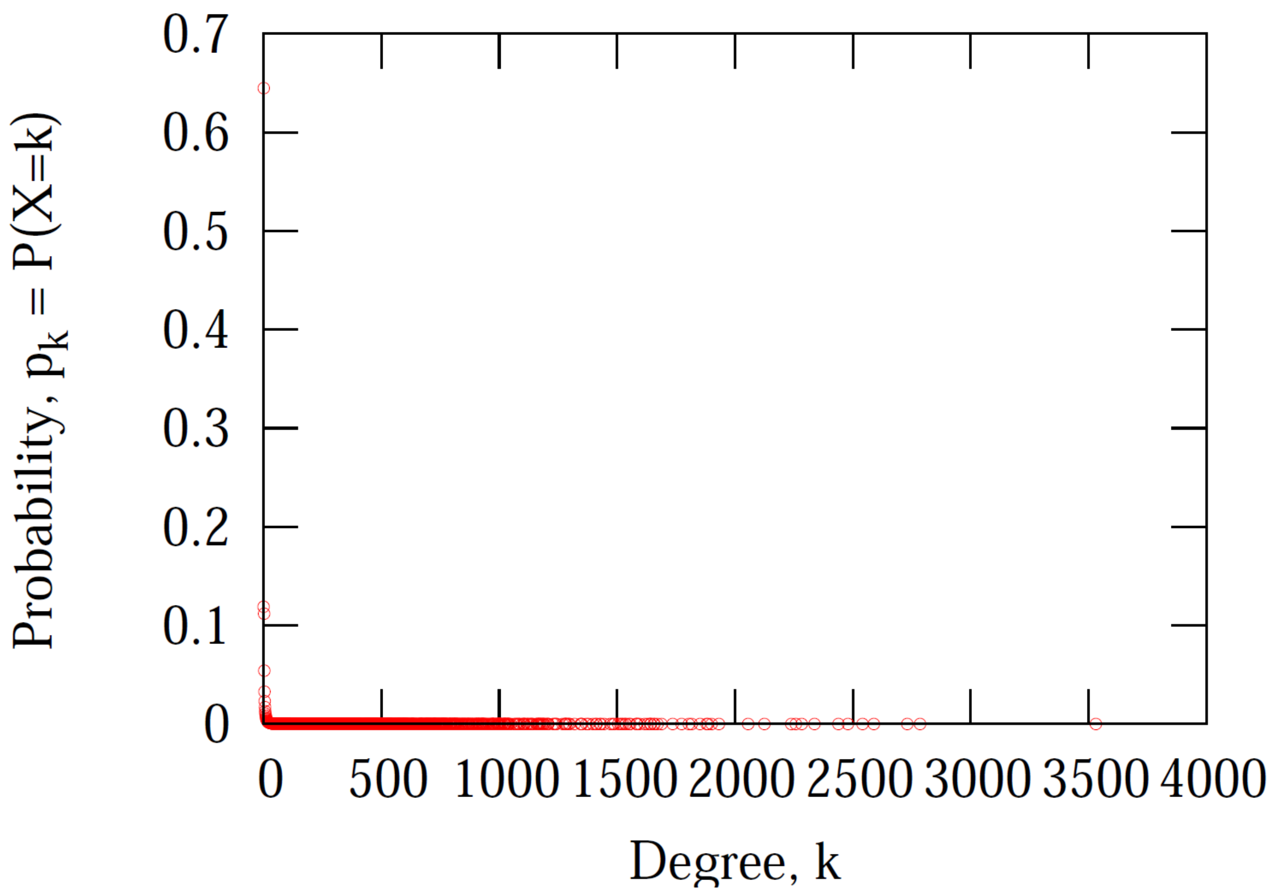
Degree distribution on the Web





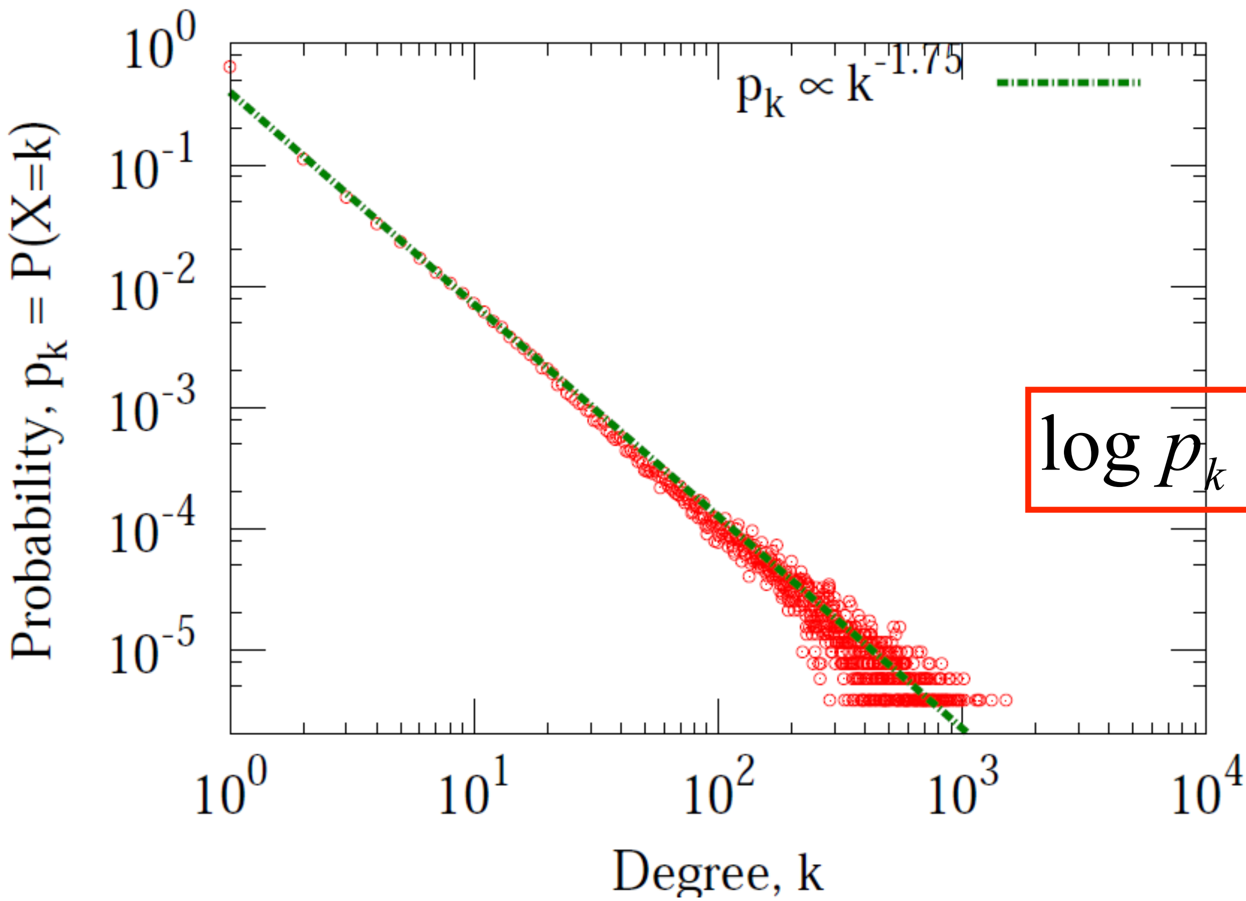
Degrees in real networks

- Take real network plot a histogram of p_k vs. k



Degrees in real networks (2)

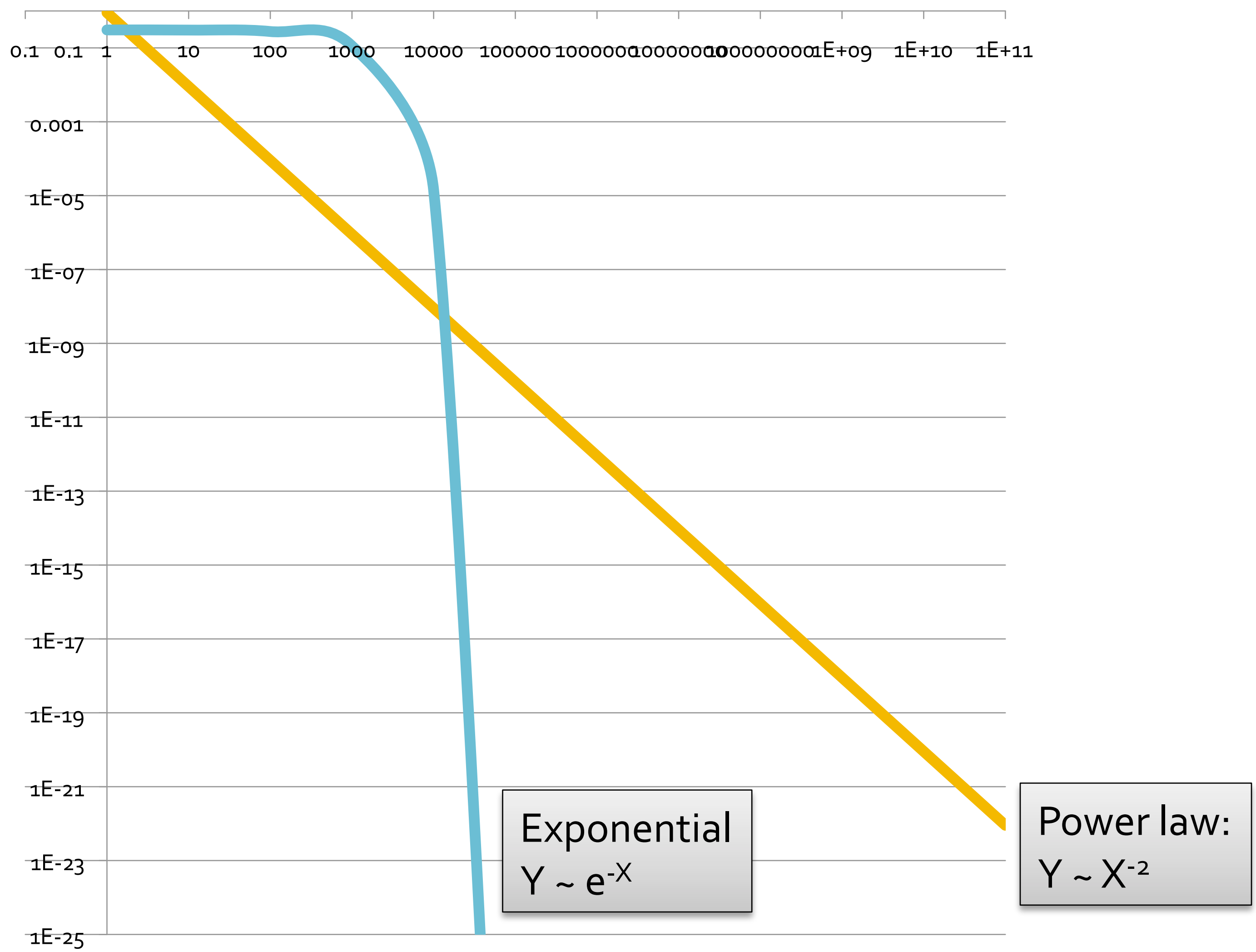
- Plot the same data on *log-log* axis:



$$p_k = \beta k^{-\alpha}$$

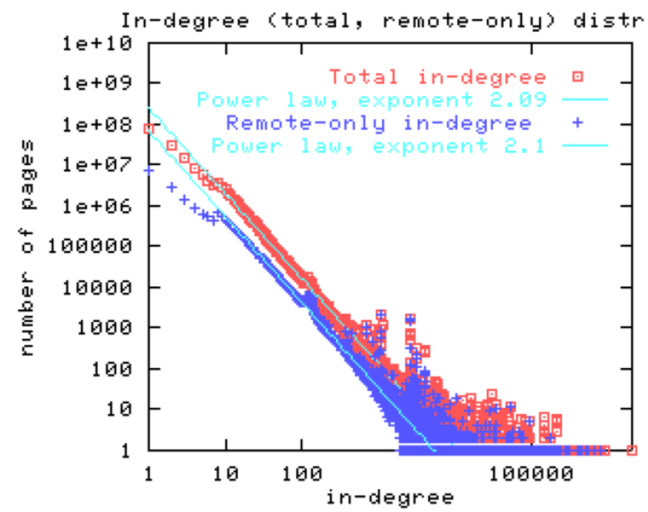
$$\log p_k = \log \beta - \alpha \log k$$

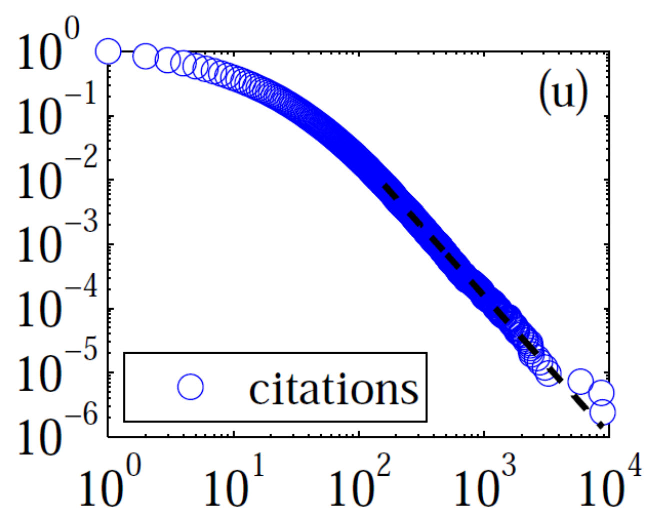
Exponential tail vs. Power-law tail



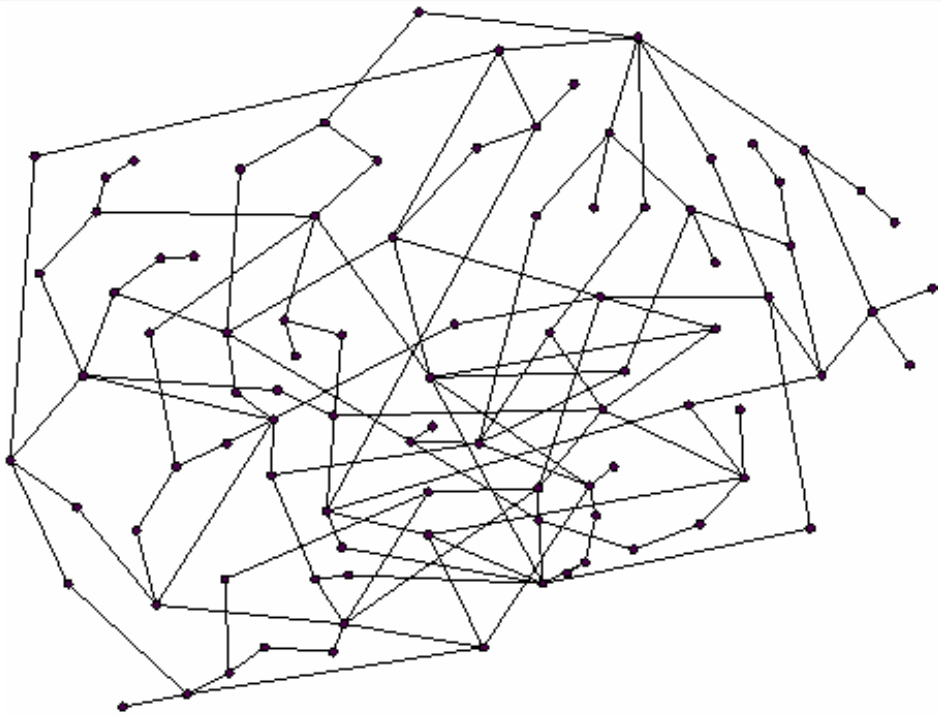
Power law degree exponents

- Power law degree exponent is typically $2 < \alpha < 3$
  - Web graph [Broder et al. 00]:
    - $\alpha_{in} = 2.1$, $\alpha_{out} = 2.4$
  - Autonomous systems [Faloutsos et al. 99]:
    - $\alpha = 2.4$
  - Actor collaborations [Barabasi-Albert 00]:
    - $\alpha = 2.3$
  - Citations to papers [Redner 98]:
    - $\alpha \approx 3$
  - Online social networks [Leskovec et al. 07]:
    - $\alpha \approx 2$



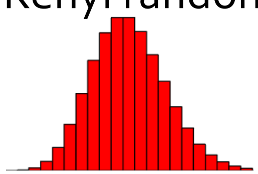


Power-law network

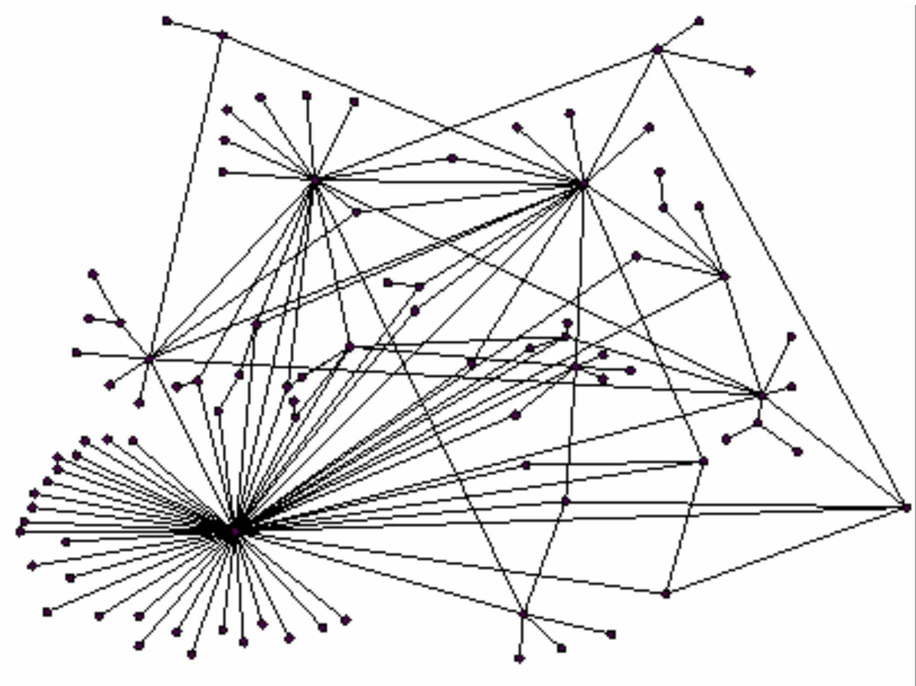


Random network

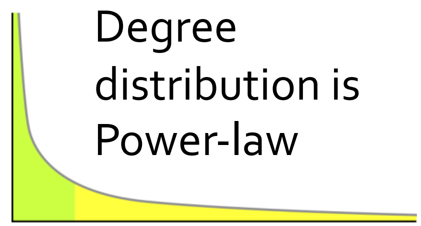
(Erdos-Renyi random graph)



Degree distribution is Binomial



Scale-free (power-law) network



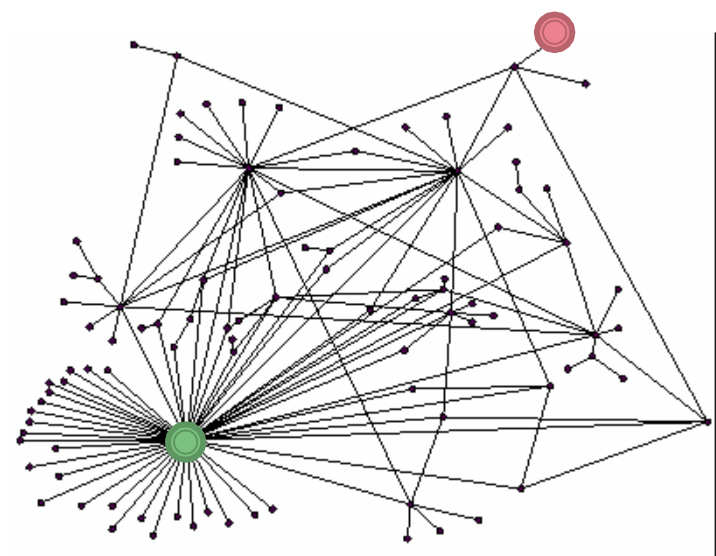
Degree distribution is Power-law

Function is scale free if:
$f(ax) = c\, f(x)$

Ranking nodes on the graph

- Web pages are not equally "important"
  - www.joe-schmoe.com v www.stanford.edu

- Since there is big diversity in the connectivity of the webgraph we can rank pages by the link structure



Links as votes

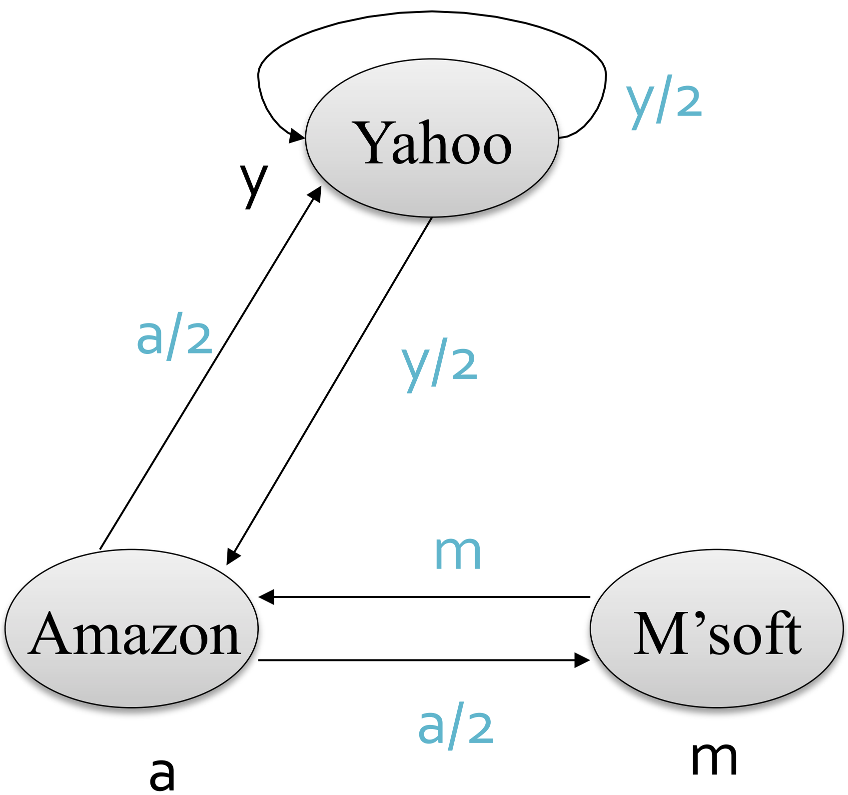
- First try:
  - Page is more important if it has more links
    - In-coming links? Out-going links?
- Think of in-links as votes:
  - www.stanford.edu has 23,400 inlinks
  - www.joe-schmoe.com has 1 inlink
- Are all in-links are equal?
  - Links from important pages count more
  - Recursive question!

Simple recursive formulation

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

Simple "flow" model

The web in 1839



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

Solving the flow equations

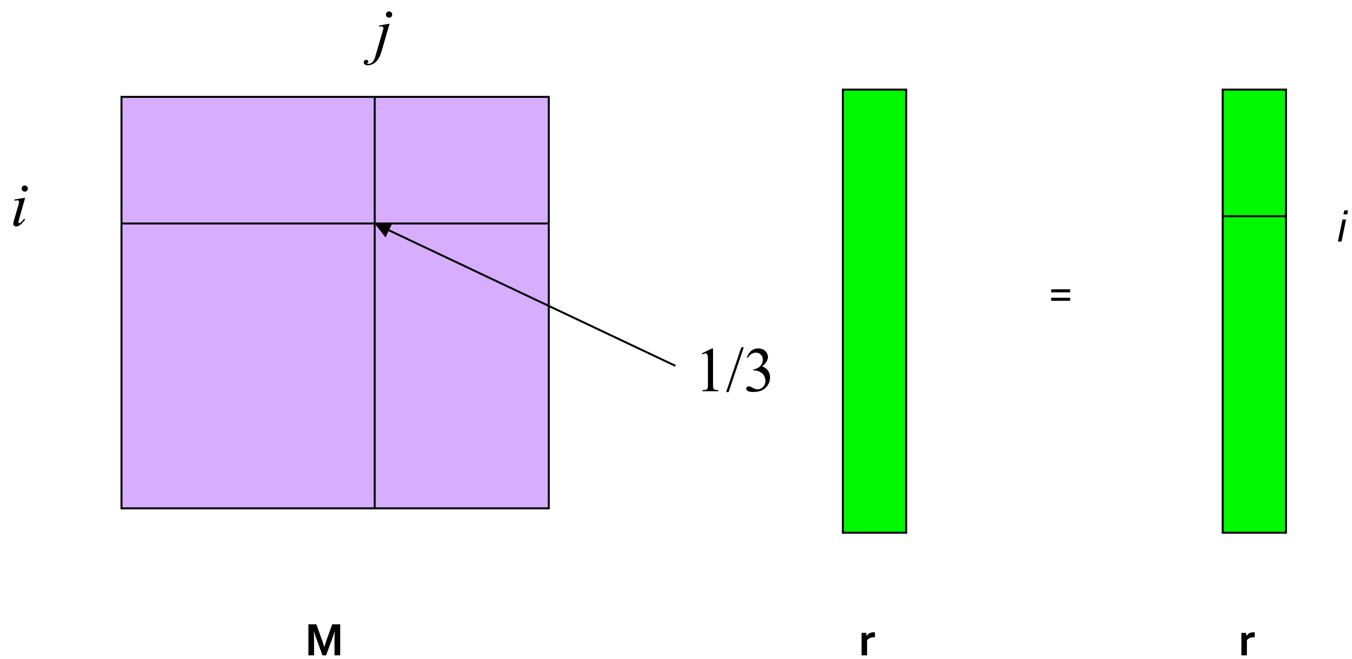
- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint forces uniqueness
  - $y+a+m = 1$
  - $y = 2/5$, $a = 2/5$, $m = 1/5$
- Gaussian elimination method works for small examples, but we need a better method for large web-size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

Matrix formulation

- Matrix M has one row and one column for each web page
- Suppose page j has n out-links
  - If $j \rightarrow i$, 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
  - $|r| = 1$

Example

Suppose page j links to 3 pages, including i



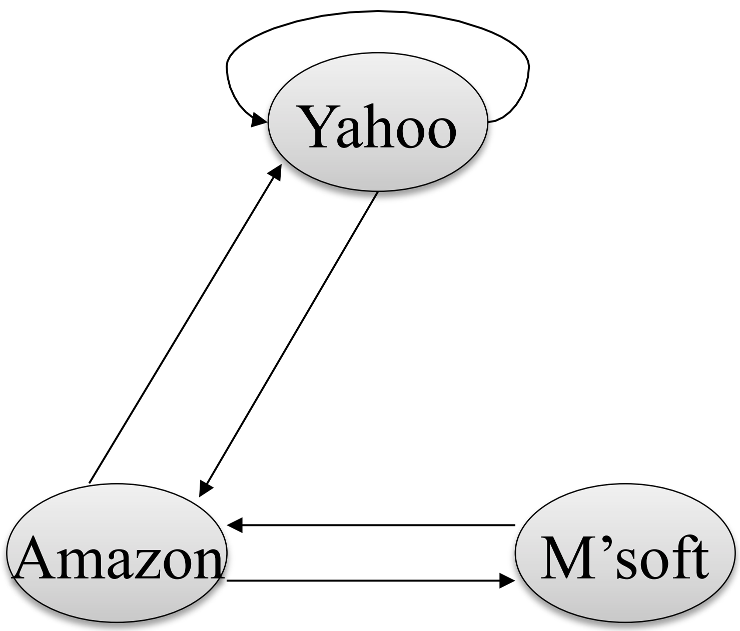
Eigenvector formulation

- 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



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

	Y!	A	MS
Y!	½	½	0
A	½	0	1
MS	0	½	0

r = Mr

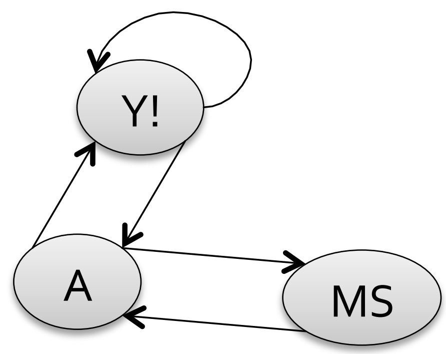
$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

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 $|\mathbf{r}^{k+1} - \mathbf{r}^k|_1 < \varepsilon$
  - $|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the L_1 norm
  - Can use any other vector norm e.g., Euclidean

Power Iteration Example

- Power iteration:
  - Set $r_i=1/n$
  - $r_i=\sum_j M_{ij} r_j$
  - And iterate



	Y!	A	MS
Y!	½	½	0
A	½	0	1
MS	0	½	0

- Example:

$$\begin{matrix} y \\ a \\ m \end{matrix} \; = \; \begin{matrix} 1/3 & 1/3 & 5/12 & 3/8 & & 2/5 \\ 1/3 & 1/2 & 1/3 & 11/24 & \ldots & 2/5 \\ 1/3 & 1/6 & 1/4 & 1/6 & & 1/5 \end{matrix}$$

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

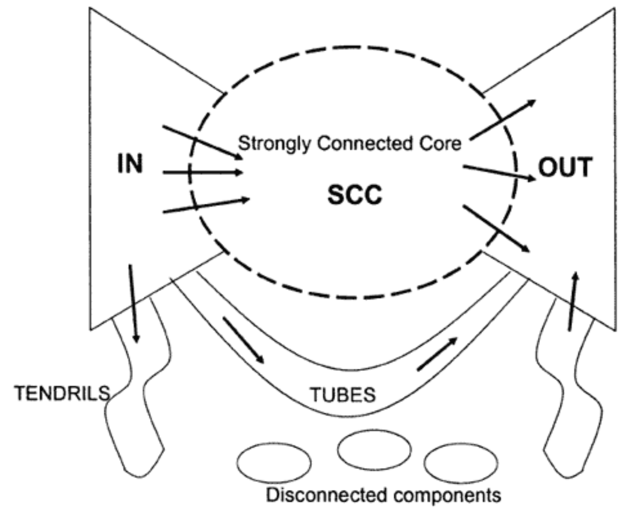
Existence and Uniqueness

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.

Problems with the "flow" model

- Some pages are "dead ends" (have no out-links)
  - Such pages cause importance to leak out



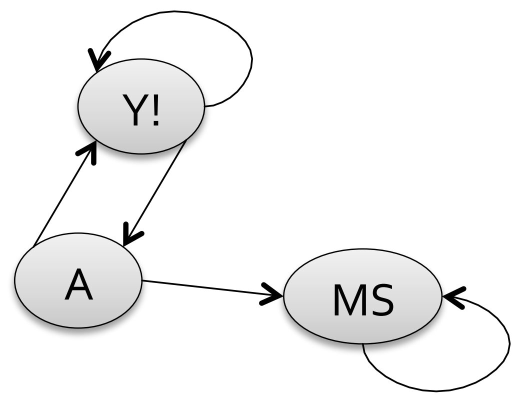
- Spider traps (all out links are within the group)
  - Eventually spider traps absorb all importance

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

Spider traps

- Power iteration:
  - Set $r_j=1$
  - $r_j=\sum_j M_{ij} r_j$
  - And iterate



	Y!	A	MS
Y!	½	½	0
A	½	0	0
MS	0	½	1

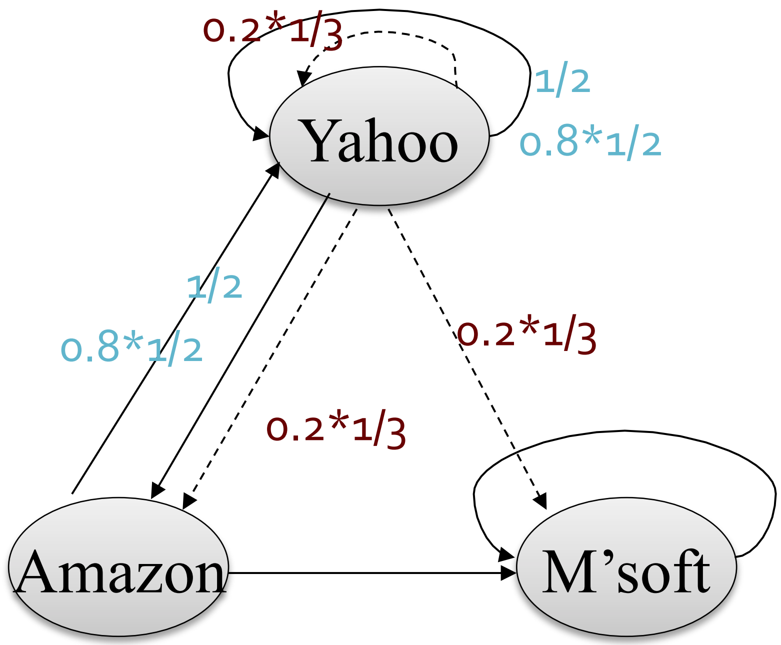
- Example:

y		1	1	¾	5/8		0
a	=	1	½	½	3/8	…	0
m		1	3/2	7/4	2		3

Solution: 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 (β = 0.8)

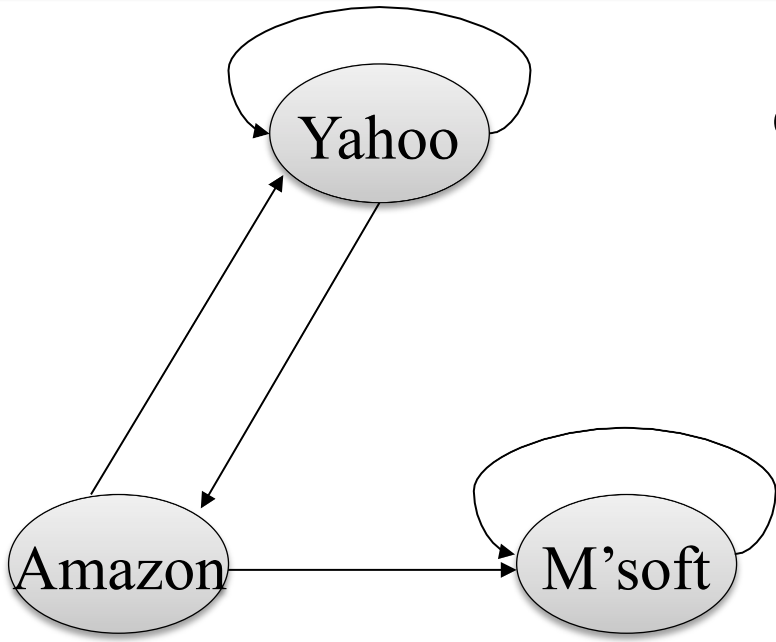


$$\begin{array}{cc} & y \\ \begin{array}{c} y \\ a \\ m \end{array} & \boxed{\begin{array}{c} 1/2 \\ 1/2 \\ 0 \end{array}} \end{array} \quad \begin{array}{c} y \\ 0.8* \boxed{\begin{array}{c} 1/2 \\ 1/2 \\ 0 \end{array}} \end{array} \quad \begin{array}{c} y \\ +\,0.2* \boxed{\begin{array}{c} 1/3 \\ 1/3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

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

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

Random teleports ($\beta = 0.8$)



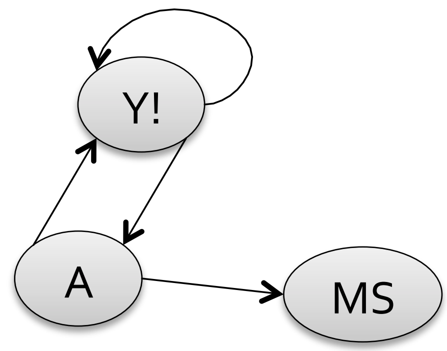
$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

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

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1 \\ 1 \\ 1 \end{array} \quad \begin{array}{c} 1.00 \\ 0.60 \\ 1.40 \end{array} \quad \begin{array}{c} 0.84 \\ 0.60 \\ 1.56 \end{array} \quad \begin{array}{c} 0.776 \\ 0.536 \\ 1.688 \end{array} \quad \begin{array}{c} \\ \ldots \\ \end{array} \quad \begin{array}{c} 7/11 \\ 5/11 \\ 21/11 \end{array}$$

Dead ends

- Power iteration:
  - Set $r_i=1$
  - $r_i=\sum_j M_{ij}r_j$
  - And iterate



	Y!	A	MS
Y!	½	½	0
A	½	0	0
MS	0	½	0

- Example:

y		1	1	¾	5/8		0
a	=	1	½	½	3/8	…	0
m		1	½	¼	¼		0

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

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

Page Rank

- Construct the N x 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 **r** = **Ar**
- Equivalently, **r** is the stationary distribution of the random walk with teleports

Computing page rank

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

Rearranging the equation

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

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

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

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

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

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

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

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 **M** on disk

Basic Algorithm:

- Initialize: $\mathbf{r}^{old} = [1/N]_N$
- Iterate:
  - Update: Perform a sequential scan of **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

Update step

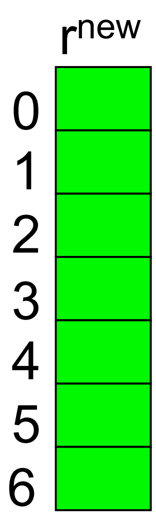
Initialize all entries of $\mathbf{r}^{new}$ to $(1-\beta)/N$

For each page p (out-degree n):

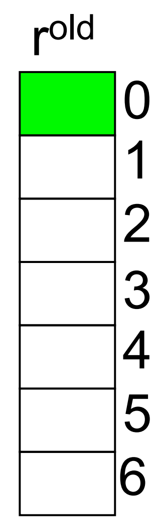
Read into memory: p, n, $dest_1, \ldots, dest_n$, $r^{old}(p)$

for j = 1..n:

$r^{new}(dest_j)$ += $\beta * r^{old}(p)/n$



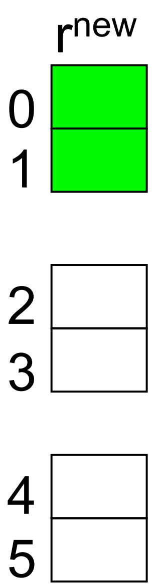
src	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23



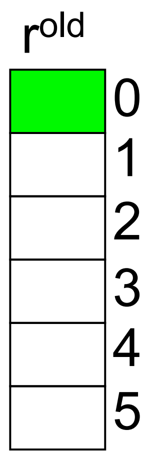
Analysis

- In each iteration, we have to:
  - Read $\mathbf{r}^{old}$ and **M**
  - Write $\mathbf{r}^{new}$ back to disk
  - IO Cost = $2|\mathbf{r}| + |\mathbf{M}|$
- What if we had enough memory to fit both $\mathbf{r}^{new}$ and $\mathbf{r}^{old}$?
- What if we could not even fit $\mathbf{r}^{new}$ in memory?
  - 10 billion pages

Block-based update algorithm



src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4



Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $\mathbf{r}^{new}$ into k blocks that fit in memory
  - Scan **M** and $\mathbf{r}^{old}$ once for each block
- k scans of **M** and $\mathbf{r}^{old}$
  - $k(|\mathbf{M}| + |\mathbf{r}|) + |\mathbf{r}| = k|\mathbf{M}| + (k+1)|\mathbf{r}|$
- Can we do better?
- Hint: **M** is much bigger than **r** (approx 10-20x), so we must avoid reading it k times per iteration

Block-Stripe Update algorithm

r^{new}

0
1

src	degree	destination
0	4	0, 1
1	3	0
2	2	1

2
3

0	4	3
2	2	3

4
5

0	4	5
1	3	5
2	2	4

r^{old}

0
1
2
3
4
5

Block-Stripe Analysis

- Break **M** into stripes
  - Each stripe contains only destination nodes in the corresponding block of $\mathbf{r}^{new}$
- Some additional overhead per stripe
  - But usually worth it
- Cost per iteration
  - $|\mathbf{M}|(1+\varepsilon) + (k+1)|\mathbf{r}|$