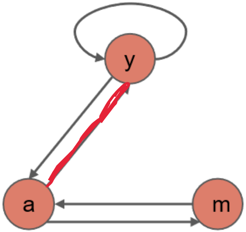
CSCI 4022 Fall 2021
PageRank Computation

Example: Perform 2 steps of power iteration on our YAM graph.

Intuition: Which node or nodes should hold the most importance?

The **column-stochastic** matrix for the graph was:

$$M = \begin{array}{c|ccc} & Y & A & M \\ \hline Y & 1/2 & \boxed{1/2} & 0 \\ A & 1/2 & 0 & 1 \\ M & 0 & 1/2 & 0 \\ & 1 & 1 & 1 \end{array}$$



Announcement and Reminders

You may use a non-graphing Calculator

To-dos:

1. Study for the exam! Content through A-priori
2. The exam is closed book, closed note, but you may construct your own full page (front and back) review and formula sheet for use on the exam. I'll leave a spot on the exam for you to submit the sheet you used for our records.
3. Exam 1 (Gradescope) opens on Wednesday October 6 at 6pm. You get 2 hours once you start, and must be completely submitted by Thursday October 6 at 6pm.
4. I'll be active on Piazza from 6p-8p tonight, so if you do the exam in that interval and have questions, I should usually respond within 10-15m.

Upcoming:

1. HW5 posted, due Oct 18 (item-basket stuff). In the meantime...

PageRank

The idea:

- Each link's vote is proportional to the importance of its source page
- If page j with importance r_j has d_j out-links (out-degree), then each link get r_j/d_j votes
- Page j's own importance is the sum of the votes on its in-links (in-degree). This leads to the **Definition:** The **rank** (in *Pagerank* for **page** j is given by

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

where d_i is the *out-degree* of node i and the sum is over all the nodes i that link to j.

Typically we *could* solve this by taking the $n \times n$ matrix representation of the $r = Mr$ system:

$$Mr - r = 0 \implies (M - I)r = 0$$

which represents a solvable linear system... but it **also** represents an equation of the form $Ax = \lambda x$ with $\lambda = 1$.

PageRank: Power Iteration

The idea: We want the eigenvector with eigenvalue of 1 of the matrix M. Then $Mr = r$ is solved by eigenvalue-eigenvector pair $(1, r)$.

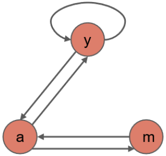
The bonus: column-stochastic matrices always have *largest* eigenvalue of 1.

The resulting algorithm is known as **power iteration**.

1. Suppose that there are N total web pages to rank.
2. Initialize: $r^{(0)} = [1/N, 1/N, \ldots 1/N]$
3. Iterate: $\underline{r^{(t+1)}} = \underline{Mr}^{(t)}$
4. Stop if: $d(r^{(t+1)}, r^{(t)})$ is small. A typical stop would be $d(r^{(t+1)}, r^{(t)}) < \varepsilon$ under appropriate norm (like L_1, so $d(r^{(t+1)}, r^{(t)}) = \sum_{i=1}^{N} |r_i^{(t+1)} - r_i^{(t)}|$ for some small ε

PageRank: Power Iteration Example

$$M = \begin{array}{c|ccc} & Y & A & M \\ \hline Y & 1/2 & 1/2 & 0 \\ A & 1/2 & 0 & 1 \\ M & 0 & 1/2 & 0 \end{array} \quad \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = M r^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

$$\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix}$$

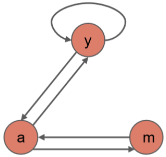
$t = \dots$	0	1	2	$\dots$	15
$r_y^{(t)}$	1/3	5/12		$\dots$	
$r_a^{(t)}$	1/2	1/3		$\dots$	
$r_m^{(t)}$	1/6	1/4		$\dots$	

1

PageRank: Power Iteration Example

$$Mr = \begin{array}{c} \\ \\ \end{array}\begin{array}{ccc} Y & A & M \end{array}$$

$$Mr = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

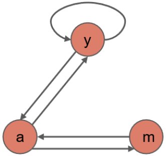


1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

$t = \dots$	0	1	2	$\dots$	15
$r_y^{(t)}$	1/3			$\dots$	
$r_a^{(t)}$	1/3			$\dots$	
$r_m^{(t)}$	1/3			$\dots$	

PageRank: Power Iteration Example

$$Mr = \begin{array}{c} \begin{array}{ccc} Y & A & M \end{array} \\ \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \end{array} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

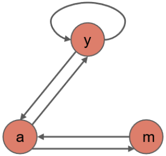


1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

$t = \dots$	0	1	2	$\dots$	15
$r_y^{(t)}$		$1/2 \cdot 1/3 + 1/2 \cdot 1/3 = 1/3$		$\dots$	
$r_a^{(t)}$		$1/2 \cdot 1/3 + 1 \cdot 1/3 = 1/2$		$\dots$	
$r_m^{(t)}$		$1/2 \cdot 1/3 = 1/6$		$\dots$	

PageRank: Power Iteration Example

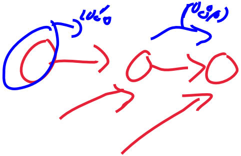
$$Mr = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{c}\begin{array}{ccc} Y & A & M \end{array} \\ \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}\end{array} \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix}$$



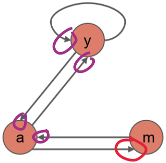
1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

$t = \dots$	0	1	2	$\dots$	15
$r_y^{(t)}$	$1/3$	$1/3$		$\dots$	
$r_a^{(t)}$	$1/3$	$1/2$		$\dots$	
$r_m^{(t)}$	$1/3$	$1/6$		$\dots$	

PageRank: Power Iteration Example



$$Mr = \begin{array}{c} \begin{array}{ccc} Y & A & M \end{array} \\ \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \end{array} \begin{bmatrix} r_y^{(14)} \\ r_a^{(14)} \\ r_m^{(14)} \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

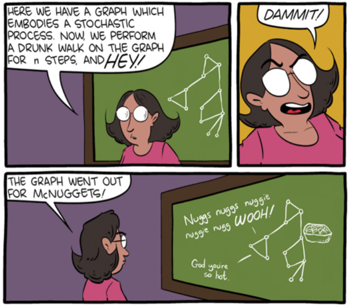
$t = \dots$	0	1	2	$\dots$	15
$r_y^{(t)}$	1/3	1/3		$\dots$	.398
$r_a^{(t)}$	1/3	1/2		$\dots$	.404
$r_m^{(t)}$	1/3	1/6		$\dots$	.197

Idea: We are handing out/initializing equal amounts of importance to all web pages, then letting the importance **diffuse** along the links between pages. Eventually, importance gets redistributed appropriately

As a Random Walk

Imagine a tiny person walking on the web…

1. At any time t, walker is on some page i
2. At time $t+1$, the walker follows a link from i, chosen uniformly random, and ends up on some page j, linked from i
3. … then the process repeats indefinitely
4. If we define $p(t) :=$ a probability vector whose ith coordinate is the probability that the walker is at page i at time t
5. **Then:** $p(t)$ is a probability distribution over the set of pages
6. **And:** That final probability distribution is the *PageRank* for each page!



On Random Walks

Let $p(t) :=$ a probability vector whose ith coordinate is the probability that the walker is at page i at time t

So $p(t)$ is a probability distribution over the set of pages...

So at time $t+1$, we can use M to update the probability vector:

$$p(t+1) = Mp(t)$$

Suppose the random walker reaches a state *eventually* where:

$$p(t+1) = Mp(t) = p(t)$$

Definition: Then $p(t)$ is a *stationary distribution* of the random walk. Recall that our original rank vector r satisfies $r = Mr$... so

Result: r can be thought of as a stationary distribution for this random walk!

Markov Theory

Suppose the random walker reaches a state eventually where:

$$p(t+1) = M p(t) = p(t)$$

↳ σ change over time.

Then $r = p(t)$ can be thought is a stationary distribution for this random walk!

This type of model where the state at time $t+1$ the probability vector $(p(t+1))$ depends on the transition matrix M and the state at time t $(p(t))$ is a first-order Markov model.

Definition: A *First-order* Markov process includes the following two assumptions:

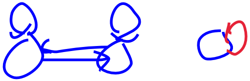
1. Past and future are independent of one another, given the present.
2. The state at time $t+1$ depends only on the state at time t (first order). This is also called the *Markov Property* and such a system is referred to as "memoryless."

Math tells us that under certain conditions the stationary distribution is unique and will be reached eventually, *no matter* the initial rank vector guess.

Stationarity and Existence

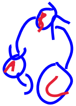
On very small problems, we might find the stationary distribution via setting up **flow** equations and looking for a solution. On larger problems, we iterate with **power iteration**. These don't always work!

- One issue is *disconnected* states.



$$\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Another is *periodic* systems.



$$\begin{array}{ccc} A & B & C \\ \end{array}$$
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

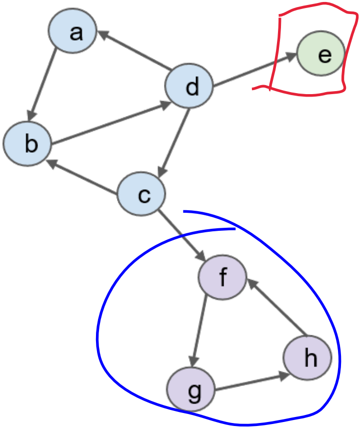
Definition: The transition probability distribution given by M is called *ergodic* if every node is *reachable* from every other state, and there are no strictly periodic cycles.

Proposition: If a Markov chain is *ergodic*, then there exists a **unique** stationary distribution for any given set of transition probabilities.

Proper Convergence

Definition: The transition probability distribution given by M is called *ergodic* if every node is *reachable* from every other state, and there are no strictly periodic cycles.

There are two main issues we must be concerned with for PageRank.



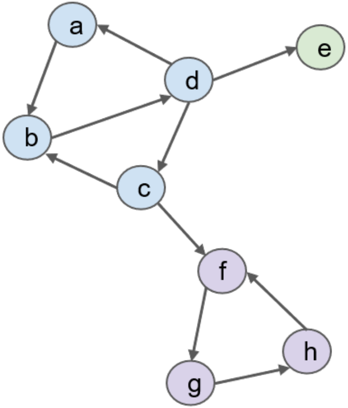
1. **Definition:** A *dead end* is a page without any outlinks. It is a subcase of a... page e

2. **Definition:** A *spider trap* is a region of the graph where all out-links are contained within that subgraph.

Proper Convergence

Definition: The transition probability distribution given by M is called *ergodic* if every node is *reachable* from every other state, and there are no strictly periodic cycles.

There are two main issues we must be concerned with for PageRank.

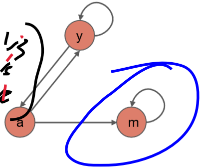


1. **Definition:** A *dead end* is a page without any outlinks. It is a subcase of a...
   Result: Random walker has nowhere to go! Importance that enters the node is *lost* and leaks out of the system.
2. **Definition:** A *spider trap* is a region of the graph where all out-links are contained within that subgraph.
   Result: Random walker stuck in the trap! Importance that enters the trap just accumulates there, absorbs all importance from the in-links.

It's a Trap!

Example: Take a few steps of power iteration on the modified spider-trap YAM graph below.

$$M = \begin{array}{c|ccc} & Y & A & M \\ \hline Y & 1/2 & 1/2 & 0 \\ A & 1/2 & 0 & 0 \\ M & 0 & 1/2 & 1 \end{array} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/6 \\ 1/2 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

$t = \dots$	0	1	2
$r_y^{(t)}$	1/3 →	1/4	
$r_a^{(t)}$	1/6 →	1/6	
$r_m^{(t)}$	1/2 →	7/12	

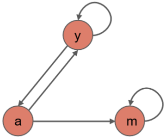
… eventually this will approach [0,0,1]!

It's a Trap!

Example: Take a few steps of power iteration on the modified spider-trap YAM graph below.

$$Mr = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{ccc} Y & A & M \\ \end{array}$$

$$Mr = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \ldots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

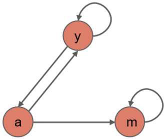
$t = \ldots$	0	1	2
$r_y^{(t)}$	1/3		
$r_a^{(t)}$	1/3		
$r_m^{(t)}$	1/3		

... eventually this will approach [0,0,1]!

It's a Trap!

Example: Take a few steps of power iteration on the modified spider-trap YAM graph below.

$$Mr = \begin{array}{c} \begin{array}{ccc} Y & A & M \end{array} \\ \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \end{array} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

$t = \dots$	0	1	2
$r_y^{(t)}$		$1/2 \cdot 1/3 + 1/2 \cdot 1/3 = 1/3$	
$r_a^{(t)}$		$1/2 \cdot 1/3 = 1/6$	
$r_m^{(t)}$		$1/2 \cdot 1/3 + 1 \cdot 1/3 = 1/2$	

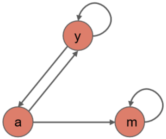
... eventually this will approach [0,0,1]!

It's a Trap!

Example: Take a few steps of power iteration on the modified spider-trap YAM graph below.

$$Mr = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{ccc} Y & A & M \\ \end{array}$$

$$Mr = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/6 \\ 1/2 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \ldots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

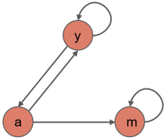
$t = \ldots$	0	1	2
$r_y^{(t)}$	$1/3$	$1/3$	
$r_a^{(t)}$	$1/3$	$1/6$	
$r_m^{(t)}$	$1/3$	$1/2$	

... eventually this will approach [0,0,1]!

It's a Trap!

Example: Take a few steps of power iteration on the modified spider-trap YAM graph below.

$$M = \begin{array}{c} \\ Y \\ A \\ M \end{array} \begin{array}{c} \begin{array}{ccc} Y & A & M \end{array} \\ \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \end{array}$$



1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = M r^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

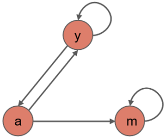
$t = \dots$	0	1	2
$r_y^{(t)}$	$1/3$	$1/3$	$1/4$
$r_a^{(t)}$	$1/3$	$1/6$	$1/6$
$r_m^{(t)}$	$1/3$	$1/2$	$7/12$

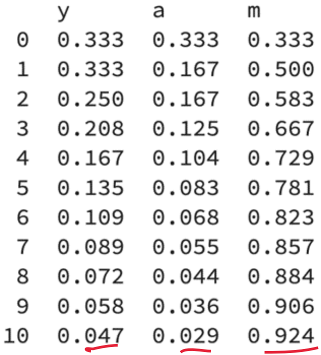
... eventually this will approach [0,0,1]!

It's a Trap!

Example: Take a few steps of power iteration on the modified spider-trap YAM graph below.

$$M = \begin{array}{c|ccc} & Y & A & M \\ \hline Y & 1/2 & 1/2 & 0 \\ A & 1/2 & 0 & 0 \\ M & 0 & 1/2 & 1 \end{array}$$





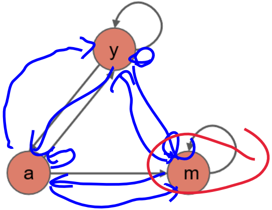
```
      y      a      m
0   0.333  0.333  0.333
1   0.333  0.167  0.500
2   0.250  0.167  0.583
3   0.208  0.125  0.667
4   0.167  0.104  0.729
5   0.135  0.083  0.781
6   0.109  0.068  0.823
7   0.089  0.055  0.857
8   0.072  0.044  0.884
9   0.058  0.036  0.906
10  0.047  0.029  0.924
```

... eventually this will approach [0,0,1]!

Activating your Trap Card:

The Google solution: at each time step, the random walker has two options:

1. With probability β, follow a real link at random.
2. With probability $1 - \beta$, *teleport* or jump to any random page, equally likely
3. Common values for β: 0.8-0.9



Counting Result: Suppose we fall into a spider trap. If we choose $\beta = 0.85$, after how many steps are we more likely than not to have escaped?

3/20 chance ⇒ average of 20/3 steps to escape.

P(escape first try) = .15

P(" second try) = (.85)(.15)

P(" 3rd try) = (.85)(.85)(.15)

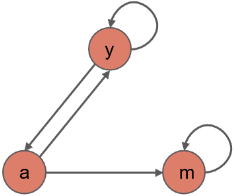
↑ Geometric RV.

↑ when do they drop < .5.

Activating your Trap Card:

The Google solution: at each time step, the random walker has two options:

1. With probability β, follow a real link at random.
2. With probability $1-\beta$, *teleport* or jump to any random page, equally likely
3. Common values for β: 0.8-0.9



Counting Result: Suppose we fall into a spider trap. If we choose $\beta = 0.85$, after how many steps are we more likely than not to have escaped?

Solution: P(still stuck after k steps) $\approx P$(no teleports in k steps) $= \beta^k$.
Goal: find the k so that $\beta^k < 0.5$.

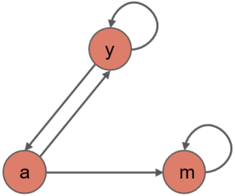
$$\log \beta^k < \log .5$$

$$k \log \beta < \log .5$$

$$k < \frac{\log .5}{\log \beta}$$

Activating your Trap Card:

The Google solution: at each time step, the random walker has two options:



1. With probability β, follow a real link at random.
2. With probability $1-\beta$, *teleport* or jump to any random page, equally likely
3. Common values for β: 0.8-0.9

Counting Result: Suppose we fall into a spider trap. If we choose $\beta = 0.85$, after how many steps are we more likely than not to have escaped?

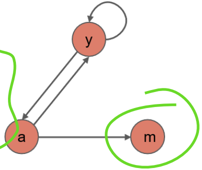
Solution: $P(\text{still stuck after k steps}) \approx P(\text{no teleports in k steps}) = \beta^k$.
Goal: find the k so that $\beta^k < 0.5$.

$B^k < .5 \implies \log \beta^k < \log 0.5 \implies k \log \beta < -\log 2$. Then: $k > \frac{-\log 2}{\log \beta} = \frac{-\log 2}{\log 0.85} = 4.26$. So with a little under a 1/6 chance (16.6%) to teleport, we're typically out by 5 iterations.

What about Dead Ends?

Example: Take a step of power iteration on the modified dead end YAM graph below.

$$M = \begin{array}{c c} & \begin{array}{ccc} Y & A & M \end{array} \\ \begin{array}{c} Y \\ A \\ M \end{array} & \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix} \end{array} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \ldots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

$t = \ldots$	0	1	2
$r_y^{(t)}$	1/3	1/3	
$r_a^{(t)}$	1/3	1/6	
$r_m^{(t)}$	1/3	1/6	

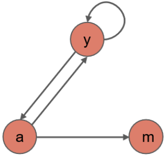
sums to... 2/3

What about Dead Ends?

Example: Take a step of power iteration on the modified dead end YAM graph below.

$$Mr = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{ccc} Y & A & M \\ \end{array}$$

$$Mr = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

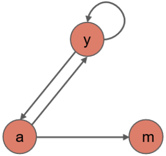
$t = \dots$	0	1	2
$r_y^{(t)}$	$1/3$	$1/2 \cdot 1/3 + 1/2 \cdot 1/3 = 1/3$	
$r_a^{(t)}$	$1/3$	$1/2 \cdot 1/3 = 1/6$	
$r_m^{(t)}$	$1/3$	$1/2 \cdot 1/3 = 1/6$	

What about Dead Ends?

Example: Take a step of power iteration on the modified dead end YAM graph below.

$$Mr = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{ccc} Y & A & M \\ \end{array}$$

$$Mr = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$



1. Init: $r^{(0)} = [1/N, \dots 1/N]$
2. Iterate: $r^{(t+1)} = Mr^{(t)}$
3. Stop if: $d(r^{(t+1)}, r^{(t)}) < tol$.

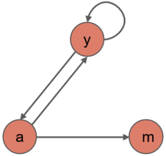
$t = \dots$	0	1	2
$r_y^{(t)}$		$1/3$	
$r_a^{(t)}$		$1/6$	
$r_m^{(t)}$		$1/6$	

We're in trouble! This vector doesn't sum to 1!

The Dead End

Example: Take a few steps of power iteration on the modified Dead End YAM graph below.

$$M = \begin{array}{c|ccc} & Y & A & M \\ \hline Y & 1/2 & 1/2 & 0 \\ A & 1/2 & 0 & 0 \\ M & 0 & 1/2 & 0 \end{array}$$



```
        y       a       m
 0  0.333   0.333   0.333
 1  0.333   0.167   0.167
 2  0.250   0.167   0.083
 3  0.208   0.125   0.083
 4  0.167   0.104   0.062
 5  0.135   0.083   0.052
 6  0.109   0.068   0.042
 7  0.089   0.055   0.034
 8  0.072   0.044   0.027
 9  0.058   0.036   0.022
10  0.047   0.029   0.018
```

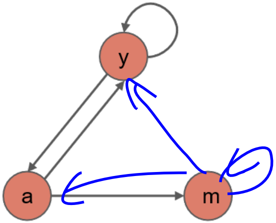
... eventually this will approach [0,0,0]!

Dead End Teleports:

A solution: when at a dead end, follow a random teleport link with probability 1.

1. This represents adjusting the adjacency matrix to require every column sum to 1!
2.

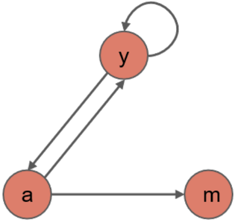
$$M = \begin{array}{c c} & \begin{array}{ccc} Y & A & M \end{array} \\ \begin{array}{c} Y \\ A \\ M \end{array} & \begin{bmatrix} 1/2 & 1/2 & 1/3 \\ 1/2 & 0 & 1/3 \\ 0 & 1/2 & 1/3 \end{bmatrix} \end{array}$$



Full PageRank:

The Google solution: a blend of the the solution to spider traps and dead ends.

1. With probability β, follow a real link at random.
2. With probability $1 - \beta$, *teleport* or jump to any random page, equally likely
3. Common values for β: 0.8-0.9
4. Brin-Page, 1998 (with tens of thousands of citations!) gives the updates PageRank formula:



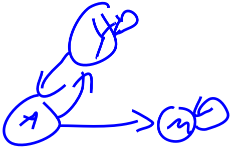
$$r_j = \overset{M}{\boxed{\sum_{i\to j}}\,\beta\boxed{\frac{r_i}{d_i}}} + \overset{\text{rest of time}}{\boxed{(1-\beta)}}\underset{\uparrow\ \text{teleport}}{\frac{1}{N}}$$

The resulting transition matrix is given by

$$A = \beta M + (1-\beta)\overbrace{\left[\frac{1}{N}\right]_{N\times N}}^{every\,entry\,is 1/N}$$

and then normalize

Brin-Page Rank in action



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

For this spider-trap example, we would have

$$A = 0.8\begin{bmatrix}1/2 & 1/2 & 0\\ 1/2 & 0 & 0\\ 0 & 1/2 & \boxed{1}\end{bmatrix} + 0.2\begin{bmatrix}1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\end{bmatrix}$$

$$= \begin{bmatrix}7/15 & 7/15 & 1/15\\ 7/15 & 1/15 & 1/15\\ 1/15 & 7/15 & 13/15\end{bmatrix} \begin{bmatrix}1/3\\ 1/3\\ 1/3\end{bmatrix}$$

Looks all good, right?

sum to 1

```
      y      a      m
0   0.333  0.333  0.333
1   0.333  0.200  0.467
2   0.280  0.200  0.520
3   0.259  0.179  0.563
4   0.242  0.170  0.588
5   0.231  0.163  0.605
6   0.225  0.159  0.616
7   0.220  0.156  0.623
8   0.217  0.155  0.628
9   0.215  0.154  0.631
10  0.214  0.153  0.633
```

must sums to 1

Brin-Page Rank in action

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

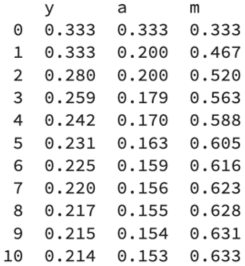
For this spider-trap example, we would have

$$A = 0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

sums to .8

sums to .2

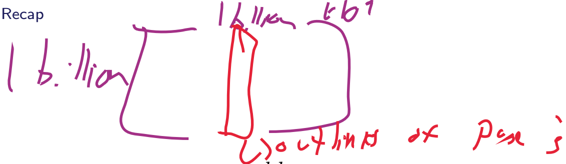
$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$



```
        y      a      m
0   0.333  0.333  0.333
1   0.333  0.200  0.467
2   0.280  0.200  0.520
3   0.259  0.179  0.563
4   0.242  0.170  0.588
5   0.231  0.163  0.605
6   0.225  0.159  0.616
7   0.220  0.156  0.623
8   0.217  0.155  0.628
9   0.215  0.154  0.631
10  0.214  0.153  0.633
```

Looks all good, right? ... but this **is not column-stochastic in the case of a dead-end**. The columns in this case each sum to "0.8 times the M matrix's columns plus 0.2." So we still needed column-stochastic M!

Brin-Page Rank



The final step: In our matrix-vector multiplication $\boldsymbol{r}^{new} = A\boldsymbol{r}^{old}$, we can also *normalize* r to ensure it sums to 1.
So let's talk about *implementation*. How big are these things?

may be...
8 15?

Say there are about $N = 10^9$ or 1 billion web pages/nodes in the graph. 999999952 0's

- Then if we need 4 bytes for each entry in A and $\boldsymbol{r}$...
- A has $N^2 = 10^{18}$ entries, so 4 times that in bytes...
- we need a billion GB. No thanks!
- So lets save some memory!

Computing Page Rank

if A is sparse:
mostly 0's

Goal: For $A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$, compute $\boxed{\boldsymbol{r}^{new} = A\boldsymbol{r}^{old}}$ **quickly**.

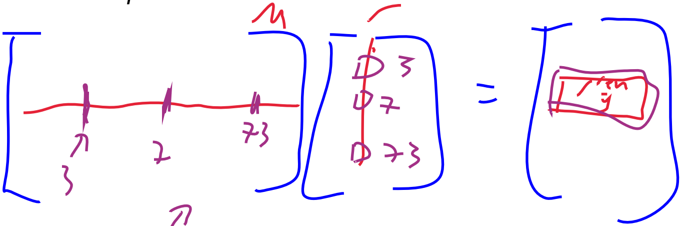
Point 1: Denote the elements $A_{ji} = \beta M_{ji} + \frac{1-\beta}{N}$

Computing Page Rank

Goal: For $A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$, compute $\boldsymbol{r}^{new} = A\boldsymbol{r}^{old}$ **quickly**.

Point 1: Denote the elements $A_{ji} = \boxed{\beta M_{ji}} + \frac{1-\beta}{N}$

Point 2: To get $\boldsymbol{r}_j^{new}$, we take the jth *row* of A and multiply it by $\boldsymbol{r}^{old}$. That's a *dot product* or a sum.



where there is a non-zero

$\sum_{\text{row } j} M_{ij}$

Computing Page Rank

Goal: For $A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$, compute $\boldsymbol{r}^{new} = A\boldsymbol{r}^{old}$ **quickly**.

Point 1: Denote the elements $A_{ji} = \beta M_{ji} + \frac{1-\beta}{N}$

Point 2: To get $\boldsymbol{r}_j^{new}$, we take the jth *row* of A and multiply it by $\boldsymbol{r}^{old}$.

That's a *dot product* or a sum.

In other words,

$$\boldsymbol{r}_j^{new} = \sum_{i=1}^{N} A_{ji}\boldsymbol{r}_i^{old}$$

Computing Page Rank

Goal: For $A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$, compute $\boldsymbol{r}^{new} = A\boldsymbol{r}^{old}$ **quickly**.

Point 1: Denote the elements $\boxed{A_{ji}} = \underbrace{\beta M_{ji}} + \underbrace{\frac{1-\beta}{N}}$

In other words,

$$\boldsymbol{r}_j^{new} = \sum_{i=1}^{N} A_{ji}\boldsymbol{r}_i^{old}$$

We can rewrite and simplify: $\boldsymbol{r}_j^{new} = \sum_{i=1}^{N}\left(\beta M_{ji} + \frac{1-\beta}{N}\right)\boldsymbol{r}_i^{old}$, or

$$\boldsymbol{r}_j^{new} = \underbrace{\sum_{i=1}^{N}\beta M_{ji}\boldsymbol{r}_i^{old}}_{\text{lots of 0's}} + \underbrace{\boxed{\sum_{i=1}^{N}\boxed{\frac{1-\beta}{N}}\boldsymbol{r}_i^{old}}}_{\text{Same \#}}$$

Computing Page Rank

Goal: For $A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$, compute $\boldsymbol{r}^{new} = A\boldsymbol{r}^{old}$ **quickly**.

Point 1: Denote the elements $A_{ji} = \beta M_{ji} + \frac{1-\beta}{N}$

In other words,

$$\boldsymbol{r}_j^{new} = \sum_{i=1}^{N} A_{ji}\boldsymbol{r}_i^{old}$$

We can rewrite and simplify: $\boldsymbol{r}_j^{new} = \sum_{i=1}^{N}\left(\beta M_{ji} + \frac{1-\beta}{N}\right)\boldsymbol{r}_i^{old}$, or

$\boldsymbol{r}_j^{new} = \sum_{i=1}^{N}\beta M_{ji}\boldsymbol{r}_i^{old} + \sum_{i=1}^{N}\frac{1-\beta}{N}\boldsymbol{r}_i^{old}$

Point 3: If we've managed to make $\sum_{i=1}^{N}\boldsymbol{r}_i^{old} = 1$, this simplifies more! The second sum is just $\frac{1-\beta}{N}$. Result:

$$\boldsymbol{r} = \boxed{\beta M\boldsymbol{r}} + \boxed{\left[\frac{1-\beta}{N}\right]_{N\times 1}}$$

Sparse sum

non-random

Computing Page Rank

Goal: For $A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$, compute $\boldsymbol{r}^{new} = A\boldsymbol{r}^{old}$.

The effect of teleports simplifed to adding a constant vector: $\boldsymbol{r} = \beta M\boldsymbol{r} + \left[\frac{1-\beta}{N}\right]_{N\times 1}$... what's the payoff?

1. M is a *sparse* matrix. Assume each node has 10 links, we only have 10N non-zero entries instead of N^2!
2. So only 40 GB to hold if $N = 10^9$! This is likely not doable in main memory, but is at least able to be stored.

Iteration:

1. Compute $\boldsymbol{r}^{new} = M\boldsymbol{r}^{old}$ ✓
2. Add a constant $(1-\beta)/N$ to every entry in $\boldsymbol{r}^{new}$ ✓
3. Renormalize $\boldsymbol{r}^{new}$ so that is sums to 1. **Must** be done if M has dead ends, but can also help with floating point precision in general.

Computing Page Rank

Algorithm: with a couple of short-cuts...

- **Input:** a convergence tolerance ε, graph G, and parameter β.
- **Output:** PageRank vector $\boldsymbol{r}$

Initialize: $\boldsymbol{r}_{old} = 1/N$ for # of nodes N. **Iterate:**

1. Let the link economy spread out importance. **For all** j: $\boldsymbol{r}_j^{new,*} = \sum_{i \to j} \beta \boldsymbol{r}_i^{old}/d_i$ (for all in-links, sum importance/degree of in-link).
2. Then Re-insert the "leaked" or "taxed" PageRank from teleports. **For all** j: $\boldsymbol{r}_j^{new,*} = \boldsymbol{r}_j^{new,*} + \frac{1-S}{N}$. Where we keep a running tally of $S = \sum_j \boldsymbol{r}_j^{new,*}$ in the prior step. This handles **both** normalization and including teleports: take whatever $\boldsymbol{r}_j^{new,*}$ currently sums to, then add what's needed to make it sum to 1... divided equally over all N entries.
3. Check convergence with some norm, e.g. **break** if $d(\boldsymbol{r}_{old}, \boldsymbol{r}_{new}) < \varepsilon$.
4. Update $\boldsymbol{r}_{old} = \boldsymbol{r}_{new}$

Sparse Mat-Vec

Assuming $\boldsymbol{r^{new}}$ fits into main memory, we can run this by accessing $\boldsymbol{r^{old}}$ and M from disk.

One step of power iteration becomes:

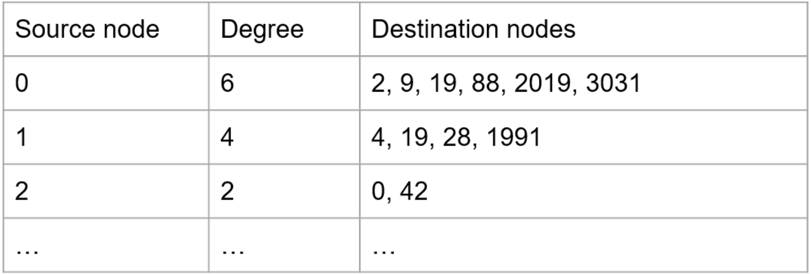
1. Initialize all entries of $\boldsymbol{r^{new}} = \frac{1-\beta}{N}$
2. For each page i with out-degree d_i...
   - 2a. Read into memory row i from M, so we get $(i, d_i, \{j_1, j_2, \dots j_{d_i}\})$
   - 2b. For j in $\{j_1, j_2, \dots j_{d_i}\}$, the destinations of i...:

$$\boldsymbol{r}_j^{new} += \beta \boldsymbol{r}_i^{old} / d_i$$

Sparse Mat-Vec

You may be asking... where did M go? We're trying to not save it into memory by reverting back to the original thought process: importance only flows out along links $i \to j$ **and** flows out relative to the number outlinks of a node d_i.

So those things are all that we want to save and use. We encode only the non-zero entries of M by keeping our links as a list of lists or a dictionary:

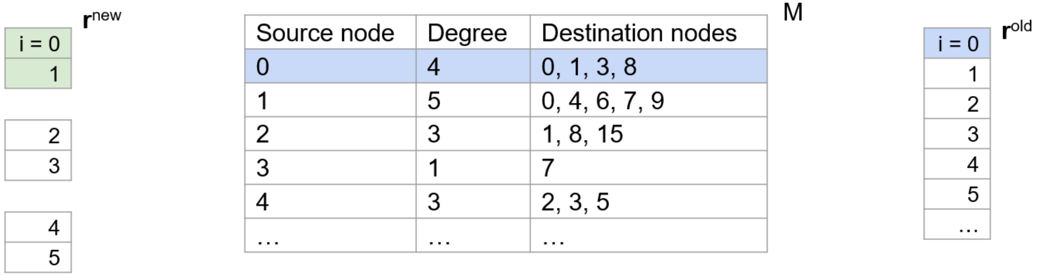


Source node	Degree	Destination nodes
0	6	2, 9, 19, 88, 2019, 3031
1	4	4, 19, 28, 1991
2	2	0, 42
...	...	...

Sparse Mat-Vec

We can even tweak the algorithm if we don't want to have to load all of M or r into memory at once. This is known as **block** updating. To not load r into memory:

1. Break $\boldsymbol{r}^{new}$ into k **blocks** or chunks, and then
2. update one chunk at a time by *scanning* through M and $\boldsymbol{r}_{old}$ once *per block*



$\mathbf{r}^{\text{new}}$

i = 0
1

2
3

4
5

M

Source node	Degree	Destination nodes
0	4	0, 1, 3, 8
1	5	0, 4, 6, 7, 9
2	3	1, 8, 15
3	1	7
4	3	2, 3, 5
…	…	…

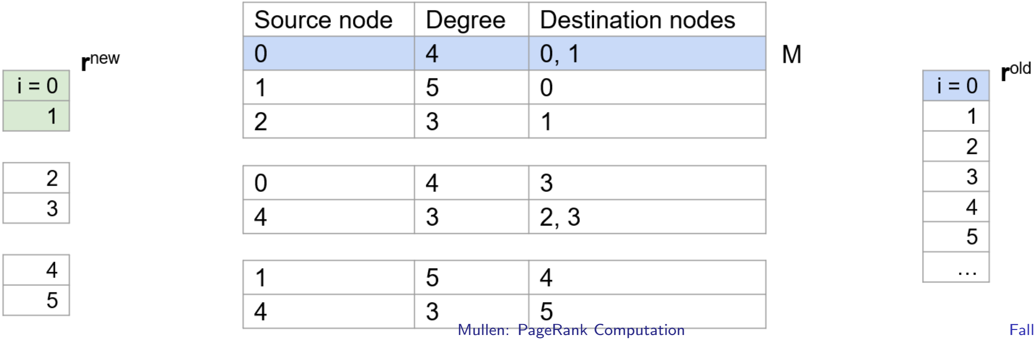
$\mathbf{r}^{\text{old}}$

i = 0
1
2
3
4
5
…

Block-Stripe

The absolute best-case for a huge M is that certain regions of M only link to themselves or one another. *For those regions*, we could save additional time by breaking M into k *stripes*, that correspond to the blocks of r where destination nodes are needed.

This is called *block-stripe* updating, and allows us to not read M repeatedly. Instead it ends up somewhat divided into topic areas!



$\mathbf{r}^{\text{new}}$

i = 0
1

2
3

4
5

M

Source node	Degree	Destination nodes
0	4	0, 1
1	5	0
2	3	1

0	4	3
4	3	2, 3

1	5	4
4	3	5

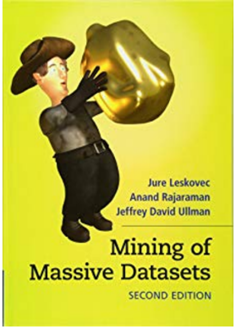
$\mathbf{r}^{\text{old}}$

i = 0
1
2
3
4
5
…

Acknowledgments

Next time: More on Graphs, and fixing cheaters: link spam and topic-specific variants of PageRank!

Some material is adapted/adopted from Mining of Massive Data Sets, by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University) `http://www.mmds.org`



Special thanks to Tony Wong for sharing his original adaptation and adoption of slide material.