

## Web Mining

### Link Analysis Algorithms Page Rank

## Ranking web pages

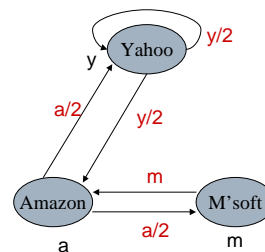
- Web pages are not equally "important"
  - [www.joe-schmoe.com](http://www.joe-schmoe.com) v [www.stanford.edu](http://www.stanford.edu)
- Inlinks as votes
  - [www.stanford.edu](http://www.stanford.edu) has 23,400 inlinks
  - [www.joe-schmoe.com](http://www.joe-schmoe.com) has 1 inlink
- Are all inlinks 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

## Simple "flow" model

The web in 1839



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

## Solving the flow equations

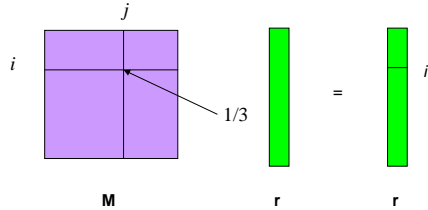
- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint forces uniqueness
  - $y+a+m = 1$
  - $y = 2/5, a = 2/5, m = 1/5$
- Gaussian elimination method works for small examples, but we need a better method for large graphs

## Matrix formulation

- Matrix **M** has one row and one column for each web page
- If page **j** has **n** outlinks and links to page **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**

## Example

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



## Eigenvector formulation

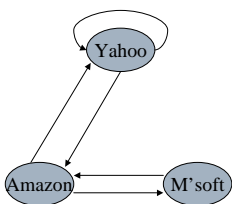
□ The flow equations can be written

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

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

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

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

$$\begin{aligned} y &= y/2 + a/2 \\ a &= y/2 + m \\ m &= a/2 \end{aligned}$$

## Power Iteration method

□ Simple iterative scheme (aka **relaxation**)

□ Suppose there are  $N$  web pages

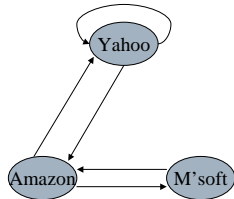
□ Initialize:  $\mathbf{r}^0 = [1/N, \dots, 1/N]^T$

□ Iterate:  $\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$

□ Stop when  $|\mathbf{r}^{k+1} - \mathbf{r}^k|_1 < \epsilon$

- $|\mathbf{x}|_1 = \sum_{i=1}^N |x_i|$  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 |     |

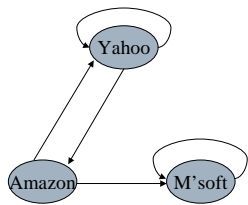
## Spider traps

□ 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 | ... |
| 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  $\beta$ , follow a link at random
  - With probability  $1-\beta$ , jump to some page uniformly at random
  - Common values for  $\beta$  are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps

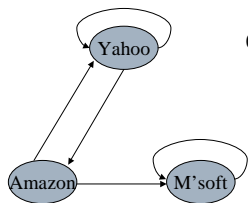
## Matrix formulation

- Suppose there are  $N$  pages
  - Consider a page  $j$ , with set of outlinks  $O(j)$
  - We have  $M_{ij} = 1/|O(j)|$  when  $|O(j)| > 0$  and  $j$  links to  $i$ ;  $M_{ij} = 0$  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  $N \times N$  matrix  $\mathbf{A}$  as follows
  - $A_{ij} = \beta M_{ij} + (1-\beta)/N$
- Verify that  $\mathbf{A}$  is a stochastic matrix
- The **page rank vector**  $\mathbf{r}$  is the principal eigenvector of this matrix
  - satisfying  $\mathbf{r} = \mathbf{A}\mathbf{r}$
- Equivalently,  $\mathbf{r}$  is the stationary distribution of the random walk with teleports

## Previous example with $\beta=0.8$



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

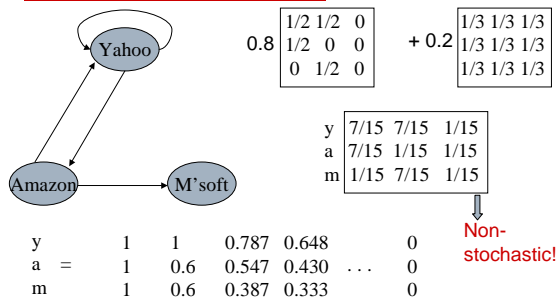
|   | y    | a    | m     |
|---|------|------|-------|
| y | 7/15 | 7/15 | 1/15  |
| a | 7/15 | 1/15 | 1/15  |
| m | 1/15 | 7/15 | 13/15 |

|   |   |      |      |       |       |
|---|---|------|------|-------|-------|
| y | 1 | 1.00 | 0.84 | 0.776 | 7/11  |
| a | 1 | 0.60 | 0.60 | 0.536 | ...   |
| m | 1 | 1.40 | 1.56 | 1.688 | 21/11 |

## Dead ends

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

## Microsoft becomes a dead end



## 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 multiply
  - $\mathbf{r}^{\text{new}} = \mathbf{A}\mathbf{r}^{\text{old}}$
- 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  $\mathbf{A}$  has  $N^2$  entries
    - $10^{18}$  is a large number!

## Sparse matrix formulation

- Although  $\mathbf{A}$  is a dense matrix, it is obtained from a sparse matrix  $\mathbf{M}$ 
  - 10 links per node, approx 10N entries
- We can restate the page rank equation
  - $\mathbf{r} = \beta\mathbf{M}\mathbf{r} + [(1-\beta)/N]\mathbf{1}_N$
  - $[(1-\beta)/N]\mathbf{1}_N$  is an  $N$ -vector with all entries  $(1-\beta)/N$
- 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  $\mathbf{r}^{\text{new}}$

## Sparse matrix encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - say 10N, or  $4 \times 10^8 \times 1 \text{ billion} = 40\text{GB}$
  - 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  $\mathbf{r}^{\text{new}}$ , plus some working memory
  - Store  $\mathbf{r}^{\text{old}}$  and matrix  $\mathbf{M}$  on disk
- Basic Algorithm:**
  - Initialize:  $\mathbf{r}^{\text{old}} = [1/N]\mathbf{1}_N$
  - Iterate:
    - **Update:** Perform a sequential scan of  $\mathbf{M}$  and  $\mathbf{r}^{\text{old}}$  and update  $\mathbf{r}^{\text{new}}$
    - Write out  $\mathbf{r}^{\text{new}}$  to disk as  $\mathbf{r}^{\text{old}}$  for next iteration
    - Every few iterations, compute  $|\mathbf{r}^{\text{new}} - \mathbf{r}^{\text{old}}|$  and stop if it is below threshold

## Update step

Initialize all entries of  $r^{new}$  to  $(1-\beta)/N$   
 For each page  $p$  (out-degree  $n$ ):  
 Read into memory:  $p, n, dest_1, \dots, dest_n, r^{old}(p)$   
 for  $j = 1..n$ :  
 $r^{new}(dest_j) += \beta * r^{old}(p) / n$

| $r^{new}$ | src | degree | destination      | $r^{old}$ |
|-----------|-----|--------|------------------|-----------|
| 0         | 0   | 3      | 1, 5, 6          | 0         |
| 1         | 1   | 4      | 17, 64, 113, 117 | 1         |
| 2         | 2   | 2      | 13, 23           | 2         |
| 3         |     |        |                  | 3         |
| 4         |     |        |                  | 4         |
| 5         |     |        |                  | 5         |
| 6         |     |        |                  | 6         |

## Analysis

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

## Block-based update algorithm

| $r^{new}$ | src | degree | destination | $r^{old}$ |
|-----------|-----|--------|-------------|-----------|
| 0         | 0   | 4      | 0, 1, 3, 5  | 0         |
| 1         | 1   | 2      | 0, 5        | 1         |
| 2         | 2   | 2      | 3, 4        | 2         |
| 3         |     |        |             | 3         |
| 4         |     |        |             | 4         |
| 5         |     |        |             | 5         |

## Analysis of Block Update

- Similar to nested-loop join in databases
  - Break  $r^{new}$  into  $k$  blocks that fit in memory
  - Scan  $M$  and  $r^{old}$  once for each block
- $k$  scans of  $M$  and  $r^{old}$ 
  - $k(|M| + |r|) + |r| = k|M| + (k+1)|r|$
- 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

| $r^{new}$ | src | degree | destination | $r^{old}$ |
|-----------|-----|--------|-------------|-----------|
| 0         | 0   | 4      | 0, 1        | 0         |
| 1         | 1   | 3      | 0           | 1         |
| 2         | 2   | 2      | 1           | 2         |
| 3         | 0   | 4      | 3           | 3         |
| 4         | 2   | 2      | 3           | 4         |
| 5         | 0   | 4      | 5           | 5         |
|           | 1   | 3      | 5           |           |
|           | 2   | 2      | 4           |           |

## Block-Stripe Analysis

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