COMP 465: Data Mining
More on PageRank

Slides Adapted From: www.mmds.org (Mining Massive Datasets)

PageRank: How to solve?

- **Power Iteration:**
  - Set \( r_j = 1/N \)
  - 1: \( r'_j = \sum_{i \rightarrow j} \frac{r_i}{d_i} \)
  - 2: \( r = r' \)
  - Goto 1

- **Example:**
  - \( r_y = \frac{1}{3} \)
  - \( r_a = \frac{1}{3} \)
  - \( r_m = \frac{1}{3} \)

Random Walk Interpretation

- Imagine a random web surfer:
  - At any time \( t \), surfer is on some page \( i \)
  - At time \( t + 1 \), the surfer follows an out-link from \( i \) uniformly at random
  - Ends up on some page \( j \) linked from \( i \)
  - Process repeats indefinitely

- Let:
  - \( p(i) \) ... vector whose \( i^{th} \) coordinate is the prob. that the surfer is at page \( i \) at time \( t \)
  - So, \( p(t) \) is a probability distribution over pages

```
\[ r_j = \sum_{i \rightarrow j} \frac{r_i}{d_{out}(i)} \]
```
The Stationary Distribution

- Where is the surfer at time $t+1$?
  - Follows a link uniformly at random
    \[ p(t + 1) = M \cdot p(t) \]
  - Suppose the random walk reaches a state
    \[ p(t + 1) = M \cdot p(t) = p(t) \]
    then $p(t)$ is stationary distribution of a random walk
- Our original rank vector $r$ satisfies
  \[ r = M \cdot r \]
  - So, $r$ is a stationary distribution for the random walk

PageRank: The Google Formulation

PageRank: Three Questions

- Does this converge?
- Does it converge to what we want?
- Are results reasonable?

\[ r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i} \]
or equivalently \[ r = Mr \]

Does this converge?

Example:
\[ r_a = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}, \quad r_b = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix} \]

Iteration 0, 1, 2, ...
**Problem: Spider Traps**

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - And iterate

- **Example:**
  - $r_y = \frac{1}{3}$, $2/6$, $3/12$, $5/24$, $0$
  - $r_a = \frac{1}{3}$, $1/6$, $2/12$, $3/24$, $0$
  - $r_m = \frac{1}{3}$, $3/6$, $7/12$, $16/24$, $1$

  All the PageRank score gets "trapped" in node $m$. 

**Solution: Teleports!**

- The Google solution for spider traps: At each time step, the random surfer has two options
  - With prob. $\beta$, follow a link at random
  - With prob. $1-\beta$, jump to some random page
- 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
Problem: Dead Ends

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$
  - And iterate

- **Example:**
  $r_y = \frac{1}{3}$, $r_a = \frac{1}{6}$, $r_m = \frac{1}{12}$

Here the PageRank "leaks" out since the matrix is not stochastic.

Solution: Always Teleport!

- **Teleports:** Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly

Why Teleports Solve the Problem?

- Why are dead-ends and spider traps a problem and why do teleports solve the problem?
  - **Spider-traps** are not a problem, but with traps PageRank scores are not what we want
    - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
  - **Dead-ends** are a problem
    - The matrix is not column stochastic so our initial assumptions are not met
    - **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution: Random Teleports

- **Google’s solution that does it all:** At each step, random surfer has two options:
  - With probability $\beta$, follow a link at random
  - With probability $1 - \beta$, jump to some random page

- **PageRank equation** [Brin-Page, 98]
  \[
  r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}
  \]

This formulation assumes that $M$ has no dead ends. We can either preprocess matrix $M$ to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.
The Google Matrix

- PageRank equation [Brin-Page, '98]
  \[ r_j = \sum_{i \to j} \beta r_i / d_i + (1 - \beta) \frac{1}{N} \]

- The Google Matrix \( A \):
  \[ A = \beta M + (1 - \beta) \left[ \frac{1}{N} \right]_{N \times N} \]

- We have a recursive problem: \( r = A \cdot r \)
  And the Power method still works!

- What is \( \beta \)?
  - In practice \( \beta = 0.8, 0.9 \) (make 5 steps on avg., jump)

Random Teleports (\( \beta = 0.8 \))

Computing Page Rank

- Key step is matrix-vector multiplication
  \[ r^{\text{new}} = A \cdot r^{\text{old}} \]
  - Easy if we have enough main memory to hold \( A, r^{\text{old}}, r^{\text{new}} \)
- 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!

- How do we actually compute the PageRank?
Matrix Formulation

- Suppose there are $N$ pages
- Consider page $i$, with $d_i$ out-links
- We have $M_{ij} = 1/|d_j|$ when $i \rightarrow j$
- and $M_{ij} = 0$ otherwise
- The random teleport is equivalent to:
  - Adding a teleport link from $i$ to every other page
  - Reducing the probability of each out-link from $1/|d_j|$ to $\beta/|d_j|$ where $\beta$ is a teleport parameter
- Equivalent: Tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

Rearranging the Equation

- $r = A \cdot r$, where $A_{ji} = \beta M_{ji} + \frac{1-\beta}{N}$
- $r_j = \sum_{i=1}^{N} A_{ji} \cdot r_i$
- $r_j = \sum_{i=1}^{N} \left[ \beta M_{ji} + \frac{1-\beta}{N} \right] \cdot r_i$
- $= \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{i=1}^{N} r_i$
- $= \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N}$ since $\sum r_i = 1$
- So we get: $r = \beta M \cdot r + \frac{1-\beta}{N} \cdot \frac{1}{N}$

Sparse Matrix Formulation

- We just rearranged the PageRank equation $r = \beta M \cdot r + \frac{1-\beta}{N} \cdot \frac{1}{N}$
- $M$ is a sparse matrix! (with no dead-ends)
- 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute $r^{new} = \beta M \cdot r^{old}$
  - Add a constant value $(1-\beta)/N$ to each entry in $r^{new}$
  - Note if $M$ contains dead-ends then $\sum r^{new}_j < 1$ and we also have to renormalize $r^{new}$ so that it sums to 1

PageRank: The Complete Algorithm

- Input: Graph $G$ and parameter $\beta$
- Directed graph $G$ (can have spider traps and dead ends)
- Parameter $\beta$
- Output: PageRank vector $r^{new}$

- Set: $r^{old}_j = \frac{1}{N}$
- repeat until convergence: $\sum_j |r^{new}_j - r^{old}_j| > \varepsilon$
  - $\forall j: r^{new}_j = \sum_{i \rightarrow j} \beta \frac{r^{old}_i}{d_i}$
- $r^{new}_j = 0$ if in-degree of $j$ is 0
- Now re-insert the leaked PageRank:
  - $\forall j: r^{new}_j = r^{new}_j + \frac{1-\beta}{N}$ where: $S = \sum r^{new}_j$
- $r^{old} = r^{new}$

Note: Here we assumed $M$ has no dead-ends $[x]_N$ is a vector of length $N$ with all entries $x$