CS425: Algorithms for Web Scale Data

Lecture 1: PageRank Formulation

Most of the slides are from the Mining of Massive Datasets book. These slides have been modified for CS425. The original slides can be accessed at: www.mmds.org
Lecture Overview

- Graph data overview
- Problems with early search engines
- PageRank Model
  - Flow Formulation
  - Matrix Interpretation
  - Random Walk Interpretation
  - Google’s Formulation
- How to Compute PageRank
Facebook social graph
4-degrees of separation [Backstrom-Boldi-Rosa-Ugander-Vigna, 2011]
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 Nets

Internet

domain1

domain2

domain3

router
Web as a Directed Graph

I'm a student at Univ. of X

My song lyrics

I teach at Univ. of X

Classes

Networks

Networks class blog

Blog post about Company Z

Blog post about college rankings

I'm applying to college

USNews College Rankings

USNews Featured Colleges
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
Early Search Engines

- Inverted index
  - Data structure that return pointers to all pages a term occurs

- Which page to return first?
  - Where do the search terms appear in the page?
  - How many occurrences of the search terms in the page?

- What if a spammer tries to fool the search engine?
Fooling Early Search Engines

- Example: A spammer wants his page to be in the top search results for the term “movies”.

- **Approach 1:**
  - Add thousands of copies of the term “movies” to your page.
  - Make them invisible.

- **Approach 2:**
  - Search the term “movies”.
  - Copy the contents of the top page to your page.
  - Make it invisible.

- **Problem:** Ranking only based on page contents
- Early search engines almost useless because of spam.
Google’s Innovations

- **Basic idea:** Search engine believes what other pages say about you instead of what you say about yourself.

- **Main innovations:**
  1. Define the importance of a page based on:
     - How many pages point to it?
     - **How important are those pages?**
  2. Judge the contents of a page based on:
     - Which terms appear in the page?
     - **Which terms are used to link to the page?**
All web pages are not equally “important”

www.joe-schmoe.com vs. www.stanford.edu

There is large diversity in the web-graph node connectivity.

Let’s rank the pages by the link structure!
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
Think of in-links as votes:
- www.stanford.edu has 23,400 in-links
- www.joe-schmoe.com has 1 in-link

Are all in-links are equal?
- Links from important pages count more
- Recursive question!
Example: PageRank Scores

- 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 $j$

$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$$

$d_i$ ... out-degree of node $i$
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: The Matrix Formulation
PageRank: Matrix Formulation

- **Adjacency matrix** $\mathbf{M}$
  - Let page $i$ have $d_i$ out-links
  - If $i \to j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$

- **Rank vector** $\mathbf{r}$: 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
  \[ \mathbf{r} = \mathbf{M} \cdot \mathbf{r} \]
Example: Flow Equations & M

\[ r_y = \frac{r_y}{2} + \frac{r_a}{2} \]
\[ r_a = \frac{r_y}{2} + r_m \]
\[ r_m = \frac{r_a}{2} \]

\[
\begin{array}{ccc}
  \ y & \ a & \ m \\
  \frac{1}{2} & \frac{1}{2} & 0 \\
  \frac{1}{2} & 0 & 1 \\
  0 & \frac{1}{2} & 0 \\
\end{array}
\]

\[ r = M \cdot r \]
Example

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

\[
\begin{align*}
M \cdot r &= r \\
\sum_{i \to j} \frac{r_i}{d_i} &= r_j
\end{align*}
\]
Exercise: Matrix Formulation

\[
\begin{pmatrix}
0 & 1/2 & 1 & 0 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 1/2 & 0 & 0
\end{pmatrix}
\cdot
\begin{pmatrix}
r_A \\
r_B \\
r_C \\
r_D
\end{pmatrix}
= 
\begin{pmatrix}
r_A \\
r_B \\
r_C \\
r_D
\end{pmatrix}
\]
A is a column stochastic matrix iff each of its columns add up to 1 and there are no negative entries.

Our adjacency matrix $M$ is column stochastic. Why?

If there exist a vector $x$ and a scalar $\lambda$ such that $Ax = \lambda x$, then:

- $x$ is an eigenvector and $\lambda$ is an eigenvalue of $A$

- The principal eigenvector is the one that corresponds to the largest eigenvalue.

The largest eigenvalue of a column stochastic matrix is 1.

$Ax = x$, where $x$ is the principal eigenvector
Eigenvector Formulation

- PageRank flow formulation:
  \[ r = M \cdot r \]

- So the rank vector \( r \) is an eigenvector of the stochastic web matrix \( M \)
  - In fact, its first or principal eigenvector, with corresponding eigenvalue 1

- We can now efficiently solve for \( r \)!
The method is called Power iteration

NOTE: \( x \) is an eigenvector with the corresponding eigenvalue \( \lambda \) 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: $\mathbf{r}^{(0)} = [1/N, \ldots, 1/N]^T$
  - Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

$\mathbf{x}_1 = \sum_{1 \leq i \leq N} |x_i|$ is the $L_1$ norm
Can use any other vector norm, e.g., Euclidean

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

\[
\begin{bmatrix}
r_y \\
r_a \\
r_m
\end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}
\]

Iteration 0, 1, 2, …

\[
\begin{align*}
r_y &= r_y / 2 + r_a / 2 \\
r_a &= r_y / 2 + r_m \\
r_m &= r_a / 2
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1/2</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1/2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

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

Iteration 0, 1, 2, …

\[
\begin{array}{c|c|c|c}
& y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1 \\
m & 0 & 1/2 & 0 \\
\end{array}
\]

\[
\begin{align*}
r_y &= r_y / 2 + r_a / 2 \\
r_a &= r_y / 2 + r_m \\
r_m &= r_a / 2
\end{align*}
\]
**Power iteration:**

A method for finding principal eigenvector (the vector corresponding to the largest eigenvalue)

- $\mathbf{r}^{(1)} = \mathbf{M} \cdot \mathbf{r}^{(0)}$
- $\mathbf{r}^{(2)} = \mathbf{M} \cdot \mathbf{r}^{(1)} = \mathbf{M} (\mathbf{Mr}^{(1)}) = \mathbf{M}^2 \cdot \mathbf{r}^{(0)}$
- $\mathbf{r}^{(3)} = \mathbf{M} \cdot \mathbf{r}^{(2)} = \mathbf{M} (\mathbf{M}^2 \mathbf{r}^{(0)}) = \mathbf{M}^3 \cdot \mathbf{r}^{(0)}$

**Claim:**

Sequence $\mathbf{M} \cdot \mathbf{r}^{(0)}, \mathbf{M}^2 \cdot \mathbf{r}^{(0)}, ... \mathbf{M}^k \cdot \mathbf{r}^{(0)}, ...$ approaches the dominant eigenvector of $\mathbf{M}$
PageRank: Random Walk Interpretation
Random Walk Interpretation of PageRank

- Consider a web surfer:
  - He starts at a random page
  - He follows a random link at every time step
  - After a sufficiently long time:
    - What is the probability that he is at page j?
    - This probability corresponds to the page rank of j.
Example: Random Walk

Time $t = 0$: Assume the random surfer is at A.

Time $t = 1$:

- $p(A, 1) = ? \quad 0$
- $p(B, 1) = ? \quad 1/3$
- $p(C, 1) = ? \quad 1/3$
- $p(D, 1) = ? \quad 1/3$
Example: Random Walk

Time t = 1:

\[ p(B, 1) = 1/3 \]
\[ p(C, 1) = 1/3 \]
\[ p(D, 1) = 1/3 \]

Time t=2:

\[ p(A, 2) = ? \]

\[ p(A, 2) = p(B, 1) \cdot p(B \rightarrow A) + p(C, 1) \cdot p(C \rightarrow A) \]
\[ = 1/3 \cdot 1/2 + 1/3 \cdot 1 = 3/6 \]
Example: Transition Matrix

\[ p(A, t+1) = p(B, t) \cdot p(B \rightarrow A) + p(C, t) \cdot p(C \rightarrow A) \]

\[ p(C, t+1) = p(A, t) \cdot p(A \rightarrow C) + p(D, t) \cdot p(D \rightarrow C) \]
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
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

---

*Rank of page $j$ = Probability that the surfer is at page $j$ after a long random walk*
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 the initial probability distribution at time $t = 0$. 
Summary So Far

- PageRank formula: 
  \[ r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i} \]
  \( d_i \) .... out-degree of node i

- Iterative algorithm:
  1. Initialize rank of each page to 1/N (where N is the number of pages)
  2. Compute the next page rank values using the formula above
  3. Repeat step 2 until the page rank values do not change much

- Same algorithm, but different interpretations
Summary So Far (cont’d)

- Eigenvector interpretation:
  - Compute the principal eigenvector of stochastic adjacency matrix $M$
    \[ r = M \cdot r \]
  - Power iteration method

- Random walk interpretation:
  - Rank of page $i$ is the probability that a surfer is at $i$ after random walk
    \[ p(t+1) = M \cdot p(t) \]
  - Guaranteed to converge to a unique solution under certain conditions
Convergence Conditions

- To guarantee convergence to a meaningful and unique solution, the transition matrix must be:
  1. Column stochastic
  2. Irreducible
  3. Aperiodic
Column Stochastic

- Column stochastic:
  - All values in the matrix are non-negative
  - Sum of each column is 1

What if we remove the edge $m \rightarrow a$?

No longer column stochastic

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
</tbody>
</table>

$$r_y = \frac{r_y}{2} + \frac{r_a}{2}$$

$$r_a = \frac{r_y}{2} + r_m$$

$$r_m = \frac{r_a}{2}$$
Irreducible

- Irreducible: From any state, there is a non-zero probability of going to another.
  - Equivalent to: Strongly connected graph

What if we remove the edge C → A?

No longer irreducible.
Aperiodic

- State $i$ has **period $k$** if any return to state $i$ must occur in *multiples of $k$ time steps*.

- If $k = 1$ for a state, it is called **aperiodic**.
  - Returning to the state at irregular intervals

- A **Markov chain** is aperiodic if all its states are aperiodic.
  - If Markov chain is irreducible, one aperiodic state means all stated are aperiodic.

A Markov chain is aperiodic if all its states are aperiodic.

If Markov chain is irreducible, one aperiodic state means all stated are aperiodic.

How to make this aperiodic?

*Add any new edge*
PageRank: The Google Formulation
PageRank: Three Questions

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

or equivalently

\[ r = Mr \]

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

Example:

\[
\begin{align*}
\mathbf{r}^a &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \\
\mathbf{r}^b &= \\
\end{align*}
\]

Iteration 0, 1, 2, …

\[
r_{j}^{(t+1)} = \sum_{i \rightarrow j} \frac{r_{i}^{(t)}}{d_{i}}
\]
Does it converge to what we want?

Example:

\begin{align*}
\mathbf{r}_a &= \begin{pmatrix} 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \end{pmatrix} \\
\mathbf{r}_b &= \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}
\end{align*}

Iteration 0, 1, 2, …

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

2 problems:

- (1) Some pages are **dead ends** (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 \( r_j = 1/N \)
  - \( r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i} \)
    - And iterate

- **Example:**

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

Iteration 0, 1, 2, …

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

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1/2</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>

\( r_y = \frac{r_y}{2} + \frac{r_a}{2} \)
\( r_a = \frac{r_y}{2} \)
\( r_m = \frac{r_a}{2} + r_m \)
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
- 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:**

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

  Iteration 0, 1, 2, …

  Here the PageRank “leaks” out since the matrix is not stochastic.

```
\[
\begin{array}{ccc}
  y & a & m \\
  \hline
  y & 1/2 & 1/2 & 0 \\
  a & 1/2 & 0 & 0 \\
  m & 0 & 1/2 & 0 \\
\end{array}
\]

$\mathbf{r}_y = \mathbf{r}_y / 2 + \mathbf{r}_a / 2$

$\mathbf{r}_a = \mathbf{r}_y / 2$

$\mathbf{r}_m = \mathbf{r}_a / 2$

**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>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>(\frac{1}{2})</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>(\frac{1}{2})</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>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{3})</td>
</tr>
<tr>
<td>a</td>
<td>(\frac{1}{2})</td>
<td>0</td>
<td>(\frac{1}{3})</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{3})</td>
</tr>
</tbody>
</table>
Why Teleports Solve the Problem?

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-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
  - **Solution**: Make matrix column stochastic by always teleporting when there is nowhere else to go
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]

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

- **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 ($\beta = 0.8$)

\[
\begin{align*}
\text{y} & = 1/3 
\begin{bmatrix}
0.33 & 0.24 & 0.26 \\
0.20 & 0.20 & 0.18 \\
0.46 & 0.52 & 0.56 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{a} & = 1/3 
\begin{bmatrix}
0.20 & 0.20 & 0.18 \\
0.20 & 0.20 & 0.18 \\
0.46 & 0.52 & 0.56 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{m} & = 1/3 
\begin{bmatrix}
0.33 & 0.24 & 0.26 \\
0.20 & 0.20 & 0.18 \\
0.46 & 0.52 & 0.56 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{A} & = 
\begin{bmatrix}
1/2 & 1/2 & 0 \\
1/2 & 0 & 0 \\
0 & 1/2 & 1 \\
\end{bmatrix} 
+ 0.2
\end{align*}
\]

\[
\begin{align*}
\text{M} & = 
\begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{[1/N]}_{N \times N} & = 
\begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix}
\] \quad \text{y} \\
\text{a} & = 7/15 
\begin{bmatrix}
1/15 & 0 \\
1/15 & 0 \\
1/15 & 0 \\
\end{bmatrix} \\
\text{m} & = 1/15 
\begin{bmatrix}
7/15 & 7/15 & 13/15 \\
1/15 & 1/15 & 1/15 \\
7/15 & 7/15 & 13/15 \\
\end{bmatrix}
\end{align*}
\]
Suppose there are \( N \) pages

Consider page \( i \), with \( d_i \) out-links

We have \( M_{ji} = 1/|d_i| \) when \( i \rightarrow j \) and \( M_{ji} = 0 \) otherwise

The random teleport is equivalent to:

- Adding a **teleport link** from \( i \) to every other page and setting transition probability to \((1-\beta)/N\)
- Reducing the probability of following each out-link from \( 1/|d_i| \) to \( \beta/|d_i| \)
- **Equivalent:** Tax each page a fraction \((1-\beta)\) of its score and redistribute evenly
How do we actually compute the 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!

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

\[
\begin{pmatrix}
\frac{1}{2} & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & 0 \\
0 & \frac{1}{2} & 1
\end{pmatrix} + 0.2 \\
= \\
\begin{pmatrix}
\frac{7}{15} & \frac{7}{15} & \frac{1}{15} \\
\frac{7}{15} & \frac{1}{15} & \frac{1}{15} \\
\frac{1}{15} & \frac{7}{15} & \frac{13}{15}
\end{pmatrix}
\]
Matrix Sparseness

- Reminder: Our original matrix was sparse.
  - On average: ~10 out-links per vertex
  - # of non-zero values in matrix M: ~10N
- Teleport links make matrix M dense.
- Can we convert it back to the sparse form?

Original matrix without teleports:

\[
\begin{pmatrix}
0 & 1/2 & 1 & 0 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 1/2 & 0 & 0
\end{pmatrix}
\]
Rearranging the Equation

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

**Note:** Here we assumed \( M \) has no dead-ends
Example: Equation with Teleports

Note: Here we assumed $M$ has no dead-ends
We just rearranged the **PageRank equation**

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

- where \([ (1-\beta)/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 \( r^{\text{new}} = \beta M \cdot r^{\text{old}} \)
  - Add a constant value \( (1-\beta)/N \) to each entry in \( r^{\text{new}} \)
  - Note if \( M \) contains dead-ends then \( \sum_j r_j^{\text{new}} < 1 \) and we also have to renormalize \( r^{\text{new}} \) so that it sums to 1
- **Input:** Graph $G$ and parameter $\beta$
  - Directed graph $G$ (cannot have dead ends)
  - Parameter $\beta$
- **Output:** PageRank vector $r^{new}$

- **Set:** $r_j^{old} = \frac{1}{N}$

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

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 $S$. 

Sparse Matrix Encoding: First Try

Store a triplet for each nonzero entry: (row, column, weight)

\[\begin{array}{ccc}
0 & 1/2 & 1 \\
1/3 & 0 & 0 \\
1/3 & 0 & 1/2 \\
1/3 & 1/2 & 0 \\
\end{array}\] …

Assume 4 bytes per integer and 8 bytes per float: 16 bytes per entry

*Inefficient:* Repeating the column index and weight multiple times
**Sparse Matrix Encoding**

- **Store entries per source node**
  - Source index and degree stored once per node
  - 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

<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** $r^{new}$ **into memory**
  - Store $r^{old}$ and matrix $M$ on disk
- **1 step of power-iteration is:**

  - **Initialize** all entries of $r^{new} = (1-\beta) / N$
  - For each page $i$ (of out-degree $d_i$):
    - Read into memory: $i$, $d_i$, $dest_1$, …, $dest_{d_i}$, $r^{old}(i)$
    - For $j = 1…d_i$
      - $r^{new}(dest_j) += \beta \ r^{old}(i) / d_i$

<table>
<thead>
<tr>
<th>source</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1, 5, 6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>17, 64, 113, 117</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13, 23</td>
</tr>
</tbody>
</table>
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?
- Break $r^{\text{new}}$ into $k$ blocks that fit in memory
- Scan $M$ and $r^{\text{old}}$ once for each block
Block-based Update Algorithm

- Break $r^{\text{new}}$ into $k$ blocks that fit in memory
- Scan $M$ and $r^{\text{old}}$ once for each block
Break $r^{\text{new}}$ into $k$ blocks that fit in memory
Scan $M$ and $r^{\text{old}}$ once for each block

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1, 3, 5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0, 5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3, 4</td>
</tr>
</tbody>
</table>
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 $r^{\text{new}}$

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

**Note:**
- $r^{\text{new}}$: New block representation
- $r^{\text{old}}$: Old block representation

---

### Block-Stripe Update Algorithm

**Break \( M \) into stripes!** Each stripe contains only destination nodes in the corresponding block of \( r^{new} \).

<table>
<thead>
<tr>
<th></th>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

\( r^{new} \)

\[\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}\]

\( r^{old} \)

\[\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}\]

**Block-Stripe Update Algorithm**

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

*Break $M$ into stripes!* Each stripe contains only destination nodes in the corresponding block of $r_{\text{new}}$.
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 it is usually worth it
- Cost per iteration of Power method:
  $$= |M|(1+\varepsilon) + (k+1)|r|$$
Some Problems with Page Rank

- 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