Web Mining
Link Analysis Algorithms
Page Rank

Ranking web pages
- Web pages are not equally "important"
  - www.joe-schmoe.com vs 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

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 Mij = 1/n
  - Else Mij = 0
- M is a column stochastic matrix
  - Columns sum to 1
- Suppose r is a vector with one entry per web page
  - r 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
  - r = Mr
- So the rank vector is an eigenvector of the stochastic web matrix
  - In fact, its first or principal eigenvector, with corresponding eigenvalue 1

Example
- Suppose M's eigenvalues are y, a, m
  - y = y/2 + a/2
  - a = y/2 + m
  - m = a/2
Power Iteration method
- Simple iterative scheme (aka relaxation)
- Suppose there are $N$ web pages
- Initialize: $r^0 = [1/N, ..., 1/N]^T$
- Iterate: $r^{k+1} = Mr^k$
- Stop when $|r^{k+1} - r^k|_1 < \varepsilon$
  - $|x|_1 = \sum_{i=1}^N |x_i|$ is the $L_1$ norm
  - Can use any other vector norm e.g., Euclidean

Power Iteration Example
- Yahoo
- M'soft
- Amazon

\[
\begin{pmatrix}
y & a & m \\
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 0 \\
m & 0 & 1/2 & 1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
y & a & m \\
y & 1/3 & 1/3 & 1/3 \\
a & 1/3 & 1/3 & 1/3 \\
m & 1/3 & 1/3 & 1/3 \\
\end{pmatrix}
\]

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

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 $A$ 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

Previous example with $\beta = 0.8$

\[
\begin{pmatrix}
y & a & m \\
y & 1.00 & 0.64 & 0.76 \\
a & 1.00 & 0.60 & 0.58 \\
m & 1.40 & 1.36 & 1.68 \\
\end{pmatrix}
\]

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

Computing page rank

Sparse matrix formulation

Sparse matrix encoding

Basic Algorithm

Update step

Analysis

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

Block-Stripe Update algorithm

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