Announcements:
- Thu May 2 – Homework 2 due and releasing Homework 3
- Thu May 9 – Project Milestone (make sure to have dataset in hand/disk)
- ETL Course Assessment today

Analysis of Large Graphs: Link Analysis, PageRank
# New Topic: Graph Data!

<table>
<thead>
<tr>
<th>High dim. data</th>
<th>Graph data</th>
<th>Infinite data</th>
<th>Machine learning</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locality sensitive hashing</td>
<td>PageRank, SimRank</td>
<td>Sampling data streams</td>
<td>SVM</td>
<td>Recommender systems</td>
</tr>
<tr>
<td>Clustering</td>
<td>Community Detection</td>
<td>Filtering data streams</td>
<td>Decision Trees</td>
<td>Association Rules</td>
</tr>
<tr>
<td>Dimensionality reduction</td>
<td>Spam Detection</td>
<td>Queries on streams</td>
<td>Perceptron, kNN</td>
<td>Duplicate document detection</td>
</tr>
</tbody>
</table>
Graph Data: Social Networks

Facebook social graph
4-degrees of separation [Backstrom-Boldi-Rosa-Ugander-Vigna, 2011]
Graph Data: Media Networks

Connections between political blogs
Polarization of the network [Adamic-Glance, 2005]
Graph Data: Information Nets

Citation networks and Maps of science
[Börner et al., 2012]
Graph Data: Communication Networks

Internet
Graph Data: Technological Networks

Seven Bridges of Königsberg

[Euler, 1735]

Return to the starting point by traveling each link of the graph once and only once.
Web as a Graph

- Web as a directed graph:
  - Nodes: Webpages
  - Edges: Hyperlinks

I teach a class on data mining.

CS547: Classes are in the SIG building

Computer Science Department at UW

University of Washington
Web as a Graph

- Web as a directed graph:
  - Nodes: Webpages
  - Edges: Hyperlinks

I teach a class on data mining.

CS547: Classes are in the SIG building.

Computer Science Department at UW.

University of Washington.
Web as a Directed Graph
How to organize the Web?

First try: Human curated Web directories
- Yahoo, DMOZ, LookSmart

Second try: Web Search
- Information Retrieval investigates:
  Find relevant docs in a small and trusted set
  - Newspaper articles, Patents, etc.
- But: Web is huge, full of untrusted documents, random things, web spam, etc.
Web Search: 2 Challenges

2 challenges of web search:

- (1) Web contains many sources of information
  Who to “trust”?  
    - **Trick:** Trustworthy pages may point to each other!

- (2) What is the “best” answer to query “newspaper”?  
  - No single right answer  
  - **Trick:** Pages that actually know about newspapers might all be pointing to many newspapers
Ranking Nodes on the Graph

- All web pages are not equally “important”
  - thispersondoesnotexist.com vs. www.uw.edu

- There is a large diversity in the web-graph node connectivity.
  Let’s rank the pages by the link structure!
Link Analysis Algorithms

- We will cover the following Link Analysis approaches for computing importances of nodes in a graph:
  - Page Rank
  - Topic-Specific (Personalized) Page Rank
  - Web Spam Detection Algorithms
PageRank: The “Flow” Formulation
Links as Votes

- **Idea: Links as votes**
  - Page is more important if it has more links
    - In-coming links? Out-going links?
  - **Think of in-links as votes:**
    - [www.uw.edu](http://www.uw.edu) has **millions** in-links
    - [thispersondoesnotexist.com](http://thispersondoesnotexist.com) has a **few thousands** in-link
  - **Are all in-links equal?**
    - Links from important pages count more
    - Recursive question!
Web pages are important if people visit them a lot.

But we can’t watch everybody using the Web.

A good surrogate for visiting pages is to assume people follow links randomly.

Leads to *random surfer* model:

- Start at a random page and follow random out-links repeatedly, from whatever page you are at.
- $\text{PageRank} = \text{limiting probability of being at a page.}$
Intuition – (2)

- Solve the recursive equation: “importance of a page = its share of the importance of each of its predecessor pages”
  - Equivalent to the random-surfer definition of PageRank

- Technically, *importance* = the principal eigenvector of the transition matrix of the Web
  - A few fix-ups needed
Example: PageRank Scores
Simple Recursive Formulation

- Each link’s vote is proportional to the importance of its source page.
- If page \( j \) with importance \( r_j \) has \( n \) out-links, each link gets \( r_j/n \) votes.
- Page \( j \)'s own importance is the sum of the votes on its in-links.

\[
r_j = \frac{r_i}{3} + \frac{r_k}{4}
\]
PageRank: The “Flow” Model

- A “vote” from an important page is worth more
- A page is important if it is pointed to by other important pages
- Define a “rank” $r_j$ for page $j$

$\begin{align*}
   r_j &= \sum_{i \rightarrow j} \frac{r_i}{d_i} \\

   d_i &\ldots \text{out-degree of node } i
\end{align*}$

"Flow" equations:

$\begin{align*}
   r_y &= r_y/2 + r_a/2 \\
   r_a &= r_y/2 + r_m \\
   r_m &= r_a/2
\end{align*}$
Solving the Flow Equations

- **3 equations, 3 unknowns, no constants**
  - No unique solution
  - All solutions equivalent modulo the scale factor

- **Additional constraint forces uniqueness:**
  - \( r_y + r_a + r_m = 1 \)
  - **Solution:** \( r_y = \frac{2}{5}, \ r_a = \frac{2}{5}, \ r_m = \frac{1}{5} \)

- Gaussian elimination method works for small examples, but we need a better method for large web-size graphs

- **We need a new formulation!**
PageRank: Matrix Formulation

- **Stochastic adjacency matrix** \( M \)
  - Let page \( i \) has \( d_i \) out-links
  - If \( i \rightarrow j \), then \( M_{ji} = \frac{1}{d_i} \) else \( M_{ji} = 0 \)
    - \( M \) is a **column stochastic matrix**
      - Columns sum to 1

- **Rank vector** \( r \): vector with an entry per page
  - \( r_i \) is the importance score of page \( i \)
  - \( \sum_i r_i = 1 \)

- The flow equations can be written
  \[
  r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}
  \]
  \[
  r = M \cdot r
  \]
Example

- Remember the flow equation: \( r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i} \)
- Flow equation in the matrix form: \( M \cdot r = r \)
- Suppose page \( i \) links to 3 pages, including \( j \)

\[
M = \begin{pmatrix}
\text{1/3} \\
\vdots \\
\text{1/3}
\end{pmatrix}
\]

\[
\begin{pmatrix}
\text{1/3} \\
\vdots \\
\text{1/3}
\end{pmatrix} \cdot \begin{pmatrix}
r_i \\
r_i \\
r_i
\end{pmatrix} = \begin{pmatrix}
r_j \\
r_j \\
r_j
\end{pmatrix}
\]
Example: Flow Equations & M

\[
\begin{align*}
\text{r}_y &= \frac{\text{r}_y}{2} + \frac{\text{r}_a}{2} \\
\text{r}_a &= \frac{\text{r}_y}{2} + \frac{\text{r}_m}{2} \\
\text{r}_m &= \frac{\text{r}_a}{2}
\end{align*}
\]

\[
r = M \cdot r
\]

\[
\begin{bmatrix}
\text{r}_y \\
\text{r}_a \\
\text{r}_m
\end{bmatrix} =
\begin{bmatrix}
\frac{1}{2} & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & 1 \\
0 & \frac{1}{2} & 0
\end{bmatrix}
\begin{bmatrix}
\text{r}_y \\
\text{r}_a \\
\text{r}_m
\end{bmatrix}
\]
Eigenvector Formulation

- The flow equations can be written
  \[ r = M \cdot r \]
- So the rank vector \( r \) is an eigenvector of the stochastic web matrix \( M \)
  - Starting from any vector \( u \), the limit \( M(M(...M(M(u))) \) is the long-term distribution of the surfers.
    - The math: limiting distribution = principal eigenvector of \( M = \text{PageRank} \).
      - Note: If \( r \) satisfies the equation \( r = Mr \), then \( r \) is an eigenvector of \( M \) with eigenvalue 1
- We can now efficiently solve for \( r \)! The method is called Power iteration

\[ \text{NOTE: } x \text{ is an eigenvector with the corresponding eigenvalue } \lambda \text{ if: } Ax = \lambda x \]
Power Iteration Method

- Given a web graph with \( n \) nodes, where the nodes are pages and edges are hyperlinks
- **Power iteration:** a simple iterative scheme
  - Suppose there are \( N \) web pages
  - Initialize: \( r^{(0)} = [1/N, \ldots, 1/N]^T \)
  - Iterate: \( r^{(t+1)} = M \cdot r^{(t)} \)
  - Stop when \( |r^{(t+1)} - r^{(t)}|_1 < \varepsilon \)

\[
r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i}
\]

- \( d_i \) .... out-degree of node \( i \)

\[ |x|_1 = \sum_{1 \leq i \leq N} |x_i| \] is the \( L_1 \) norm
  - Can use any other vector norm, e.g., Euclidean

About 50 iterations is sufficient to estimate the limiting solution.
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:**

  $$
  \begin{pmatrix}
  r_y \\
  r_a \\
  r_m
  \end{pmatrix} = \begin{pmatrix}
  1/3 \\
  1/3 \\
  1/3
  \end{pmatrix}
  $$

  Iteration 0, 1, 2, …
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:**

$$
\begin{pmatrix}
r_y \\
r_a \\
r_m
\end{pmatrix} =
\begin{pmatrix}
1/3 & 1/3 & 5/12 & 9/24 & 6/15 \\
1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\
1/3 & 1/6 & 3/12 & 1/6 & \ldots & 3/15
\end{pmatrix}
$$

$\text{Iteration } 0, 1, 2, \ldots$

$$
\begin{array}{ccc}
\text{y} & \text{a} & \text{m} \\
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1 \\
m & 0 & 1/2 & 0 \\
\end{array}
$$

$$
\begin{align*}
r_y &= r_y/2 + r_a/2 \\
r_a &= r_y/2 + r_m \\
r_m &= r_a/2
\end{align*}
$$
Why Power Iteration works? (1)

- **Power iteration:**
  A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)
  - $r^{(1)} = M \cdot r^{(0)}$
  - $r^{(2)} = M \cdot r^{(1)} = M(Mr^{(1)}) = M^2 \cdot r^{(0)}$
  - $r^{(3)} = M \cdot r^{(2)} = M(M^2r^{(0)}) = M^3 \cdot r^{(0)}$

- **Claim:**
  Sequence $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \ldots M^k \cdot r^{(0)}, \ldots$ approaches the dominant eigenvector of $M$
Why Power Iteration works? (2)

- **Claim:** Sequence $M \cdot r^{(0)}$, $M^2 \cdot r^{(0)}$, ..., $M^k \cdot r^{(0)}$, ... approaches the dominant eigenvector of $M$

- **Proof:**
  
  - Assume $M$ has $n$ linearly independent eigenvectors, $x_1, x_2, ..., x_n$ with corresponding eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$, where $\lambda_1 > \lambda_2 > \cdots > \lambda_n$
  
  - Vectors $x_1, x_2, ..., x_n$ form a basis and thus we can write:
    
    $$ r^{(0)} = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n $$
    
    - $M r^{(0)} = M(c_1 x_1 + c_2 x_2 + \cdots + c_n x_n)$
    
    $$ = c_1 (M x_1) + c_2 (M x_2) + \cdots + c_n (M x_n) $$
    
    $$ = c_1 (\lambda_1 x_1) + c_2 (\lambda_2 x_2) + \cdots + c_n (\lambda_n x_n) $$

  - **Repeated multiplication on both sides produces**
    
    $$ M^k r^{(0)} = c_1 (\lambda_1^k x_1) + c_2 (\lambda_2^k x_2) + \cdots + c_n (\lambda_n^k x_n) $$
Why Power Iteration works? (3)

- **Claim:** Sequence $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \ldots M^k \cdot r^{(0)}, \ldots$ approaches the dominant eigenvector of $M$

- **Proof (continued):**
  - Repeated multiplication on both sides produces
    \[
    M^k r^{(0)} = c_1 (\lambda_1^k x_1) + c_2 (\lambda_2^k x_2) + \cdots + c_n (\lambda_n^k x_n)
    \]
  - \[
  M^k r^{(0)} = \lambda_1^k \left[ c_1 x_1 + c_2 \left( \frac{\lambda_2}{\lambda_1} \right)^k x_2 + \cdots + c_n \left( \frac{\lambda_n}{\lambda_1} \right)^k x_n \right]
  \]
  - Since $\lambda_1 > \lambda_2$ then fractions $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1}, \ldots < 1$
    and so $\left( \frac{\lambda_i}{\lambda_1} \right)^k = 0$ as $k \to \infty$ (for all $i = 2 \ldots n$).
  - **Thus:** $M^k r^{(0)} \approx c_1 (\lambda_1^k x_1)$
    - Note if $c_1 = 0$ then the method won’t converge
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:
- $p(t)$ ... 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
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
Existence and Uniqueness

- A central result from the theory of random walks (a.k.a. Markov processes):

For graphs that satisfy certain conditions, the stationary distribution is unique and eventually will be reached no matter what is the initial probability distribution at time $t = 0$. 
PageRank: The Google Formulation
PageRank: Three Questions

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

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

Example:

\[
\begin{align*}
   r_a &= \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix} \\
   r_b &= \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}
\end{align*}
\]

Iteration 0, 1, 2, …

\[
\begin{align*}
   r_j^{(t+1)} &= \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i}
\end{align*}
\]
Does it converge to what we want?

- Example:

  \[
  r_a = \begin{bmatrix}
  1 & 0 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 & 0
  \end{bmatrix}
  \]

  \[
  r_b = \begin{bmatrix}
  1 & 0 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 & 0
  \end{bmatrix}
  \]

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

- Iteration 0, 1, 2, …
PageRank: Problems

2 problems:

- (1) Dead ends: Some pages have no out-links
  - Random walk has “nowhere” to go to
  - Such pages cause importance to “leak out”

- (2) Spider traps:
  (all out-links are within the group)
  - Random walk gets “stuck” in a trap
  - And eventually spider traps absorb all importance
Problem: Spider Traps

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

- **Example:**

  $\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 & 2/6 & 3/12 & 5/24 & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & 1 \end{bmatrix}$

  All the PageRank score gets “trapped” in node $m$. 

  $r_y = r_y/2 + r_a/2$
  $r_a = r_y/2$
  $r_m = r_a/2 + r_m$
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
- $\beta$ is typically 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:**

$$
\begin{pmatrix}
  r_y \\
  r_a \\
  r_m
\end{pmatrix} =
\begin{pmatrix}
  1/3 & 2/6 & 3/12 & 5/24 & 0 \\
  1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\
  1/3 & 1/6 & 1/12 & 2/24 & 0
\end{pmatrix}
$$

Here the PageRank score “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

```
<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>½</td>
<td>½</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>½</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>½</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>½</td>
<td>½</td>
<td>⅓</td>
</tr>
<tr>
<td>a</td>
<td>½</td>
<td>0</td>
<td>⅓</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>½</td>
<td>⅓</td>
</tr>
</tbody>
</table>
```
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]

  $$r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

  $d_i$ ... out-degree of node $i$

  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]
  \[
  r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}
  \]

- **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 ($\beta = 0.8$)

\[
\begin{bmatrix}
1/2 & 1/2 & 0 \\
1/2 & 0 & 0 \\
0 & 1/2 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
13/15 \\
1/15 \\
1/15 \\
\end{bmatrix}
\]
How do we actually compute the PageRank?
Computing PageRank

- **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!

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

\[
\begin{bmatrix}
\frac{1}{2} & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & 0 \\
0 & \frac{1}{2} & 1 \\
\end{bmatrix}
+ \begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix}
= \begin{bmatrix}
7/15 & 7/15 & 1/15 \\
7/15 & 1/15 & 1/15 \\
1/15 & 7/15 & 13/15 \\
\end{bmatrix}
\]
Rearranging the Equation

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

**Note:** Here we assume \( \mathbf{M} \) has no dead-ends

\([x]_N \ldots \text{a vector of length } N \text{ with all entries } x\)
Sparse Matrix Formulation

- We just rearranged the PageRank equation

\[ r = \beta M \cdot r + \left[ \frac{1 - \beta}{N} \right] \]

- where \([(1-\beta)/N]_N\) is a vector with all \(N\) entries \((1-\beta)/N\)

- \(M\) is a sparse matrix! (with no dead-ends)
  - 10 links per node, approx 10\(N\) 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 \(\sum_j r_j^{new} < 1\) 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:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j |r_j^{new} - r_j^{old}| < \varepsilon$
    - $\forall j$: $r_j^{new} = \sum_{i \rightarrow j} \beta \frac{r_i^{old}}{d_i}$
      - $r_j^{new} = 0$ if in-degree of $j$ is 0
    - **Now re-insert the leaked PageRank:**
      - $\forall j$: $r_j^{new} = r_j^{new} + \frac{1-S}{N}$
        - **where:** $S = \sum_j r_j^{new}$
    - $r^{old} = r^{new}$

If the graph has no dead-ends then the amount of leaked PageRank is $1-\beta$. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing $S$. 
Some Problems with PageRank

- Measures generic popularity of a page
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (**next**)
- Uses a single measure of importance
  - Other models of importance
  - **Solution:** Hubs-and-Authorities
- Susceptible to Link spam
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank
Sparse Matrix Encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - Say 10N, or $4 \times 10 \times 1 \text{ billion} = 40 \text{GB}$
  - Still won’t fit in memory, but will fit on disk

<table>
<thead>
<tr>
<th>source node</th>
<th>degree</th>
<th>destination nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>17, 64, 113, 117, 245</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13, 23</td>
</tr>
</tbody>
</table>
Basic Algorithm: Update Step

- Assume enough RAM to fit $r^\text{new}$ into memory
  - Store $r^\text{old}$ and matrix $M$ on disk
- **1 step of power-iteration is:**
  
  Initialize all entries of $r^\text{new} = (1-\beta) / N$
  
  For each page $i$ (of out-degree $d_i$):
    - Read into memory: $i$, $d_i$, $\text{dest}_1$, ..., $\text{dest}_{d_i}$, $r^\text{old}(i)$
    - For $j = 1...d_i$
      - $r^\text{new}(\text{dest}_j) += \frac{\beta}{d_i} r^\text{old}(i)$

Assuming no dead ends
Analysis

- **Assume enough RAM to fit** $r^{new}$ **into memory**
  - Store $r^{old}$ and matrix $M$ on disk
- **In each iteration, we have to:**
  - Read $r^{old}$ and $M$
  - Write $r^{new}$ back to disk
- **Cost per iteration of Power method:**
  $= 2|r| + |M|$

- **Question:**
  - What if we could not even fit $r^{new}$ in memory?
Block-based Update Algorithm

- Break $r^{new}$ into $k$ blocks that fit in memory
- Scan $M$ and $r^{old}$ once for each block
Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $r^{\text{new}}$ into $k$ blocks that fit in memory
  - Scan $M$ and $r^{\text{old}}$ once for each block
- Total cost:
  - $k$ scans of $M$ and $r^{\text{old}}$
  - Cost per iteration of Power method:
    \[ k(|M| + |r|) + |r| = k|M| + (k + 1)|r| \]
- 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

Break $M$ into stripes! Each stripe contains only destination nodes in the corresponding block of $r^{new}$.
Block-Stripe Analysis

- Break $M$ into stripes
  - Each stripe contains only destination nodes in the corresponding block of $r^{\text{new}}$
- Some additional overhead per stripe
  - But it is usually worth it
- Cost per iteration of Power method:
  \[ |M| (1 + \varepsilon) + (k + 1) |r| \]
Some Problems with PageRank

- **Measures generic popularity of a page**
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (next)

- **Uses a single measure of importance**
  - Other models of importance
  - **Solution:** Hubs-and-Authorities

- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank