Machine Learning and Data Mining

Link Analysis Algorithms

Page Rank

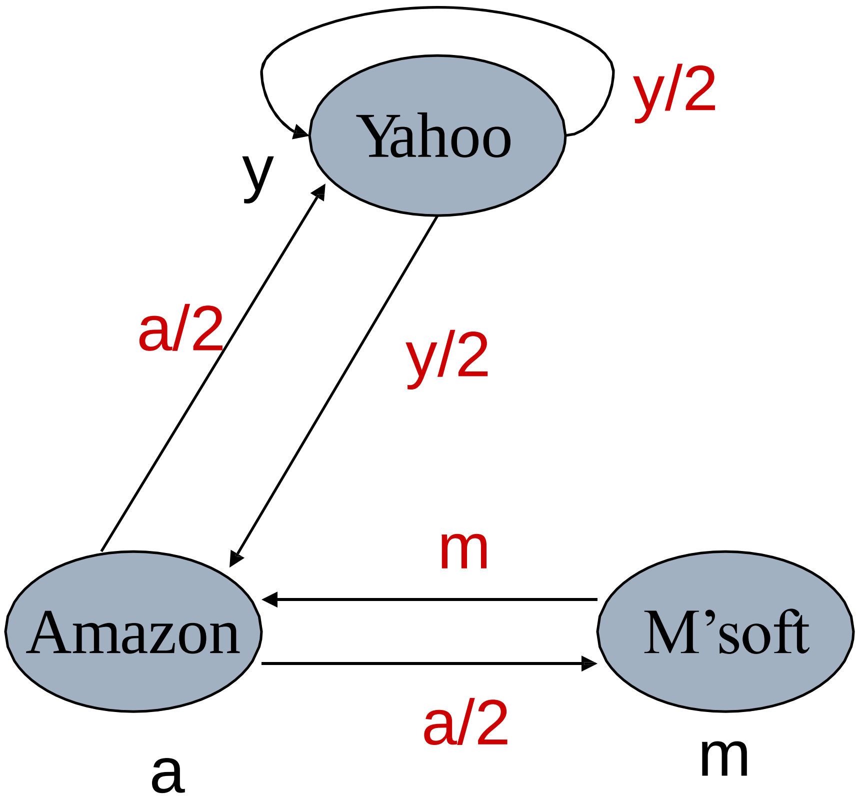
Ranking web pages

- Web pages are not equally "important"
  - www.joe-schmoe.com v www.stanford.edu
- Inlinks as votes
  - www.stanford.edu has 23,400 inlinks
  - www.joe-schmoe.com has 1 inlink
- 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

Simple “flow” model



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

Solving the flow equations

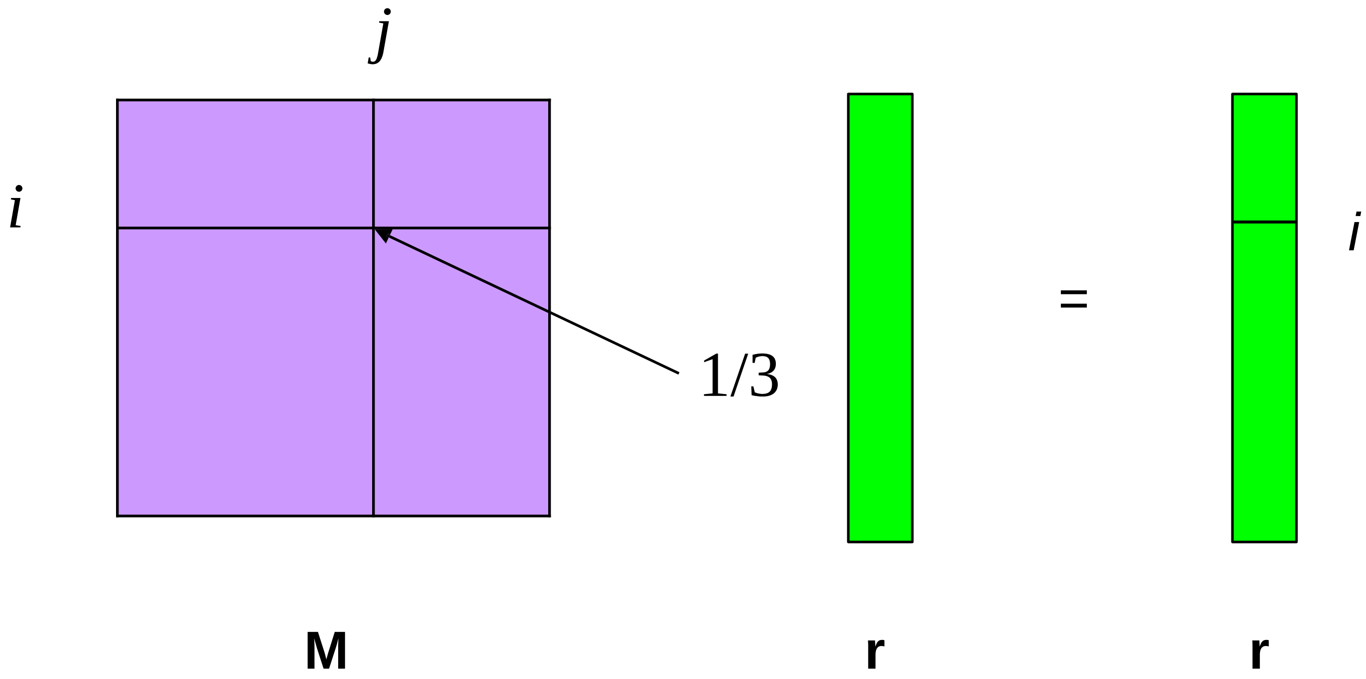
- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint forces uniqueness
  - y+a+m = 1
  - y = 2/5, a = 2/5, m = 1/5
- Gaussian elimination method works for small examples, but we need a better method for large graphs

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
  - $|\mathbf{r}| = 1$

Example

Suppose page *j* links to 3 pages, including *i*



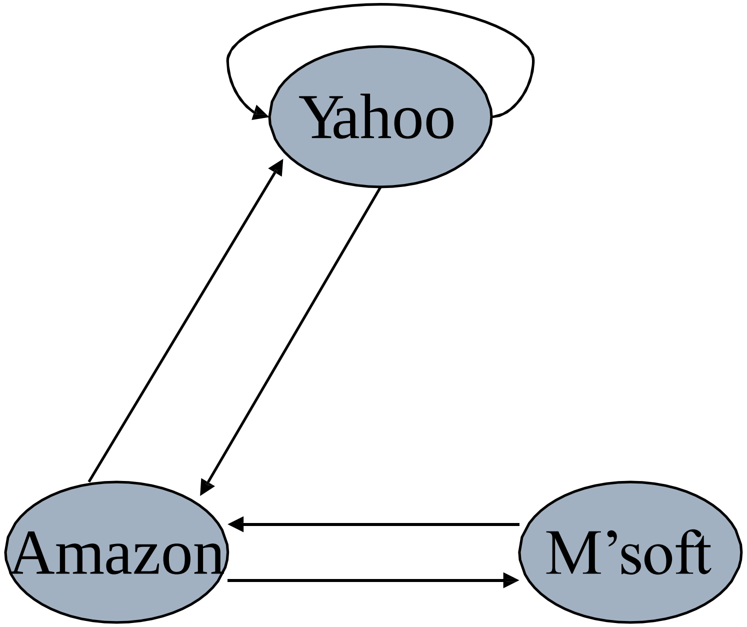
Eigenvector formulation

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

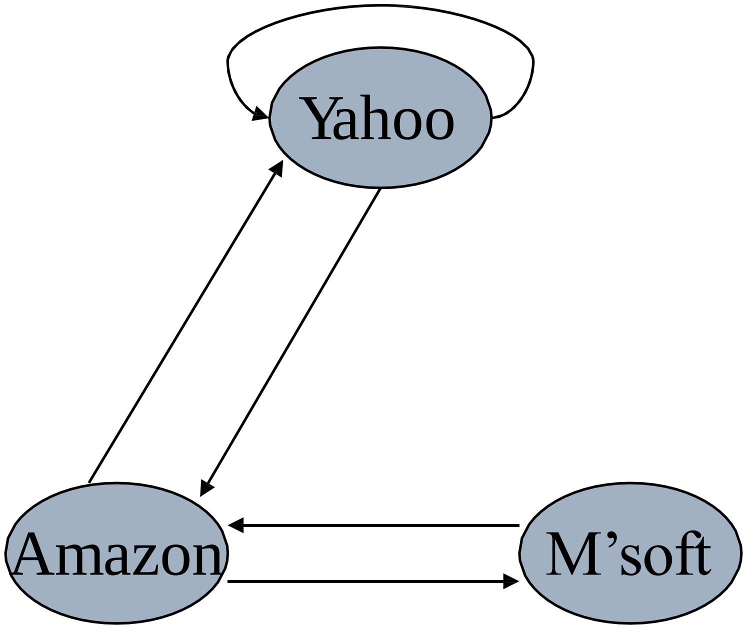
r = Mr

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

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



	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

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 **p**(t) be a vector whose i^{th} component is the probability that the surfer is at page i at time t
  - **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

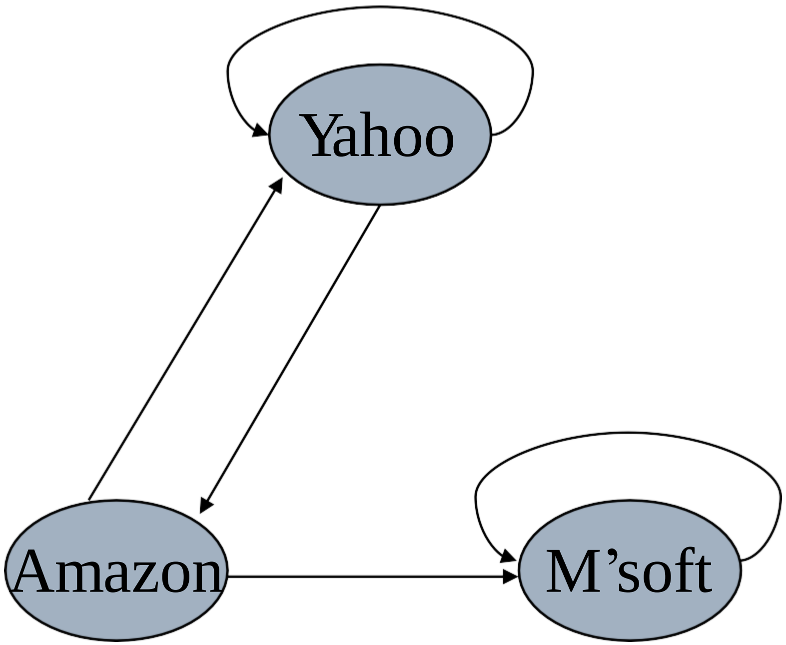
A central result from the theory of random walks (aka Markov processes):

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
- Spider traps violate the conditions needed for the random walk theorem

Microsoft becomes a spider trap



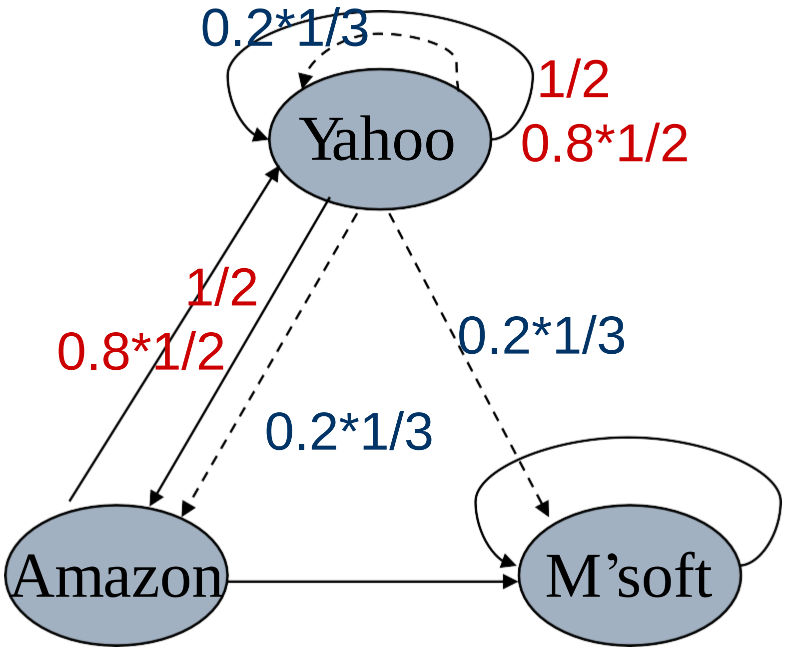
	y	a	m
y	1/2	1/2	0
a	1/2	0	0
m	0	1/2	1

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

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 ($\beta = 0.8$)

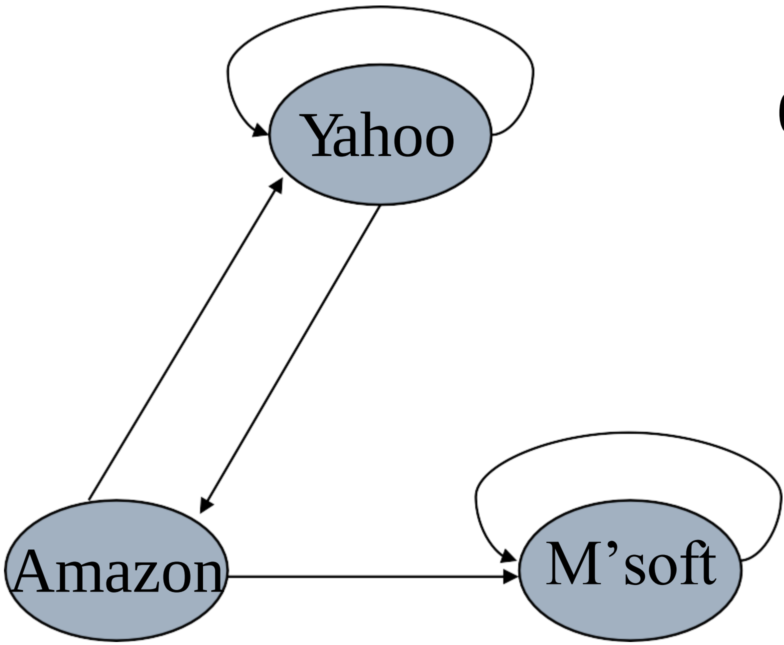


$$\begin{matrix} & y \\ y & 1/2 \\ a & 1/2 \\ m & 0 \end{matrix} \quad 0.8* \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \end{bmatrix}^{y} + 0.2* \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}^{y}$$

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

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

Random teleports ($\beta = 0.8$)



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

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

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

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

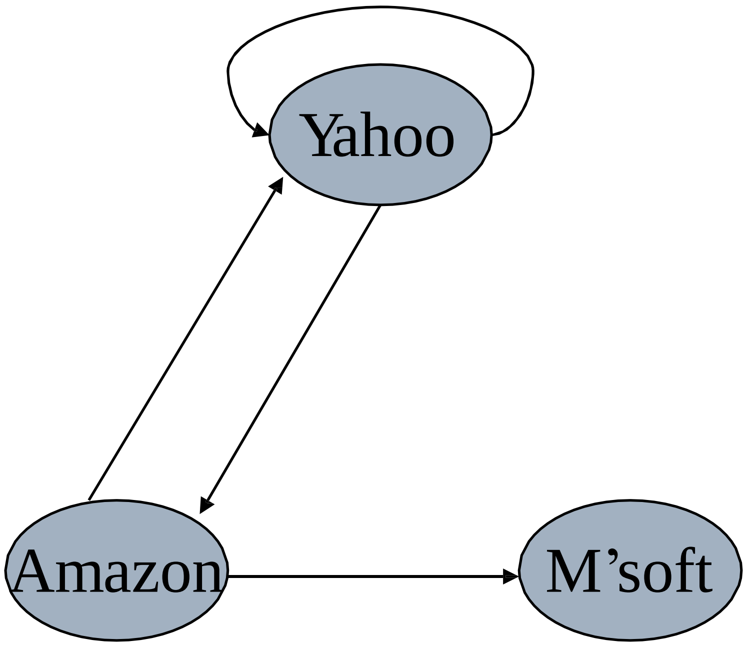
Page Rank

- Construct the N£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

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!

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1 \\ 1 \\ 1 \end{array} \quad \begin{array}{c} 1 \\ 0.6 \\ 0.6 \end{array} \quad \begin{array}{c} 0.787 \\ 0.547 \\ 0.387 \end{array} \quad \begin{array}{c} 0.648 \\ 0.430 \\ 0.333 \end{array} \quad \ldots \quad \begin{array}{c} 0 \\ 0 \\ 0 \end{array}$$

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 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

$\mathbf{r} = \mathbf{A}\mathbf{r}$, 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 $|\mathbf{r}| = 1$

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

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 **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

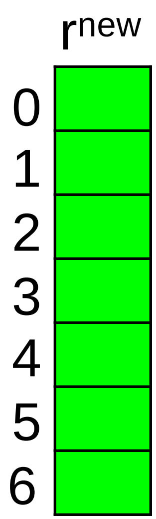
Initialize all entries of $\mathbf{r}^{new}$ to $(1-\beta)/N$

For each page p (out-degree n):

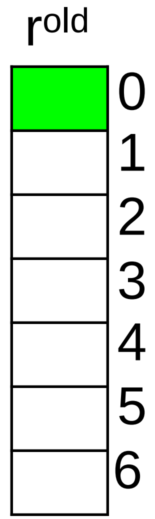
  Read into memory: p, n, $dest_1, \ldots, dest_n$, $r^{old}(p)$

  for j = 1..n:

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



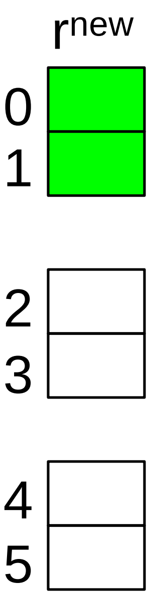
src	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23



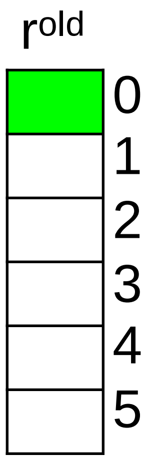
Analysis

- In each iteration, we have to:
  - Read $\mathbf{r}^{old}$ and $\mathbf{M}$
  - 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



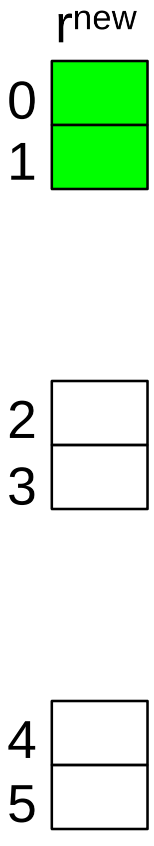
src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4



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

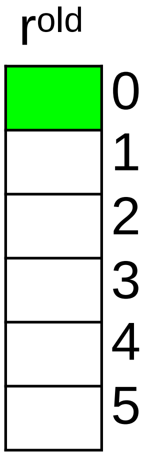
Block-Stripe Update algorithm



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

0	4	3
2	2	3

0	4	5
1	2	5
2	2	4



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
  - $|\mathbf{M}|(1+\varepsilon) + (k+1)|\mathbf{r}|$