The PageRank Algorithm

Cesar O. Aguilar

SUNY Geneseo
Department of Mathematics

April 30, 2021

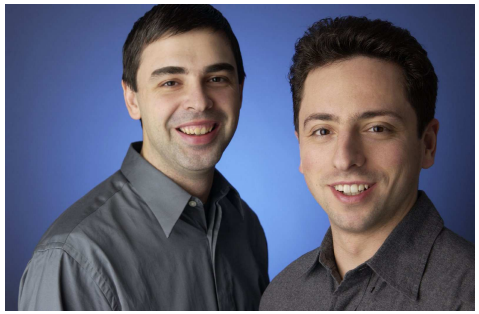
Search Engines

- Internet search engines perform a two-stage process:
  - **Stage 1**: Traditional text processing
  - **Stage 2**: The pages are sorted and displayed based on a pre-computed **ranking** that is query-independent
- How Google search works:
  https://www.youtube.com/watch?v=Md7K90FfJhg
- The pre-computed ranking is based on the **hyperlink** structure of the web
- **Basic idea of ranking method**: If many pages link to page P_i then P_i must be an important page and thus will have a high numerical rank
- The rank of each page is called its **PageRank**, named after Larry Page one of the founders of Google
- Before the mid 1990's, traditional text processing was the predominant method

PageRank and HITS

- In 1998, Jon Kleinberg from IBM (now a CS professor at Cornell) presented the **HITS** algorithm (Hyperlink-Induced Topic Search)
- At Stanford, doctoral students Sergey Brin and Larry Page were busy working on a similar project which they had begun in 1995

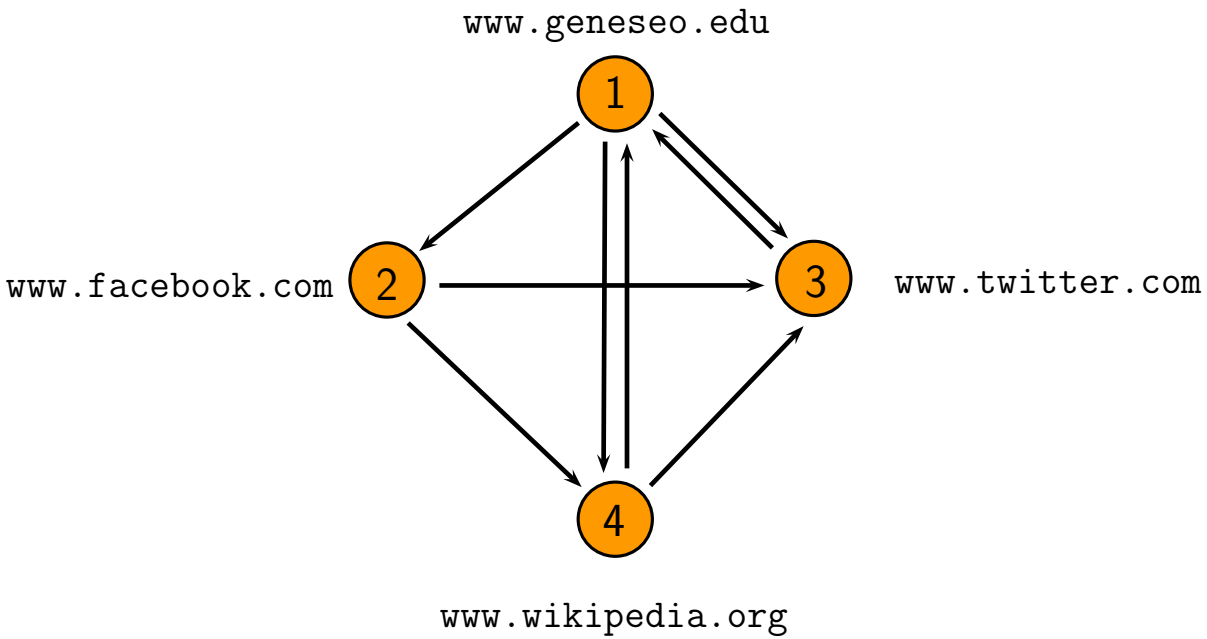
  "In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/ ."



Larry Page and Sergey Brin

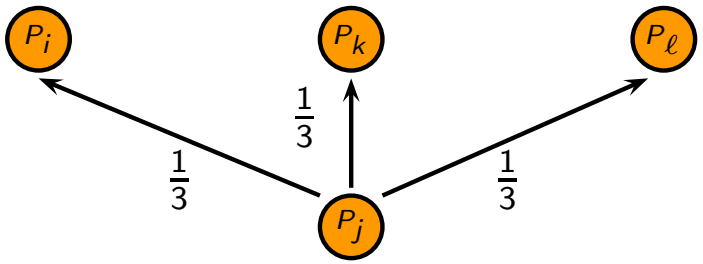
PageRank and HITS

- In both algotihms, the web is modeled as a **directed graph**
- Vertices represent webpages
- The directed edges represent hyperlinks
- Goal of algorithms: Determine important pages on a particular topic based on the hyperlink structure



A Description of the PageRank Algorithm

- The purpose of the PageRank algorithm is to produce a numerical rank $0 < x_i \leq 1$ for each page P_i
- Each in-link is viewed as a recommendation or vote
- The vote of each page is equally distributed among all its out-going links



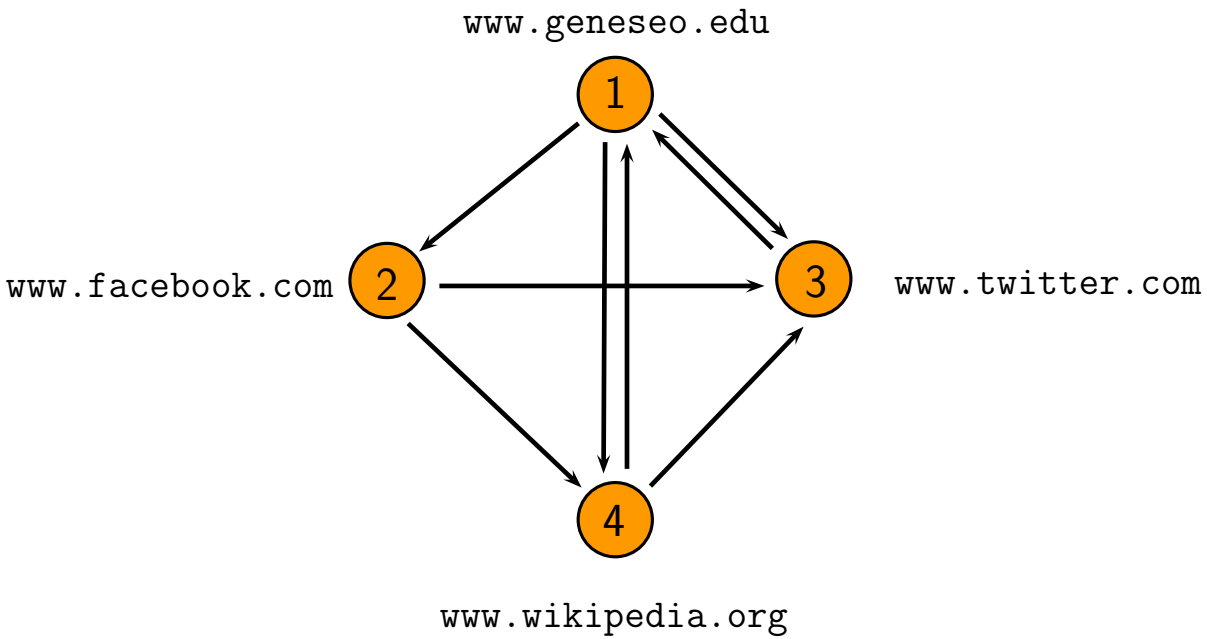
- The PageRank x_i of page P_i is:

$$x_i = \sum_{j \to i} \frac{x_j}{|N_j|}$$

  - N_j is the number of out-links from page j
  - $j \to i$ means page j links to page i

- Do this for each x_i and get a set of equations for all the PageRanks

Google's PageRank Algorithm



- For page $i = 1$: $x_1 = \frac{x_3}{1} + \frac{x_4}{2}$
- For page $i = 2$: $x_2 = \frac{x_1}{3}$
- For page $i = 3$: $x_3 = \frac{x_1}{3} + \frac{x_2}{2} + \frac{x_4}{2}$
- For page $i = 4$: $x_4 = \frac{x_1}{3} + \frac{x_2}{2}$

Google's PageRank Algorithm

- The equations for the PageRanks:

$$
\begin{aligned}
x_1 &= x_3 + \tfrac{1}{2}x_4 \\
x_2 &= \tfrac{1}{3}x_1 \\
x_3 &= \tfrac{1}{3}x_1 + \tfrac{1}{2}x_2 + \tfrac{1}{2}x_4 \\
x_4 &= \tfrac{1}{3}x_1 + \tfrac{1}{2}x_2
\end{aligned}
$$

- We can write all four equations in matrix form:

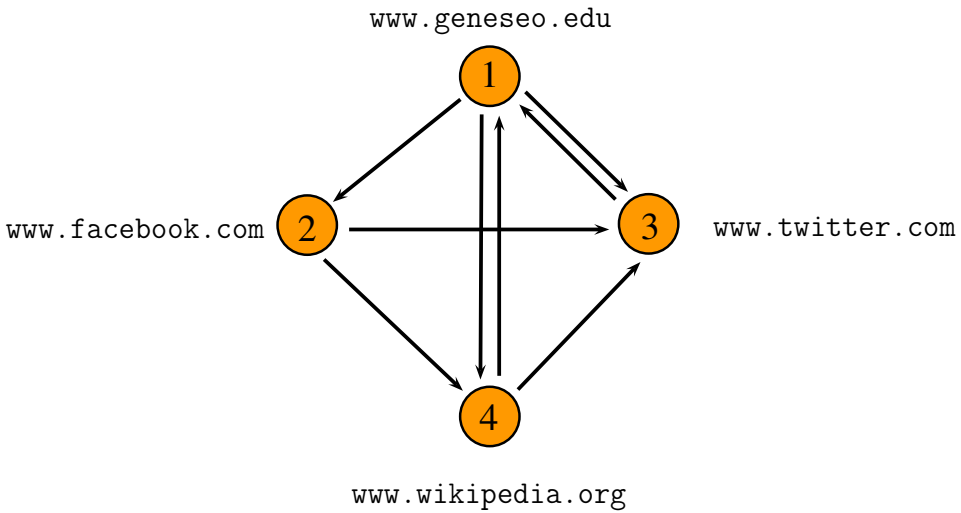
$$
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}
$$

- This is a linear equation for the unknown vector $\mathrm{x} = (x_1, x_2, x_3, x_4)$:

$$
\mathrm{x} = \mathrm{A} \cdot \mathrm{x}
$$

- This is an eigenvalue-eigenvector problem

Example



$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

- $\lambda = 1$ is an eigenvalue with corresponding PageRank eigenvector

$$x^* = (0.721, 0.240, 0.540, 0.360)$$

- The ranking of the webpages is determined by ordering the entries of x^*:

0.721	0.540	0.360	0.240
P_1	P_3	P_4	P_2
`geneseo.edu`	`twitter.com`	`wikipedia.org`	`facebook.com`

Google's PageRank Algorithm

- For large matrices, finding eigenvectors is not easy
- In 2006, $n = 8.1 \times 10^9$ (8.1 billion) and directly solving the equations is not feasible
- Instead, an iterative method called the **Power Method** is used
- One starts with an initial guess, say $x_0 = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$
- Then one updates the guess by computing

$$x_1 = A \cdot x_0$$

- One updates again

$$x_2 = A \cdot x_1 = A^2 \cdot x_0$$

- And continue iterating:

$$x_k = A^k \cdot x_0$$

- What happens in the limit?

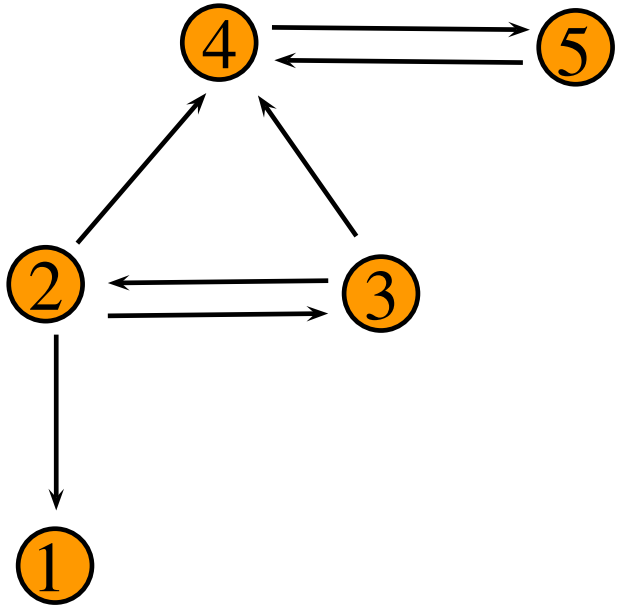
$$\lim_{k \to \infty} (A^k \cdot x_0)$$

Google's PageRank Algorithm

Some natural questions and concerns:

- Under what conditions will $\lim_{k\to\infty}(A^k \cdot x_0)$ exist and be unique?
- If $\lim_{k\to\infty}(A^k \cdot x_0)$ exists, will it be a positive vector?
- Can x_0 be chosen arbitrarily?
- If n is very large (and it is!), computing $A^k \cdot x_0$ is not going to be easy
- Each matrix-vector multiplication requires n^2 multiplications
- Storing A is not going to be easy; for example, if $n = 24 \times 10^6$ need approximately 4191 TB to store A
- Let's first deal with the math and then with the implementation

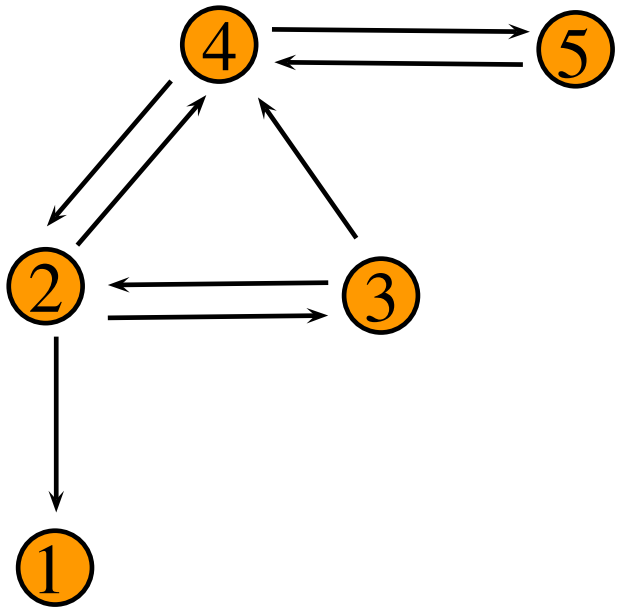
Example



$$A = \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{2} & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Starting with $x_0 = (\frac{1}{5}, \ldots, \frac{1}{5})$ we obtain that for $k \geq 39$, the vectors $x_k = A^k x_0$ cycle between $(0, 0, 0, 0.28, 0.40)$ and $(0, 0, 0, 0.40, 0.28)$
- Why? Nodes 4 and 5 form a **cycle** (internet is not **strongly connected**)

Example



$$A = \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{2} & 0 & 1 \\ 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

- If we adjust for the cycle node 1 is still a **dangling node** (e.g. a node with no out-links)
- Starting with $x_0 = (\frac{1}{5}, \ldots, \frac{1}{5})$ results in $x_k \to 0$

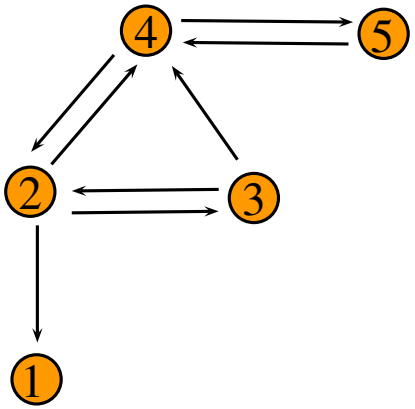
Adjustments Made by Brin and Page

- **Adjustment 1**: To deal with a dangling node, replace the associated zero-column with the vector $\frac{1}{n}\mathrm{e} = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$
- Justification: The node now links to all pages and so recommends all nodes equally
- **Adjustment 2**: A surfer may abandon the hyperlink structure of the web by ocassionally moving to a random page by typing its address in the browser (fixes the connectivity issue)
- With these adjustments, the new transition matrix becomes

$$\mathrm{G} = \alpha\overline{\mathrm{A}} + (1-\alpha)\frac{1}{n}\mathrm{J}$$

- $\overline{\mathrm{A}}$: adjustment for dangling nodes; $\overline{\mathrm{A}} = \mathrm{A} + \frac{1}{n}\mathrm{e}\cdot\mathrm{a}^T$
- $(1-\alpha)\frac{1}{n}\mathrm{J}$: adjustment for lack of connectivity
- G is called the Google matrix, and Google uses $\alpha = 0.85$

Example: $G = \alpha \overline{A} + (1-\alpha)\frac{1}{n} J$



$$A = \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{2} & 0 & 1 \\ 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

$$G = \alpha \underbrace{\begin{bmatrix} \frac{1}{5} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{5} & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{5} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{5} & \frac{1}{3} & \frac{1}{2} & 0 & 1 \\ \frac{1}{5} & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}}_{\overline{A}} + \frac{(1-\alpha)}{5} \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}}_{J}$$

Mathematical Justifications for the Adjustments

- G is **positive** and **column-stochastic** matrix
- $\lambda = 1$ is an eigenvalue of G and all eigenvalues have magnitude ≤ 1
- Apply a theorem in matrix analysis (Perron-Frobenius):

Theorem. For the Google matrix G the following hold:

(i) $\lambda = 1$ is the largest (in magnitude) eigenvalue and it is simple.
(ii) There is a unique real and positive vector x^* such that $G \cdot x^* = x^*$ and whose entries sum to one.
(iii) For any initial probability vector x_0 it holds that

$$\lim_{k \to \infty} G^k x_0 = x^*$$

The vector x^* is known as the PageRank vector

Computational and Storage Issues

- Power Method: need to compute $G \cdot x$ many times
- G is completely dense and so $G \cdot x$ requires n^2 multiplications
- Fortunately, we can decompose G:

$$G = \alpha A + \frac{\alpha}{n} e \cdot a^T + \frac{(1-\alpha)}{n} e \cdot e^T$$

- To compute $G \cdot x$ we really just need to compute:

$$\text{(i) } A \cdot x \qquad \text{(ii) } a^T x = \sum_{i=1}^{n} a_i x_i \qquad \text{(iii) } e^T x = \sum_{i=1}^{n} x_i$$

- The average webpage has about 10 outlinks, so A has about $10n$ non-zero entries
- A is a **sparse** matrix ($n = 24 \times 10^6$ need ≈ 5.4 GB to store A)
- This means that computing $A \cdot x$ reduces to about $10n$ multiplications!!!! And $10n$ is **A LOT LESS** than n^2 when n is big

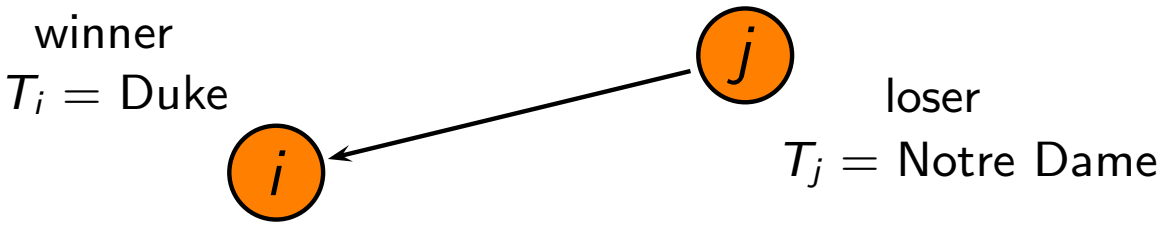
Other Applications of PageRank

The PageRank algorithm is one of several used to determine which vertices in a network are the most important (**node centrality**)

- **Biology**: GeneRank, ProteinRank
- **Engineering**: MonitorRank (debugging), Linux kernel (dependencies between functions), traffic flow and human movement
- **Bibliometrics**: CiteRank (journals), AuthorRank (authors)
- **Recommender systems**: ItemRank (Netflix, Amazon)
- **Sports**: FIFA Soccer, **NCAA Division I Football**
  - The only NCAA-sponsored sport without an officially organized tournament to determine its champion (March Madness)
  - Bowl Championship Series (BCS): Selection system in place from 1998-2013 to select top 2 teams for championship game
  - Used AP Poll, USA Today Poll, and computer rankings by experts
  - Favored 6 conferences resulting in several controversies and lawsuits

College Football Team Ranking

- Teams now ranked by a 13-member committee (CFP)
- At the end of the season, the top 4 ranked teams compete in a playoff tournament to determine the National Champion
- PageRank has been proposed to rank the teams
- The setup:
  - **Vertices:** Teams $= \{T_1, T_2, T_3, \ldots, T_{128}\}$
  - **Arcs:** From j to i if team T_j losses to T_i



- If $s(i,j) = \text{Score}(T_i) - \text{Score}(T_j)$, the arcs will have weights:

$$A(i,j) = \frac{s(i,j)}{\sum_{j \to k} s(k,j)}$$

College Football Team Ranking ($n = 128$)

Rank	Page	Record
1.	Clemson	13-1
2.	Pittsburgh	7-5
3.	Alabama	13-1
4.	Miami (Florida)	8-4
5.	Virginia Tech	9-4
6.	Ohio State	11-2
7.	Michigan	10-3
8.	Oklahoma	11-2
9.	Tennessee	8-4
10.	Oklahoma State	9-3
11.	Louisville	9-4
12.	Wisconsin	11-3
13.	North Carolina	6-5
14.	Houston	8-4
15.	Penn State	11-3
16.	Washington	11-2
17.	Northwestern	7-5
18.	Florida State	9-3
19.	Southern California	10-3
20.	Louisiana State	7-4

College Football Team Ranking ($n = 128$)

Rank	Page	Record
1.	Clemson	13-1
2.	Pittsburgh	7-5
3.	Alabama	13-1
4.	Miami (Florida)	8-4
5.	Virginia Tech	9-4
6.	Ohio State	11-2
7.	Michigan	10-3
8.	Oklahoma	11-2
9.	Tennessee	8-4
10.	Oklahoma State	9-3
11.	Louisville	9-4
12.	Wisconsin	11-3
13.	North Carolina	6-5
14.	Houston	8-4
15.	Penn State	11-3
16.	Washington	11-2
17.	Northwestern	7-5
18.	Florida State	9-3
19.	Southern California	10-3
20.	Louisiana State	7-4

- Pittsburgh only team to beat Clemson (43-42)
- Miami beat Pittsburgh (51-28)
- Virginia Tech beat Miami (37-16)
- Tennessee beat Virginia Tech (45-24)

Clemson losing to Pittsburgh is like `facebook.com` having a link to your personal website

College Football Team Ranking ($n = 128$)

What if Pittsburgh hadn't upset Clemson?

Rank	Page	Record
1.	Clemson	13-1
2.	Pittsburgh	7-5
3.	Alabama	13-1
4.	Miami (Florida)	8-4
5.	Virginia Tech	9-4
6.	Ohio State	11-2
7.	Michigan	10-3
8.	Oklahoma	11-2
9.	Tennessee	8-4
10.	Oklahoma State	9-3
11.	Louisville	9-4
12.	Wisconsin	11-3
13.	North Carolina	6-5
14.	Houston	8-4
15.	Penn State	11-3
16.	Washington	11-2
17.	Northwestern	7-5
18.	Florida State	9-3
19.	Southern California	10-3
20.	Louisiana State	7-4

Rank	Page	Record
1.	Clemson	14-0
2.	Alabama	13-1
3.	Ohio State	11-2
4.	Michigan	10-3
5.	Oklahoma	11-2
6.	Washington	11-2
7.	Louisville	9-4
8.	Penn State	11-3
9.	Wisconsin	11-3
10.	Houston	8-4
11.	Virginia Tech	9-4
12.	Southern California	10-3
13.	Florida State	9-3
14.	Louisiana State	7-4
15.	Miami (Florida)	8-4
16.	Florida	9-4
17.	Tennessee	8-4
18.	Colorado State	6-6
19.	San Diego State	10-3
20.	Oklahoma State	9-3

Research Questions

- How do you adjust the modeling so that if a low tier team beats a high tier one, the final PageRanks are not skewed like in the 2016 College FBS?
- What other networks possess this *upset* phenomenon and how to deal with it there?