

# PageRank

Ben Burns, Dan Magazu, Lucas Chagas,  
Thomas Webster, Trung Do

Fall 2021

# Table of Contents

Motivation

Background

Formalizing PageRank

Solving PageRank as a Matrix

Applications

# Outline

Motivation

Background

Formalizing PageRank

Solving PageRank as a Matrix

Applications

# Motivation

- ▶ Problem: the internet has a *lot* of web pages
- ▶ A lot of the information out there either isn't relevant to us, or is inaccurate
- ▶ Motivation: we want a program that, when provided a phrase, returns webpages with information relevant to the input
- ▶ Intuitive solution: return back websites that either contain that phrase, or contain similar phrases
- ▶ This helps us find more *relevant* pages, but we can't know if we're getting the best information (much less *accurate* information) without manually going through each result
- ▶ We desire a stronger solution

# Outline

Motivation

Background

Formalizing PageRank

Solving PageRank as a Matrix

Applications

# PageRank

- ▶ Invented by Sergey Brin and Larry Page (1998)<sup>1</sup>
  - ▶ Publication marks them becoming co-founders of Google
- ▶ Idea: we want some way to numerically score each webpage based on how "important" it is
- ▶ Algorithm numerically scores each page  $p$  based on
  - ▶ How many other pages link to  $p$  (or "cite" it)
  - ▶ The "importance" of each of  $p$ 's citations
- ▶ We then numerically order pages to rank them
- ▶ PageRank: the procedure for scoring each website
- ▶ Google: the database that indexes the PageRank of each website for search

---

<sup>1</sup>Many use the year of the original manuscript, 1996  6/25

# Underlying Assumptions

- ▶ Running our basic search engine gives us a collection of pages with information relevant to our query
- ▶ Assumption: More "important" and useful websites will be the ones with proportionally more inbound links
- ▶ Pages with very reliable, primary information are likely to be cited by lots of website authors, and therefore will have lots of "flow" into them
- ▶ Assumption: websites with no outgoing links are treated as linking to every other website
- ▶ Else, these pages would just take in flow without ever redistributing, and eventually all users get stuck on these pages

# Finding Less Important Information

- ▶ A worry you may have is that we'll only find pages with lots of inbound links
- ▶ You can still find niche information by making your query more specific so that it won't match more general pages
- ▶ Searching "PageRank" will likely get you Wikipedia, but "Anatomy of PageRank architecture" gives you the original research literature.
- ▶ Your returned urls are still proportionally important results, your query just filtered out the numerically more "important", yet less relevant pages.



# Outline

Motivation

Background

Formalizing PageRank

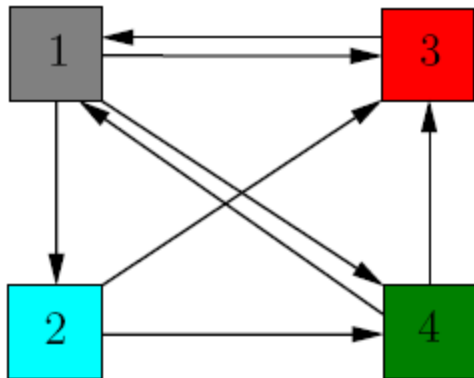
Solving PageRank as a Matrix

Applications

# Formalizing the PageRank problem

- ▶ We're going to construct a directed graph  $G = (V, E)$
- ▶ For each website we consider, we construct a node  $v_i \in V$
- ▶ For two distinct nodes  $v_i, v_j \in V$ , the *directed* edge  $v_i v_j \in E$  iff there is a link on website  $i$  that goes to website  $j$ .
- ▶ If  $v_i$  and  $v_j$  are not distinct (a website is linking to itself), we ignore the link and do not construct a loop edge.
  - ▶  $G$  is not a psuedograph
- ▶ Multiple hyperlinks on page  $i$  to page  $j$  are all represented by the single, directed edge
  - ▶  $G$  is not a multigraph.

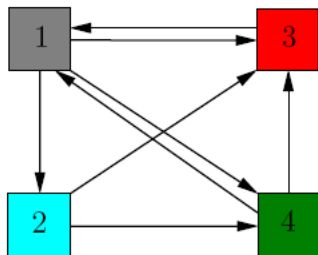
# Visual Representation



- ▶ We have a set of 4 websites
- ▶ Each edge represents a hyperlink from the origin node to the destination node

# Applying PageRank Values

- ▶ At first, we assign each vertex  $v \in V$  with a weight of  $\frac{1}{|V|}$ .
- ▶ All vertices have equal weight, and our weights sum up to 1. The weight of a particular vertex  $v_i$  is denoted  $PR(v_i)$ .
- ▶ Consider the set  $O_i$  of vertices that  $v_i$  has an edge to:
  - ▶  $O_i = \{v_j | v_i v_j \in E\}$  ( $O$  for outbound)
- ▶ A user on page 1 can choose to click a link to traverse to either 2, 3, or 4. In other words,  $O_1 = \{2, 3, 4\}$ .



# Traversing from Page to Page

- ▶ Assumption: A user on page  $i$  is equally likely to choose to visit each vertex in  $O_i$  (our set of vertices that  $v_i$  cites)
- ▶ In our example from before, the probability that our user on page 1 visits page 2 is  $P(v_2) = \frac{1}{|O_1|} = \frac{1}{3}$
- ▶ So a third of page 1's visitors will "flow" to page 2, a third to page 3, and a third to page 4
- ▶ From  $v_i$ ,  $P(v_j) = \frac{1}{|O_i|}$  if  $v_j \in O_i$ , and 0 else.

# Iteration Model

- ▶ All at once, all users will click one of the links on their current page
- ▶ At "click 0", the PageRank value of all vertices
$$PR(v, 0) = \frac{1}{|V|}$$
- ▶ On click 1, each page will equally split its PageRank value among the pages it cites
- ▶ Page 1 starts with  $PR(v_1, 0) = \frac{1}{4}$ , and contributes  $\frac{1}{12}$  to each of 2, 3, and 4 on click 1
- ▶ Conversely, page 1 will receive nothing from page 2, all of  $PR(v_3, 0)$ , and half of  $PR(v_4, 0)$ . So after click 1,

$$\begin{aligned} PR(v_1, 1) &= 0 \cdot PR(v_2, 0) + PR(v_3, 0) + \frac{1}{2}PR(v_4, 0) \\ &= \frac{1}{4} + \frac{1}{8} = \frac{3}{8} \end{aligned}$$

# What else do we need?

- ▶ We want a function  $f(v_i, v_j)$  defined as:

$$f(v_i, v_j) := \frac{\text{\# of links from page } i \text{ to page } j}{\text{total \# of links out of page } j}$$

- ▶ In our simplified case, this will be  $\frac{1}{|O_j|}$  if  $j$  links to  $i$ , and 0 else
- ▶ Additionally, at some point, our user who is randomly clicking links will eventually stop clicking links
- ▶ At each iteration we consider a constant probability  $d :=$  the probability that our user stops traversing the graph
  - ▶ Various studies have settled on  $d \approx 0.85$

# General PageRank equation

- For an arbitrary graph, we obtain the PageRank value of vertex  $v_i$  at iteration  $t + 1$  from the following:

$$PR(v_i, t + 1) = \frac{1 - d}{|V|} + d \sum_{v_j \in I_i} f(v_i, v_j) PR(v_j, t)$$

$O_i$  is set of vertices with edges outbound from  $v_i$        $I_i$  is the set of vertices with edges inbound to  $v_i$

$$f(v_i, v_j) = \begin{cases} \frac{1}{|O_j|} & i \in O_j \\ 0 & \text{else} \end{cases}$$



# Outline

Motivation

Background

Formalizing PageRank

Solving PageRank as a Matrix

Applications

# Matrix Construction

- ▶ The idea behind PageRank is construct and solve a matrix
- ▶ Each row  $i$  represents the PageRank equation for  $PR(v_i, t)$
- ▶ Each column  $j$  represents the PageRank value contributed by  $j$  to each vertex  $v_i$
- ▶ The solved for  $t + 1$  iteration PageRank values will therefore be a vector:

$$\begin{bmatrix} PR(v_1, t + 1) \\ PR(v_2, t + 1) \\ \vdots \\ PR(v_n, t + 1) \end{bmatrix}$$

## Boiling it Down to an Eigenvalue

- ▶ For our iteration process, we want to be able to input a vector of PageRank values at iteration  $t$  vector  $\mathbf{R}(t)$ , and get back a vector of PageRank values at iteration  $t + 1$  ( $\mathbf{R}(t + 1)$ )
- ▶ In other words, we can determine PageRank values as

$$\begin{bmatrix} PR(v_1, t + 1) \\ PR(v_2, t + 1) \\ \vdots \\ PR(v_n, t + 1) \end{bmatrix} = A \cdot \begin{bmatrix} PR(v_1, t) \\ PR(v_2, t) \\ \vdots \\ PR(v_n, t) \end{bmatrix}$$

for an  $n \times n$  matrix  $A$

- ▶ For our values to converge, our input vector will approximately equal our output vector
- ▶ In other words, we're searching for an eigenvector

# Defining Our Matrix $A$

- Observe our general PageRank equation

$$PR(v_i, t+1) = \frac{1-d}{|V|} + d \sum_{v_j \in I_i} f(v_i, v_j) PR(v_j, t)$$

- We define the matrix  $A$  to the  $n \times n$  matrix where  $A_{ij} := f(v_i, v_j)$
- We then multiply this matrix by our  $t$  iteration PageRank vector
- Finally, we add the result by a  $n$  length vector of  $\frac{1-d}{|V|}$  terms

$$\mathbf{R}(t+1) = \begin{bmatrix} (1-d)/|V| \\ (1-d)/|V| \\ \vdots \\ (1-d)/|V| \end{bmatrix} + \begin{bmatrix} f(v_1, v_1) & \dots & f(v_1, v_n) \\ f(v_2, v_1) & \dots & f(v_2, v_n) \\ \vdots & \ddots & \vdots \\ f(v_n, v_1) & \dots & f(v_n, v_n) \end{bmatrix} \cdot \mathbf{R}(t)$$

## Determining Existence of $PR$ Eigenvector

- ▶ Each column  $i$  of our matrix is the series of terms  $f(v_1, v_i)$  through  $f(v_n, v_i)$

# Outline

Motivation

Background

Formalizing PageRank

Solving PageRank as a Matrix

Applications

# Applications

- ▶ PageRank is perhaps the most famous search ranking algorithm. Googles high-quality search engine results are directly correlated to PageRank
- ▶ There are a remarkable wide variety of applications of the PageRank algorithm that apply to non-search engine contexts.
- ▶ Its simplicity and elegance allow PageRank to be a more general and powerful tool.
- ▶ Let's look at the applications of PageRank and its connection to Twitter.

# Twitter

- ▶ In 2010, Twitter was lagging behind and was lacking a user recommendation service.
- ▶ This was perceived both externally and internally as a critical gap in Twitter's product offerings, so quickly launching a high-quality product was a top priority.
- ▶ Twitter is unique because of the asymmetric nature of the following relationship—a user can receive messages from another without reciprocation.
- ▶ This differs substantially from other social networks such as Facebook or LinkedIn, where social ties can only be established with the consent of both participating members.



# Introducing PageRank

- ▶ This works well with PageRank because we can determine outbound and inbound links.
- ▶ A user  $u$  is likely to follow those who are followed by users that are similar to  $u$ .
- ▶ These users are in turn similar to  $u$  if they follow the same (or similar) users.
- ▶ Therefore using Page Rank, Twitter is able to offer unique recommendations for users.
- ▶ By analyzing who the user follows and who those users follow, the PageRank algorithm will allow Twitter to make specific recommendations for each user.
- ▶ Essentially, the more outbound links that an account receives correlate to a higher PageRank score.