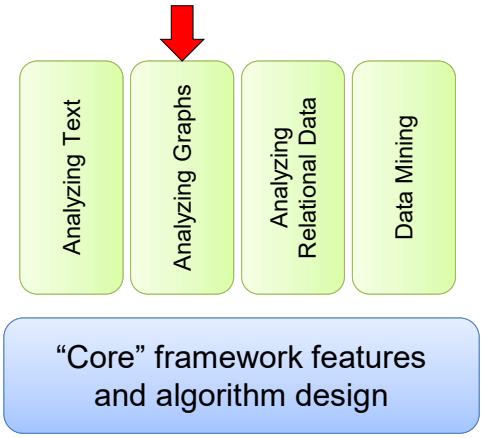
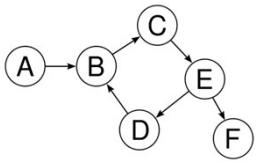
Data-Intensive Distributed Computing CS431/451/631/651

Module 5 – Analysing Graphs

Structure of the Course



What's a Graph



- G = (V, E) (set of vertices and set of edges)
- Direction:
  - Undirected – edge (u, v) implies edge (v, u)
  - Directed – edge (u, v) does not imply edge (v, u)
- Edges may or may not be labelled
  - (Numerical labels are usually called "weights")
- This should not be news to anybody!

Vocab Reminders

- Vertex (or “node”) – The circle thingies
- Edge (or “link”) – connects one Vertex to another
  - (Or to itself, perhaps)
- If there is edge (u, v) then:
  - v is an “out-neighbour” of u
  - u is an “in-neighbour” of v
- In-Degree(v) – how many edges lead into v
- Out-Degree(v) – how many edges lead out from v

Example Graphs

- Roadways, Water mains, power lines, other SimCity things.
- Social Networks (a person is a vertex, edges are “friends”)
- Actual computer networks
- The internet (just a big network, innit?)

Most graphs are sparse: number of edges is closer to $|V|$ than it is to $|V|^2$

Most graphs we’re interested in, anyway

Representation

In the past you've probably seen three representations

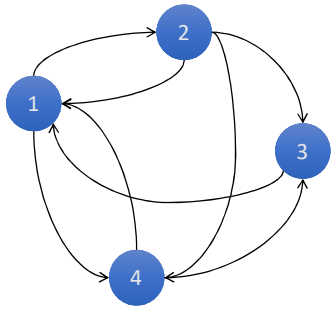
- Adjacency Matrix
- Adjacency List
- Edge List

Adjacency Matrix

An n x n matrix M.

M_{ij} = 1 iff there's an edge between v_i and v_j

	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0



Adjacency Matrix

PRO:

- Who doesn't love a matrix?
- Useful for mathematical operations
- CUDA loves matrix operations
- Fast neighbourhood check (both in and out)

CON:

- Mostly 0s in a sparse graph (wasted space)
- O(n) to find neighbourhood of v (in and out)

Other con from our perspective – how do you partition this for parallel work???

Adjacency List

Row i is a list of the out-neighbours of vertex v_i

	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

Same idea as posting lists, right?
Also hey, now it's a stripe? Key – vertex label Value – sparse adjacency vector

Adjacency List

PRO

- Smaller than the matrix (especially for small graphs)
- Even easier to find out-neighbours of v (directly stored)

CON

- Difficult to find in-neighbours of v
  - have to search all other adjacency lists

The con isn't really a con per se – if you need in-neighbours you can just have a second adjacency list for in-neighbours. This doubles the amount of storage needed, but for a sparse graph it's still much improved from the adjacency matrix.

Edge List

- Just a list of all the edges

	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

(1, 2)
(1, 4)
(2, 1)
(2, 3)
(2, 4)
(3, 1)
(4, 1)
(4, 3)

Though they're in sorted order here, that's because of how the matrix is converted to an edge list, not because it's a necessary part of the format!

Edge List

PRO

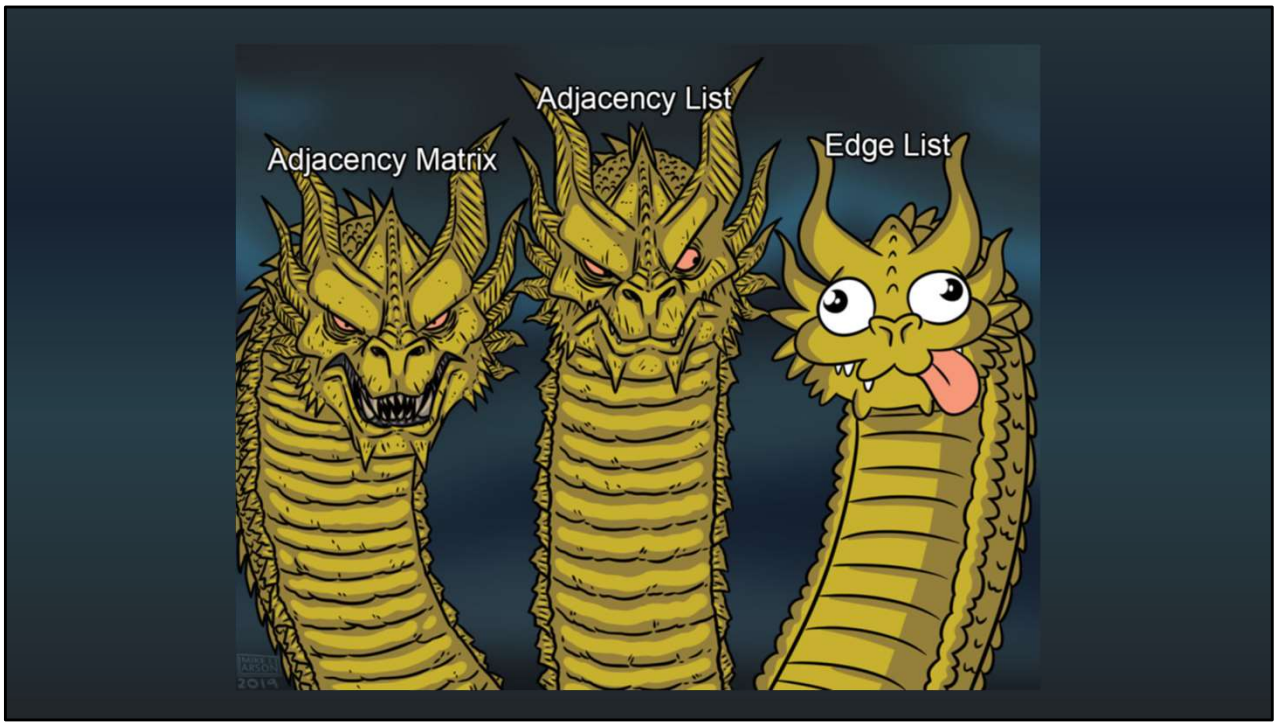
- Easy to add an edge (just append)
- Simple

CON

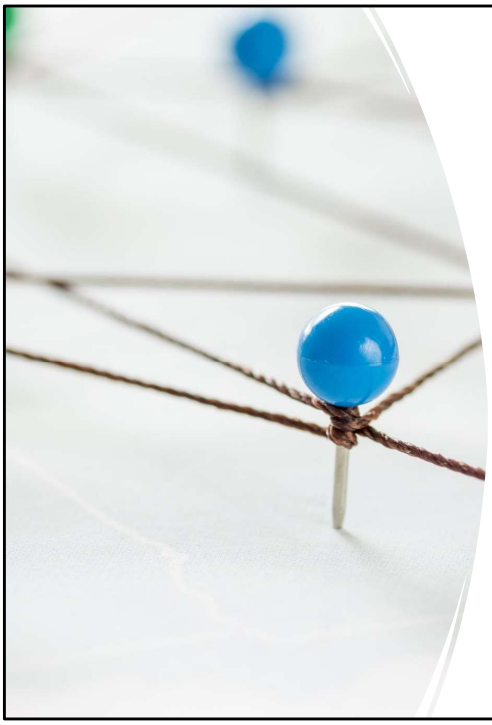
- Hard to find neighbours
- Hard to find anything, really

All they've got going for them is simplicity. It's OK if you're building a graph by generating edges all over the place…but to do anything useful with the graph, you'll want to then convert to a more suitable representation.

LIFE-HACK: If you have an edge list, groupByKey will turn it into an adjacency list!



Often an edge list is easy to generate – you're extracting a graph representation from unstructured data. **However**, the only thing you do with it is group the edges into one of the other representations!



Graph Problems

- Shortest Path – Google Maps, Delivery Planning
- Minimum Spanning Tree – Utility Lines
- Min-Cut – Utility Lines, Disease Spread
- Max-Flow – Airline scheduling
- Graph Colouring – Planning Final Exams
- Bi-Partite Matching – Dating Sites
- PageRank – Google

What does the web look like?

- 4.77 billion pages – 50 billion (vertices)
  - It changes by a lot every day
- 100 billion – 1 trillion links (directed edges)

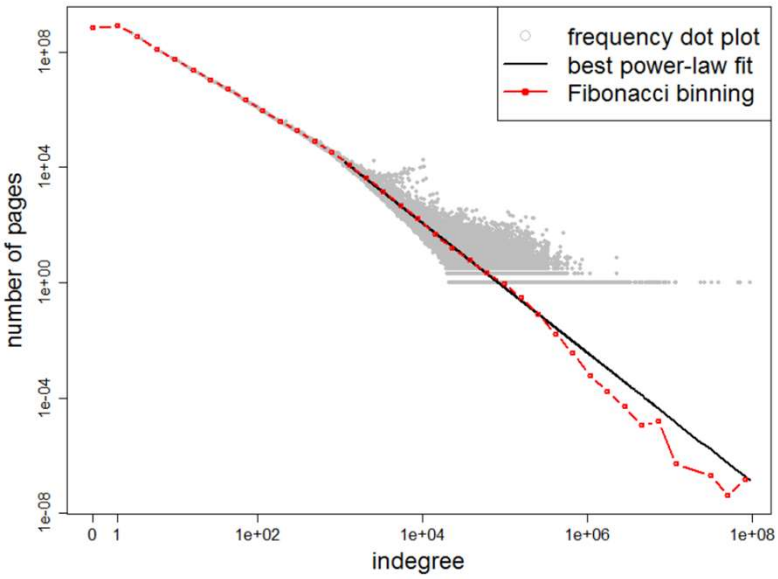
Sources: worldwidewebsize.com, Tilburg University Research

That's big

Power laws again:

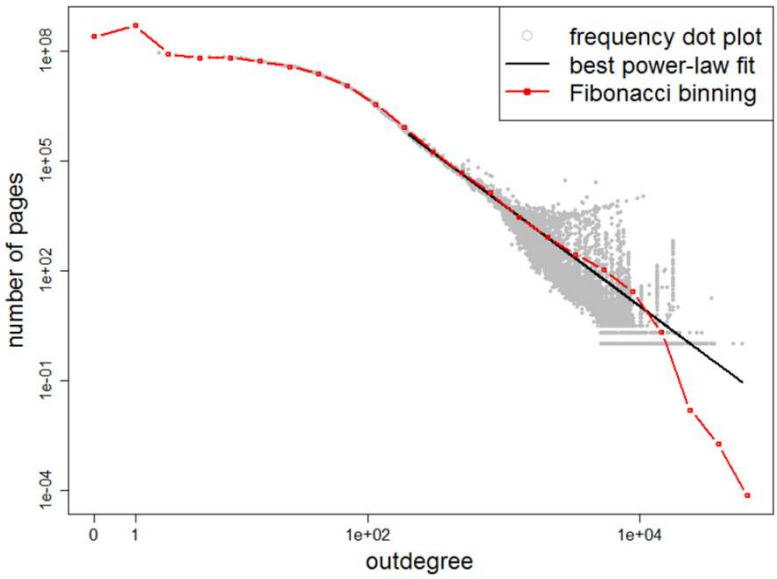
Fraction of pages that have k links: $P(k) \sim k^{-\gamma}$

"Dark Web" means pages not indexed by Google / Bing, it's not nefarious per se.

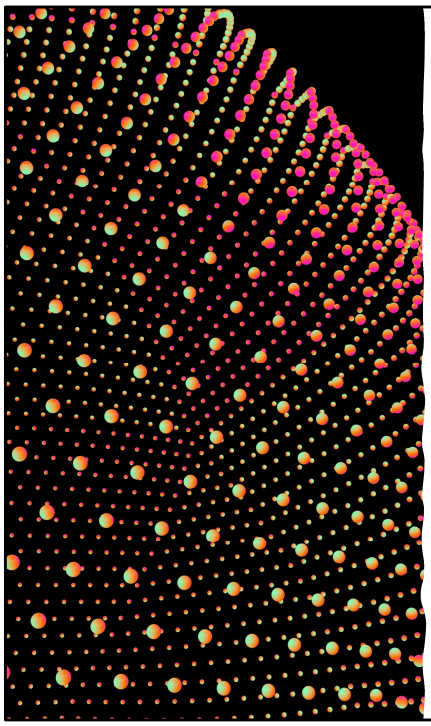


Lots of pages have 0 or 1 in-links

Fibonacci binning – like regular binning but the bin size increases by the Fibonacci sequence – use this instead of linear binning if you're going to be plotting in log-log space. Linear binning is for linear space!



The most common out-degree is 1



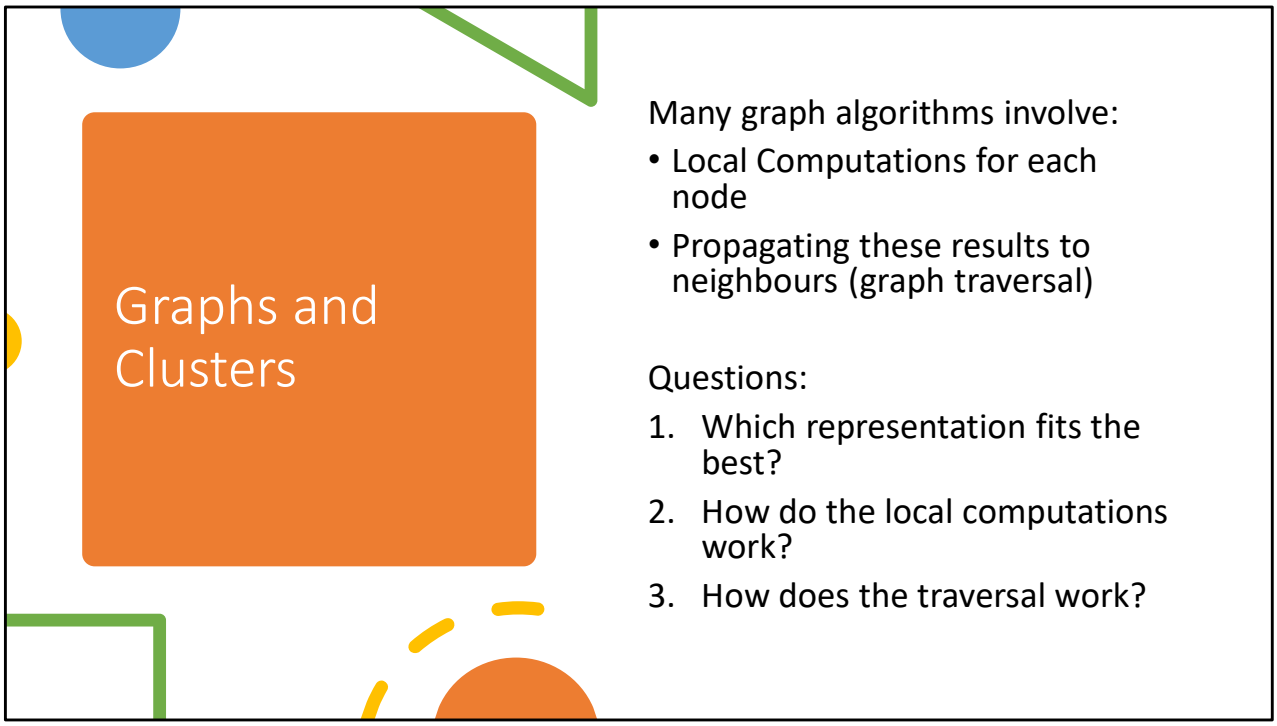
So when you say big, you mean…

60+ GB

The web graph is Big Data™

Clearly, we need Hadoop!

How do we do this on MapReduce (or Spark)?


Graphs and Clusters
Many graph algorithms involve:
• Local Computations for each node
• Propagating these results to neighbours (graph traversal)
Questions:
1. Which representation fits the best?
2. How do the local computations work?
3. How does the traversal work?

Answer Key

- (Dan hopes there was some discussion there)
- Local computation means **INDEPENDENT**
  - Sounds very much like a map-like task
- Adjacency List makes the most sense
  - KVP – key is a vertex, value is its adjacency list
- Propagation
  - Shuffle – Collecting the propagated messages is reduce-like

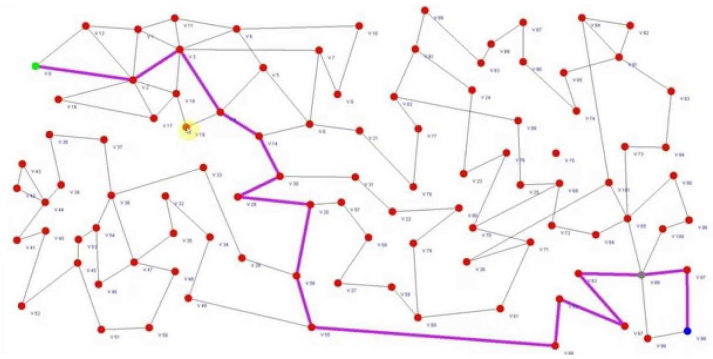
It's reduce-like as sending messages to neighbours is a non-local computation.
Thought – If we override the hash partitioner we can try to partition in such a way to minimize the number of inter-partition edges.

(Often just doing a BFS will work as there are lots of cliquey sorts of components to the graph. See Bespin's Range Partitioner)

Single-Source Shortest Path

Problem: Find the shortest path from a single node (source) to all other nodes

(shortest might mean fewest links, or lowest total weight)



Dijkstra's Algorithm

- You 100% have seen this in CS240 or CS234, don't even pretend

Step 1

Set all nodes as unvisited, with D = infinity

Set source node's D to 0

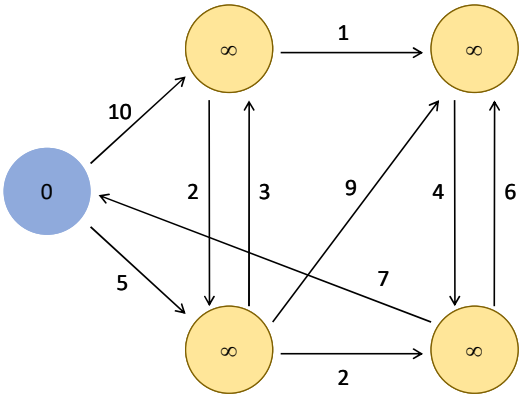
Step 2

Let v be the unvisited node with lowest distance

For all out-neighbours u of v:

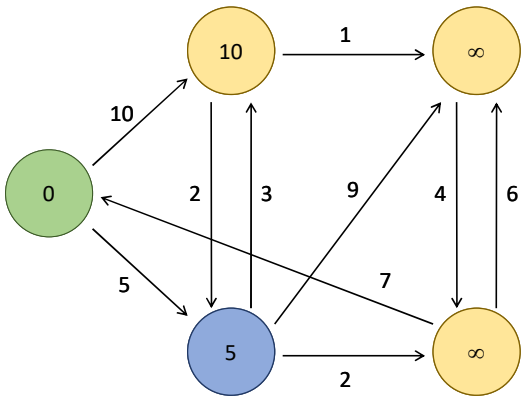
D[u] = min(D[u], D[v] + edge(v, u).weight)

Dijkstra's Algorithm Example



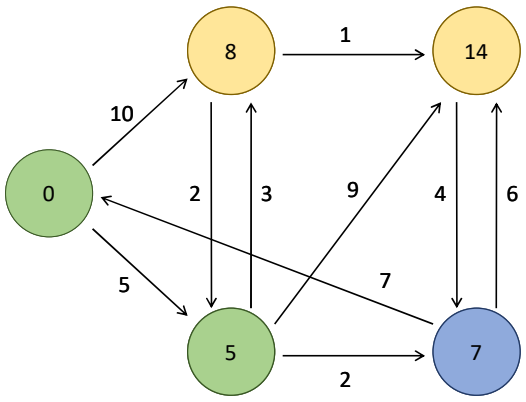
Example from CLR

Dijkstra's Algorithm Example



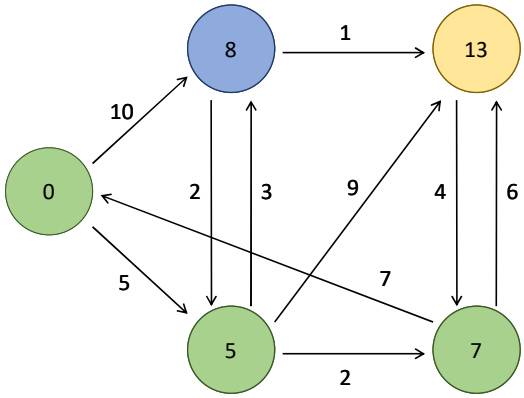
Example from CLR

Dijkstra's Algorithm Example



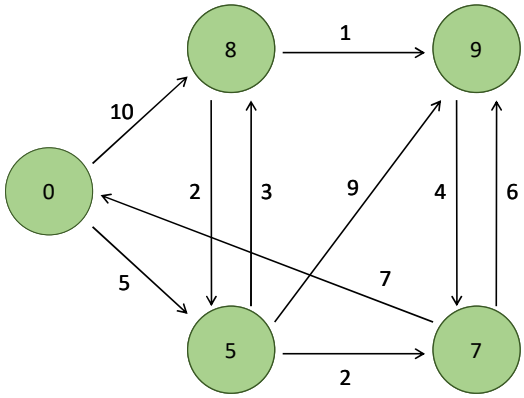
Example from CLR

Dijkstra's Algorithm Example



Example from CLR

Dijkstra's Algorithm Example



Example from CLR

Dijkstra on MapReduce

Not Parallel ☹	
Why?	• “Minimum D” – not local
So instead, we use:	• Parallel Breadth-First Search (pBFS)

Simple Case – Unweighted Graph

Want the fewest "hops" from source to destination

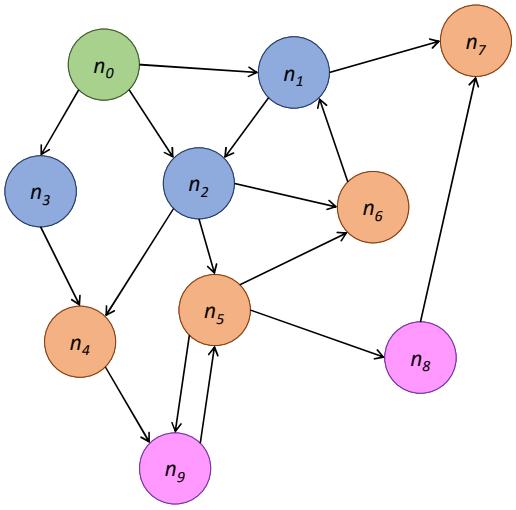
Inductive Definition

- dist(s) = 0
- dist(v) = 1 if there is an edge from s to v
- dist(v) = 1 + $\min_u$ (dist(u)) (for all u s.t. there is an edge u -> v)

I couldn't get a non-ugly way to express that in the little u for min

Simple Case – Unweighted Graph

- Iteration
  - 0
  - 1
  - 2
  - 3



Implementation on Hadoop

Keys – node n

Values – (d -- distance to n, adjacency list of n)

Mapper:

for m in adjacency list, emit (m : d + 1)

also emit (n : d)

Reducer:

Update distance to node n based on (m : dist) messages from mapper

Iteration

To iterate, the output of the reducer gets passed to another (identical) job as the input.

How? Isn't the adjacency list gone?

UGH. Yes. OK, so…also emit the adjacency list…

Pseudocode

```
def map (id, node):
  emit(id, node)
  for m in node.adjList:
    emit(m, node.d + 1))
```

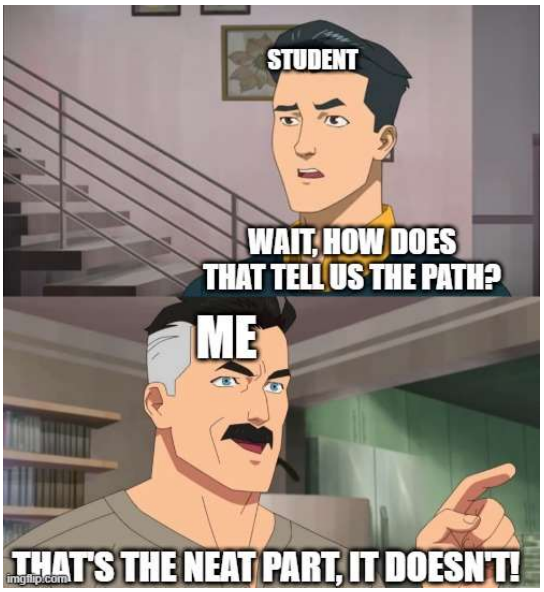
```
def reduce (id, values):
  d = infinity
  node = None
  for o in values:
    if isNode(o):
      node = o
    else:
      d = min(d, o)
   node.d = min(node.d, d)
   emit(id, node)
```

Python-like Pseudocode for maximum brevity

Node is a (Writable) object with 2(?) fields:
d – distanceFromSource
adjList: A list of IDs for the out-neighbours of the node

With Hadoop MapReduce, the Value type can be ObjectWritable, and you can use reflection to see what kind of object it actually is.
(With Scala Spark you can do basically the same thing, and with PySpark it'll be "automatic")

Improvement: a "last updated" field – in iteration k, if id node was not last updated in iteration k-1, do not emit messages from the mapper



It's not actually hard to modify this to include the path.

Option 1: Node has a field "path that results in given d"

Option 2: Node has a field "previous"

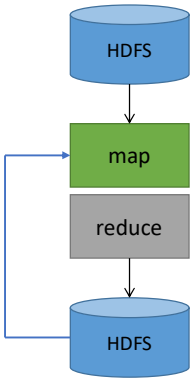
Option 1 – Bigger messages, bigger final output

Option 2 – Reconstructing the path from the final output is more time consuming (but presumably the path is short, so not really)

MapReduce Iteration

1. Do the job
2. If we need to keep going, make a new job that reads the output from the last job
3. See step 1

What's the issue here?



This feels a lot like the diagram was taken from the "Why Spark is better than MapReduce" slides. (That's because it was)

Adjacency lists are big, and are sent needlessly!
Lots of "to disk and back" forced by MapReduce Framework

Quitting Time

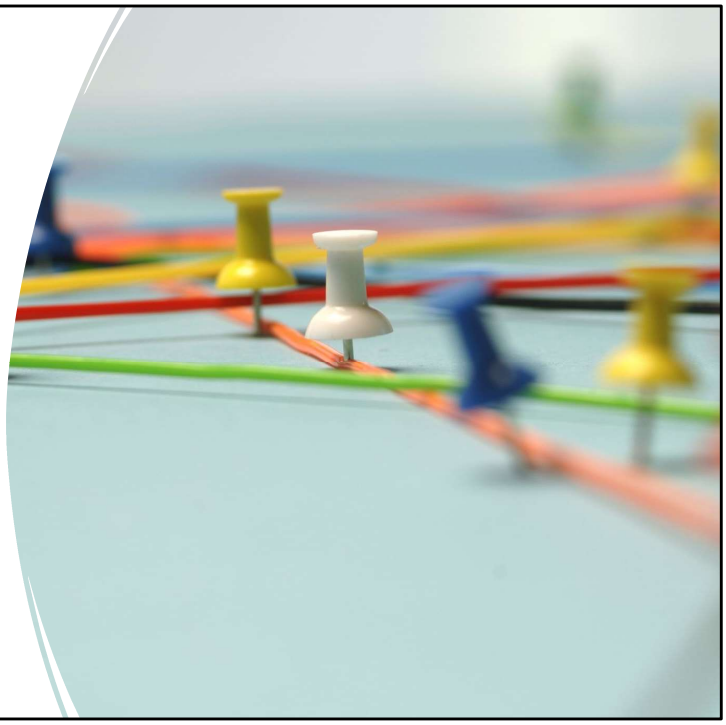
- The most important part of recursion is knowing when to stop
- The second most important part of recursion is knowing when to stop
- The third most important part of recursion is knowing when to stop

<pause for laughter>

When should this iteration stop?
If no node's distance gets updated, then the next iteration will send the same messages

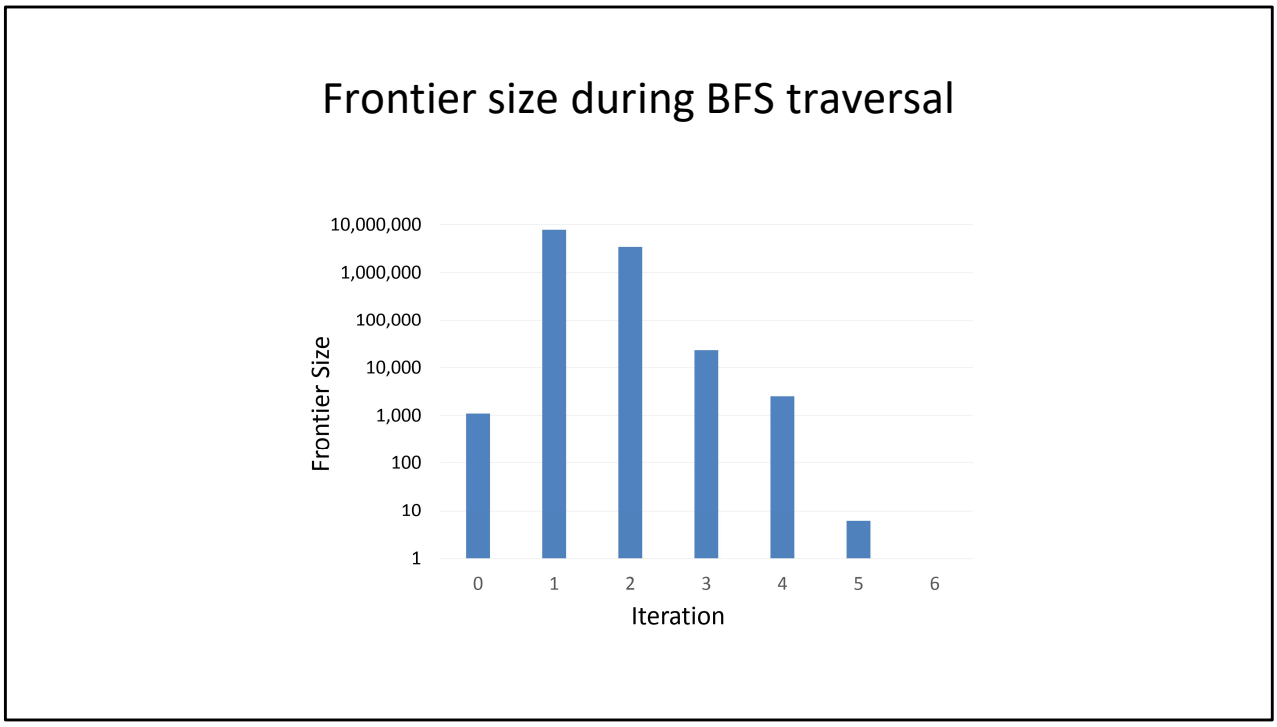
Kevin Bacon

- All nodes with min distance of k are “visited” in iteration k
- Why? Did you not see the colourful diagram???
- So how many to search the entire graph?
  - 6?



It’s not actually 6. Depends on the graph, really.

https://www.csauthors.net/distance



Six, six is good.
This is something like 16M nodes with an average of 1000 links per node.
It'll vary with the distribution of links.

The gnutella p2p graph for the graph assignment, the frontier will still be going after 11 iterations.

Pseudocode (Weighted Edges)

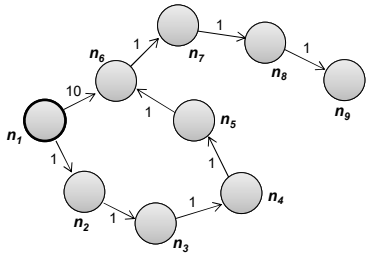
```
def map (id, node):
  emit(id, node)
  for m in node.adjList:
    emit(m.id, node.d + m.w))
```

```
def reduce (id, values):
  d = infinity
  node = None
  for o in values:
    if isNode(o):
      node = o
    else:
      d = min(d, o)
   node.d = min(node.d, d)
   emit(id, node)
```

That's it, that's the only change

Weighted Edges, Termination

- You can still update a note after you first “discover” it
  - Fuzzy Frontier
- Stopping condition unchanged
  - Stop when no changes
- More iterations needed
  - Could be a lot more



# of iterations needed
Unweighted – length of the longest “shortest path” [usually short]
Weighted – length of the longest cycle-free path

BFS vs Dijkstra

- Dijkstra only investigates the lowest-cost node on the frontier
- Parallel BFS investigates all paths in parallel
  - It's a simple optimization to restrict to the frontier
  - But you're still sending the adjacency list back and forth

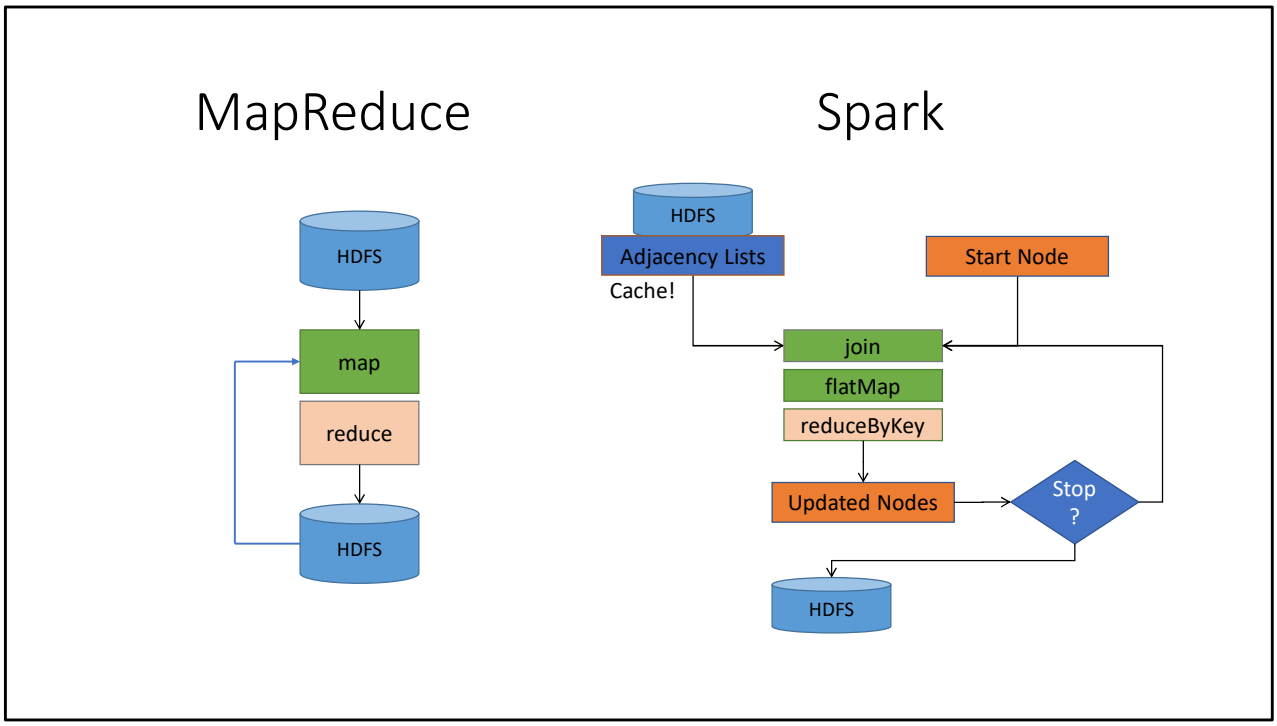
- Can we do better on MapReduce?

No, we don't have global state

Issues with MapReduce Iteration

Everything is written to HDFS, then loaded again

We must send the entire graph structure to the reducers each iteration, only to have them send it back again



Legend: Green Box – A local transformation (no shuffles, i.e. narrow dependencies). Pale Orange – Reduce-like transformation, wide dependencies (shuffles)
Blue Box – A cached RDD
Dark Orange box – Uncached RDD

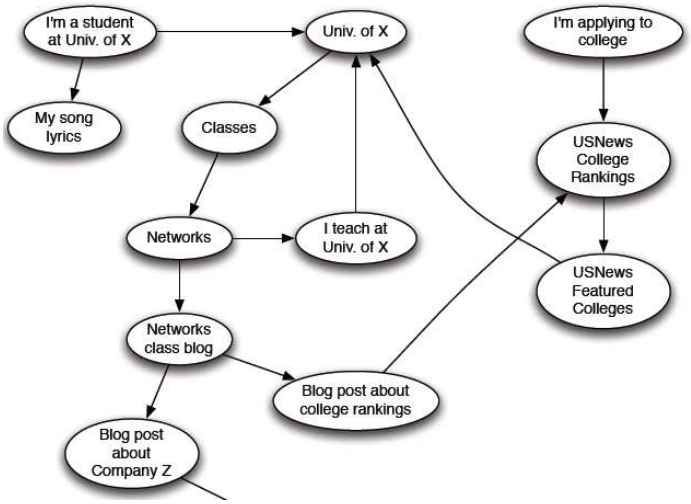
Note that although there are two reduce-like transformations (join and reduceByKey) they have the same keys so there will be narrow dependencies (meaning no shuffles). Green means no shuffles. -- well, the start node gets shuffled, but it's literally one value out of billions. Doesn't count.

Conceptually: Each node receives "here's your best path" and then announces this to its neighbours.
Probably there is more than one node per partition…but still…

Something Borrowed

- Next Slides are thanks to Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University)

Web as a Directed Graph

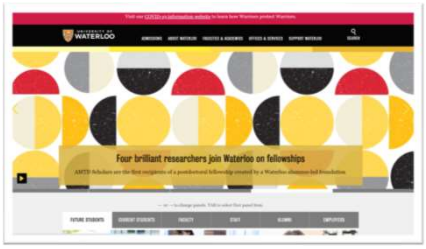


45

Who to trust?

Query: University of Waterloo

uwaterloo.ca



fakeuw.ca

Ranked retrieval fails!

46

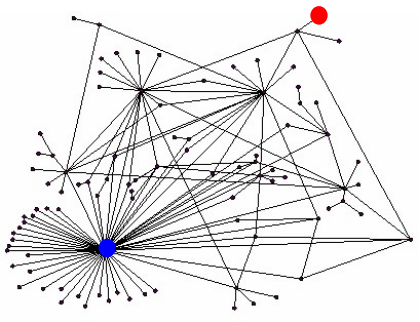
Web Search Challenge

- **Web contains many sources of information**
  Who to "trust"?
  - **Trick:** Trustworthy pages may point to each other!

Ranking Nodes on the Graph

- **All web pages are not equally "important"**
  www.joeschmoe.com vs. www.stanford.edu

- There is large diversity in the web-graph node connectivity. **Let's rank the pages by the link structure!**



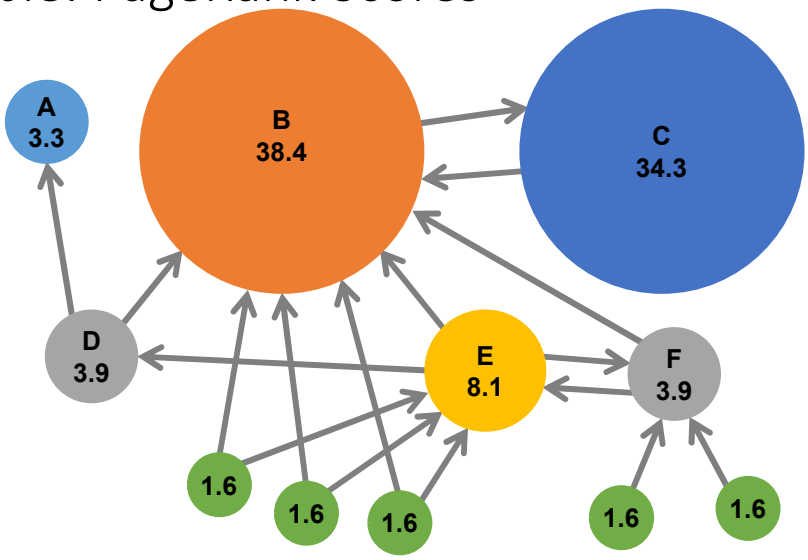
48

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.stanford.edu has 23,400 in-links
  - www.joeschmoe.com has 1 in-link
- **Are all in-links equal?**
  - **Links from important pages count more**
  - Recursive question!

Example: PageRank Scores

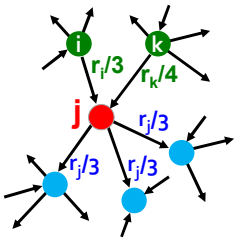


51

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

$r_j = r_i/3 + r_k/4$



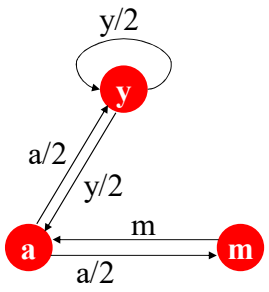
52

PageRank: The "Flow" Model

- **Define a "rank" r_j for page j**

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

d_i … out-degree of node i



"Flow" equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

53

Solving the Flow Equations

Flow equations:

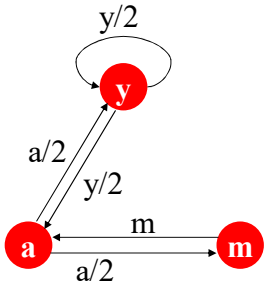
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

- **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!**

54

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



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

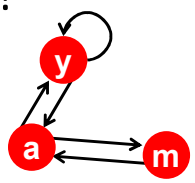
55

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**

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

Iteration 0, 1, 2, …



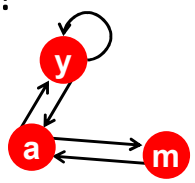
	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

56

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

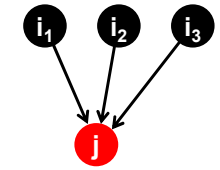
$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

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

Iteration 0, 1, 2, …

Random Walk Interpretation

- **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

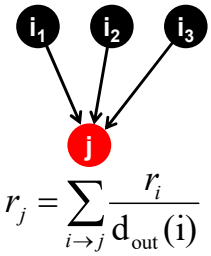


$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

58

Students, please, I am begging you, understand this one thing if you understand anything: There isn't ACTUALLY a random walker. We're not going to simulate a person. We're going to IMAGINE there is such a person, and construct a **distribution** of what page that **might** be on.

Random Walk Interpretation



- **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

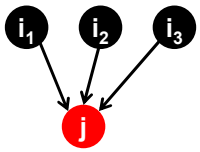
59

The Stationary Distribution

- **Where is the surfer at time *t+1*?**
  - Follows a link uniformly at random

$$\boldsymbol{p(t+1) = M \cdot p(t)}$$

- Suppose the random walk reaches a state $\boldsymbol{p(t+1) = M \cdot p(t) = p(t)}$
  then $\boldsymbol{p(t)}$ is **stationary distribution** of a random walk



$$p(t+1) = \mathrm{M} \cdot p(t)$$

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 the initial probability distribution at time **t = 0**

61

Dan adds: "certain conditions"???
Ah, here's a textbook definition: A graph is "ergodic" (cool word) if there exists a distribution pi such that for all initial distributions p_0 $\lim_{t->inf} p_t$ = pi

What is required to be ergodic?

1. It must be connected and not bi-partite.
    1. Connected is obvious, as if there's a disconnected component, initial distribution with all 0s in that region will never converge to a distribution with anything but 0s in that component
    2. Bi-partite is less immediately obvious, but we'll see in example in a couple more slides!

Hold up, what conditions?

In an *ergodic* graph, no matter the initial distribution, a random walk will always converge on a unique stationary distribution

Necessary and Sufficient Conditions:

- The graph must be connected
- The graph must be *vertex aperiodic*
  - $\forall v$ the length of all cycles involving v must have a GCD of 1

PageRank:
The Google Formulation

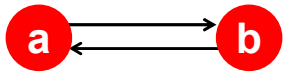
PageRank: Three Questions

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

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

64

Does this converge?



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

- **Example:**

	=			
r_a	1	0	1	0
r_b	0	1	0	1

Iteration 0, 1, 2, …

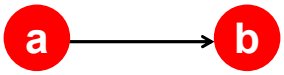
65

Dan: Clearly it does not converge in this case (bipartite graph, as promised). Fortunately, the internet as a whole is not bipartite. So we're good? (No, we are most certainly not)

Anyway here's another cool word for you: vacillate – to waiver between two options.
In a bi-partite graph, the two partitions (in the graph sense, not the Spark sense) will have X, 1-X as their total "importance" in the initial distribution, and
Will vacillate between (X, 1-X) and (1-X, X). That's because each node in partition 1 will send all of its importance to partition 2, and vise versa.
In terms of a single walk, instead of a probability distribution, the walker will alternate between which partition they are in.

However, the requirement is stronger than "not bipartite" but, again, "vertex aperiodic"

Does it converge to what we want?



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

- **Example:**

	=			
r_a	1	0	0	0
r_b	0	1	0	0

Iteration 0, 1, 2, …

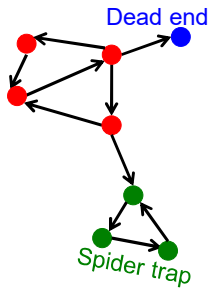
66

Dan: Clearly not. This says “neither site has any importance”…and the importance has been lost. Does not sum to 1! This is not a connected graph.

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"
  - **NOT CONNECTED ⟹ NOT ERGODIC**



- **(2) Spider traps:** (all out-links are within the group)
  - Random walker gets "stuck" in a trap
  - And eventually spider traps absorb all importance
  - **NOT CONNECTED ⟹ NOT ERGODIC**

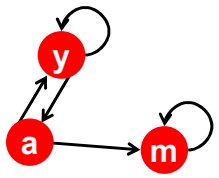
67

Dan: The internet has dead ends, and "spider traps" – the bipartite subgraph was just a simple example of that. So the internet has neither property required for random walks to converge to a stationary distribution! That's not good!

Spider Trap? The bots that crawl the web are called "spiders" (because they crawl the web, you see). PageRank is based on an "imaginary" spider, and that spider would get trapped in this topology. (Google's actual spiders might too, but probably have a queue system and never follow a link to a page they've visited before…at any rate we're not going to talk about that particular hurdle)

Problem: Spider Traps

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



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2$
$r_m = r_a/2 + r_m$

- **Example:**

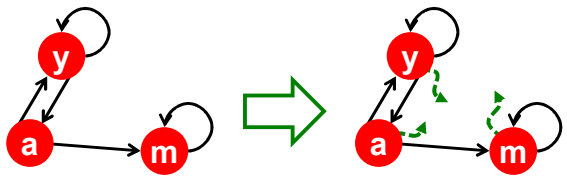
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

All the PageRank score gets "trapped" in node m.

Solution: Teleports!

- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-β**, jump to some random page
  - Common values for β are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**

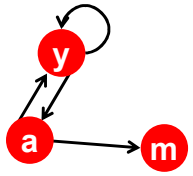


69

The 1999 PageRank paper uses “d” (stands for “damping factor”) instead of Beta.

Problem: Dead Ends

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



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2$
$r_m = r_a/2$

- **Example:**

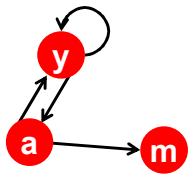
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

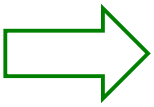
Iteration 0, 1, 2, …

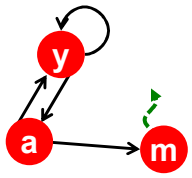
Here the PageRank "leaks" out since the matrix is not stochastic.

Solution: Always Teleport!

- **Teleports:** Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly







	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Why Teleports Solve the Problem?

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

72

Alternative solution to dead-ends: add a self-loop. Then a dead-end becomes a spider trap. (This however will add importance to dead-ends, and as we saw, dead ends are actually the second most common type of page, after pages with only a single outbound link)

Solution: Random Teleports

- **Google's solution that does it all:**
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1\text{-}\beta$, jump to some random page

- **PageRank equation** [Brin-Page, 98]

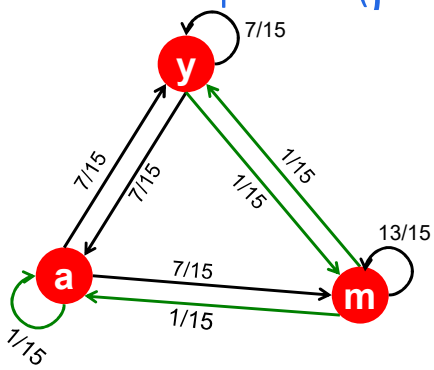
$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta)\frac{1}{N}$$

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.

73

Random Teleports ($\beta = 0.8$)



M

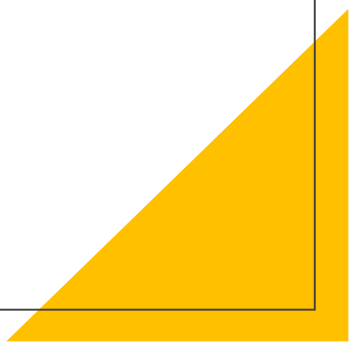
$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}_{NxN}$$

$$\mathbf{A} = \begin{array}{c} \\ y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

$$\begin{array}{c} \\ y \\ a \\ m \end{array} = \begin{bmatrix} 1/3 & 0.33 & 0.24 & 0.26 \\ 1/3 & 0.20 & 0.20 & 0.18 \\ 1/3 & 0.46 & 0.52 & 0.56 \end{bmatrix} \cdots \begin{bmatrix} 7/33 \\ 5/33 \\ 21/33 \end{bmatrix}$$

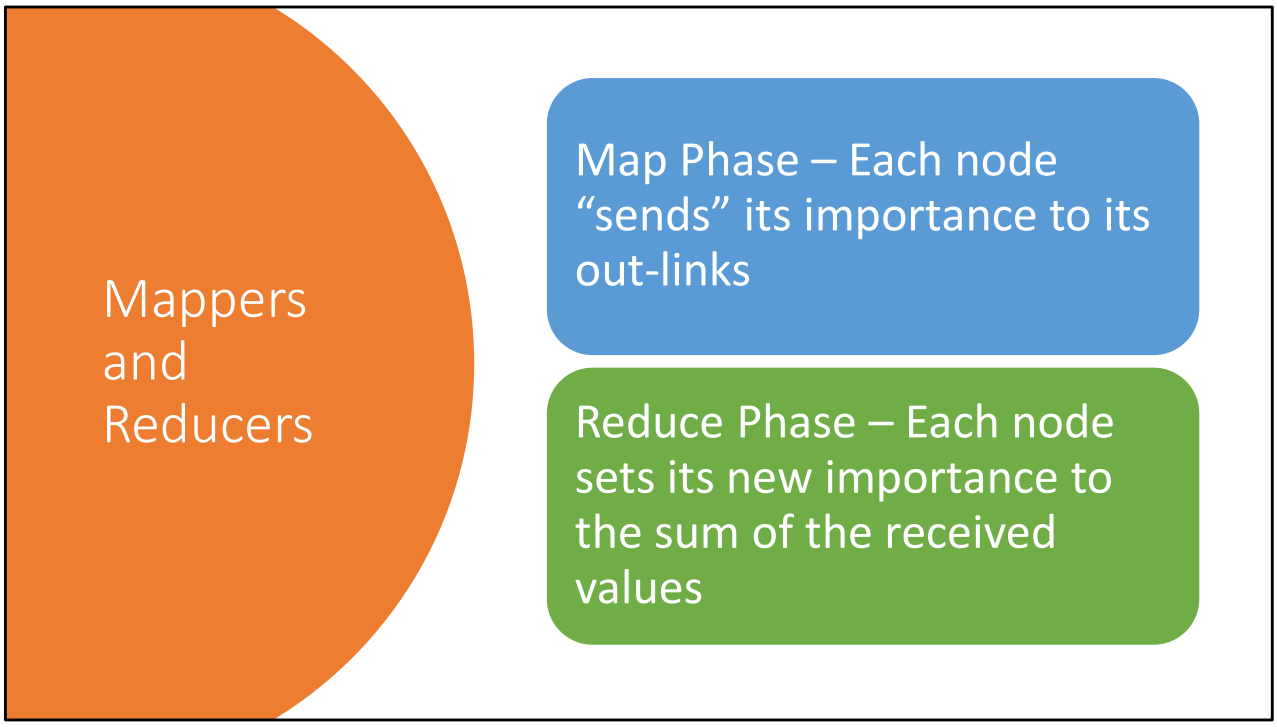
PageRank with MapReduce

We now return to Dan's slides



Any errors are now Dan's fault.

Ah, giant orange circle, I missed you.


Mappers and Reducers
Map Phase – Each node "sends" its importance to its out-links
Reduce Phase – Each node sets its new importance to the sum of the received values

Pseudocode

```
def map(id, n):
  emit(id, n)
  p = n.rank / len(n.adj)
  for m in n.adj:
    emit(m, p)
```

```
def reduce(id, msgs):
  n = None
  sum = 0
  for o in msgs:
    if o is Node:
      n = o
   else:
     s += o
  n.rank = s
  emit(id, n)
```

The way bespin does the whole “is node” is a custom class, “is a node” is just a field indicating the type of the message

PageRank vs. BFS

	PageRank	BFS
Map	PR/N	d+1
Reduce	sum	min

A large class of graph algorithms involve:

Local computations at each node

Propagating results: "traversing" the graph

79

Missing two things:

- Random Jumps (1-β)
- Dead End Jumps (always)

How can we add these?

Random Jumps

- Every node has the same chance of jumping: $1 - \beta$
- If it jumps, it jumps to ANY random page (1/N for a particular page)
- Only need to change reducer side

New Reducer

```
def reduce(id, msgs):
  n = None
  sum = 0
  for o in msgs:
    if o is Node:
      n = o
   else:
     s += o
  n.rank = s * β + (1 – β) / N
  emit(id, n)
```

What????

Well, all N send (1 – β) of their rank to all N nodes in the graph. Ranks sum to 1, so the entire network is sending a TOTAL of (1 – β) split evenly across the network. Each node then gets (1 – β) / N of that

What about dead-ends?

Dead-Ends send all their weight everywhere, instead of only some of it

- Option 1: Replace dead-ends with “links to everyone, everywhere”
  - No changes to the code
  - That’s a lot of messages
- Option 2: Post-process to redistribute “missing mass”
  - Avoids making N-degree nodes
  - Aww man, I have to think?

Option 1 is REALLY BAD. Why? We saw the scatterplot, we know that the second most popular kind of page is a dead-end! Now suddenly the rank 2 goes from degree 0 to degree N. The assumption of a sparse graph is now wrong.

Post-Processing

R = sum of all ranks

1 – R = “missing mass” (sum of ranks of dead ends)

Add (1 – R)/N to all nodes

See Bespin for details.

Seems like a lot of work to compute R? Do we need to load all of the files in the driver? (NO, do not do that!)
Opposite of side-loading: side-unloading? Have the reducer write a file that contains its portion of R. Driver just loads these files (small)

Easier with Spark – double accumulator (MapReduce counters only count integers)

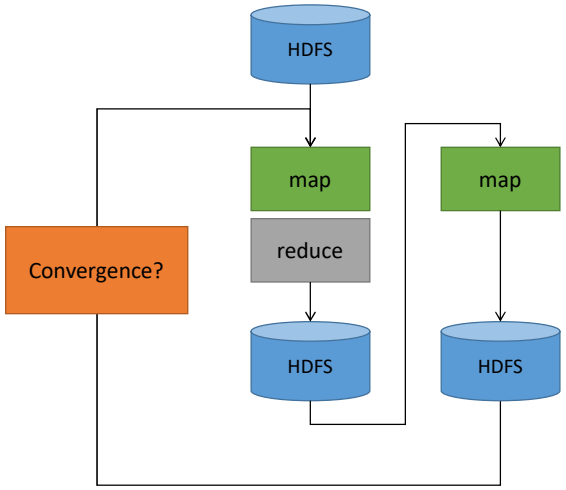
Have to do an extra pass though. Map (send mass) -> Reduce (compute new mass) -> Map (add missing mass)
OR DO WE? DUN DUN DUN
Map (add missing mass and then send mass) -> Reduce (compute new mass)

Only need one extra map as the FINAL step. After iteration finished, add missing mass one last time.
OR – Just leave it missing? If dead ends have lost some mass, we’re just spreading that evenly across all nodes. This won’t change rankings.

Post Processing

Note: You can move the “post processing” map into the “next map”

(Just remember to adjust the convergence test)



Pretty sure I’ve mentioned this before, but if you set the number of reducers for a MapReduce job to 0, then it will be a map-only job –
The “intermediate” files from the Map task will be the final output of the job.

Alternative

Mappers send:

- node.rank divided evenly amount out-links
- special case: dead-ends send entire rank to “everyone” (everyone being a special key)

Reducer adds (1/N) x “everyone” rank to each sum

Complication – you must send one “everyone” per reducer PER NODE. That’s a lot of messages! ☹
But: Collect “everyone” using IMC (in-memory combiner) and each mapper sends 1 extra message per reducer.

How to send to all reducers:

```
for i in range(numReducers):
  emit(EVERYONE, (i, missingMass)
```

Partitioner, if key is EVERYONE, will send this to reducer i (taking advantage of the fact that a MapReduce partitioner can examine both key and value)

As usual: sort the special messages first, reducer holds “lost and found” total in memory.

Small Problem

N = 1 billion. The individual masses will be small.

Solution: store the logs

This isn't really a problem conceptually, just a limitation of floating point representations.

Log Masses

$a = \log m\,, b = \log n$

$m \times n \;\rightarrow a + b$

$$m + n \rightarrow \begin{cases} b + \log(1 + e^{a-b}) & a < b \\ a + \log(1 + e^{b-a}) & a \geq b \end{cases}$$

For once, log is natural log, not base 2 or base 10

Why Log Masses Work

- Ranks are in [0,1] -- or, really, (0,1)
- Mathematically, ~ ¼ of all float values are also in this range
  - Sign + = ½ , exponent negative = ½
- Log(x) : [0,1] => [-∞, 0]
  - Now we're using ½ of the values instead of ¼
- It also lets us store MUCH smaller numbers without underflow
  - Most pages will have a very small but non-zero PageRank so this is important

Ugh, this slide isn't wrong, it's just missing the point. Sorry ☹

The first point doesn't help much. 1 extra bit of precision.

The second point is true, but unnecessary for PageRank
A double can represent values on the order of 2^{-1023} , i.e. 1e-308 as normal probabilities, but for logs, this can go so close to 0 that most calculators just say 0.

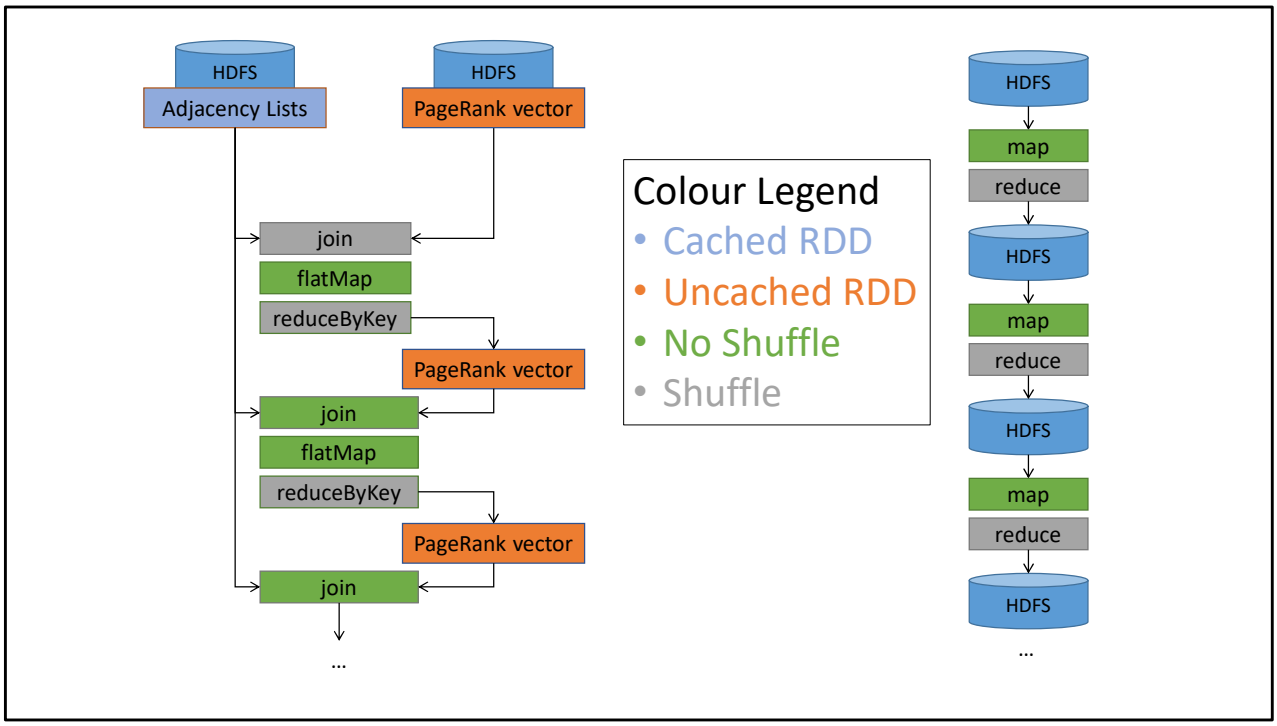
Still, even with 100 billion pages, a double can go small enough that even pages with the lowest possible rank and highest possible out degree would not come anywhere close to the smallest representable magnitudes for a Double type.

So why do we do it if our numbers aren't small enough to warrant it? Well, because what we've really done is made the operations more mathematically stable. The main underflow risk here is not that the numbers get too small to represent, but we're likely to be adding small and large numbers. (E.g. one node sends 1e-12 rank, and another sends 1e-2, which will result in a lot of rounding of error. With log probabilities we end up compressing the range used so there will be less rounding off errors!)

Let's Spark

Problems with MapReduce

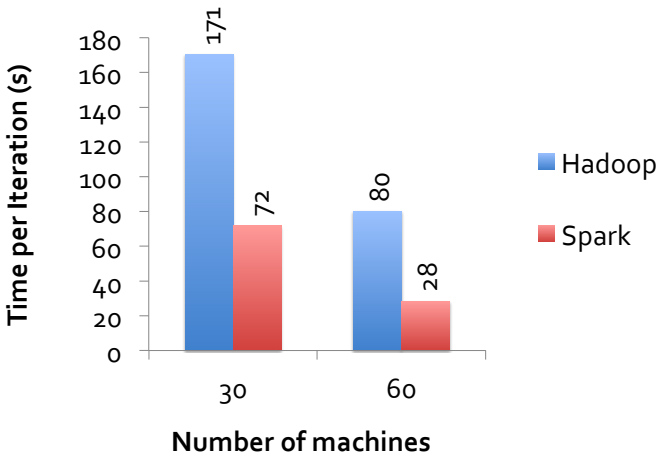
- Sending Adjacency lists with each iteration (never changes)
- Needless Shuffling
- Needless Filesystem Access
- Verbose programs
- Each iteration is a new job
  - Hadoop has a fairly long start-up time per job



Note – the joins (other than the first one) won't need to shuffle if you partition the adjacency list by key ahead of time –
If you don't, well, git gud. Sorry, I mean "you should try to design your algorithms to avoid shuffles as much as you can manage"

After reduceByKey, the PageRank vector will be partitioned by node ID. If the adjacency list is too, then we'll have narrow dependencies.

MapReduce vs. Spark



Source: http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/matei-zaharia-part-2-amp-camp-2012-standalone-programs.pdf

PageRank in Spark

```
val lines = sc.textFile("the-internet.txt")
val damp = 0.85
val links = lines.map{ s =>
     val parts = s.split("\\s+")
     (parts(0), parts(1))
}.distinct().groupByKey().cache()

var ranks = links.mapValues(v => 1.0)

for (i <- 1 to iters) {
  val contribs = links.join(ranks).values.flatMap{ case (urls, rank) =>
    val size = urls.size
    urls.map(url => (url, rank / size))
  }
  ranks = contribs.reduceByKey(_ + _).mapValues((1 - damp) + damp * _)
}
```

Source - https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/SparkPageRank.scala

I pulled “damp” out as a constant. No magic numbers!
I also removed boilerplate for setting up SC, and changed the textFile to be a funny name instead of a parameter…

Note that this assumes that there are no dead links. It also scales everything by N (meaning: masses sum to N, not to 1).

PageRank in PySpark

```
lines = sc.textFile(“the-internet.txt”).map(lambda r: r[0])
# Loads all URLs from input file and initialize their neighbors.
links = lines.map(lambda urls: parseNeighbors(urls)).distinct().groupByKey().cache()

# Loads all URLs with other URL(s) link to from input file and initialize ranks of them to one.
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

# Calculates and updates URL ranks continuously using PageRank algorithm.
for iteration in range(iterations):
        # Calculates URL contributions to the rank of other URLs.
        contribs = links.join(ranks).flatMap(lambda url_urls_rank: computeContribs(
            url_urls_rank[1][0], url_urls_rank[1][1]  # type: ignore[arg-type]
        ))
        # Re-calculates URL ranks based on neighbor contributions.
        ranks = contribs.reduceByKey(add).mapValues(lambda rank: rank * 0.85 + 0.15)
```

Source - https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py

I didn’t include the helpers but you can go to the URL for the details.
Note that the input format is different, so you’ll need to write your own anyway!
Note that this assumes that there are no dead links. It also scales everything by N (meaning: masses sum to N, not to 1).

PageRank in MapReduce

See Bespin Implementation

I can't paste it here, it's 600 lines long


Page Rank Improvements
96

Remember Search?

Old Search Ranking – TF and DF and logarithms

Flaw: Term Spam.

Set div to not render: Spam spam spam spam spam spam spam spam …

New Search Ranking – Page Rank

New Term Spam:

A SEO walks into a bar pub inn roadhouse saloon tavern alehouse beer house beer garden public house drinkery beer ale draught wine…

97

You no longer get benefit from repeating a term, but do want to spam synonyms to increase the chance of containing a term that might be searched for

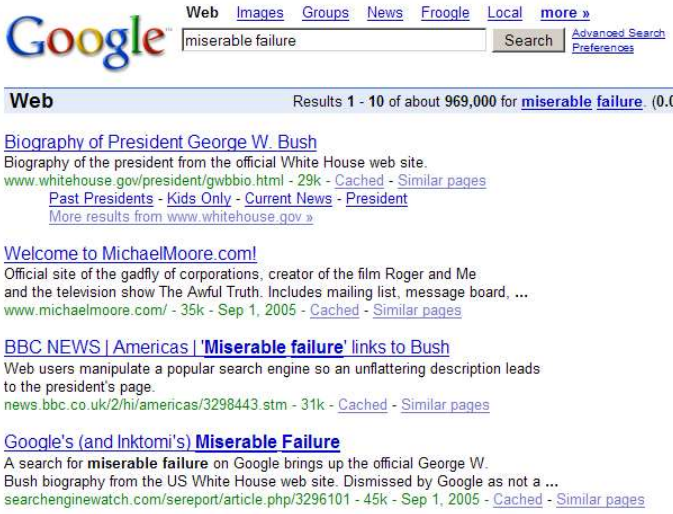
Solution?

Trust what others say about you, not what you say about yourself:

Use link text (and surrounding text) as terms, instead of contents of page

Remember “tragic love story” vs “star-crossed romance”? Solved.

It has its own problems, though



Forum Spam / Comment Spam

What if you go to every page that allows posting, and link to your webpage?

Now YouTube, Facebook, CBC News, etc. all link to you.

They have high rank => You have high rank

You also chose the link text, so you're picking your own terms again


Spam Farming
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets,
http://www.mmds.org

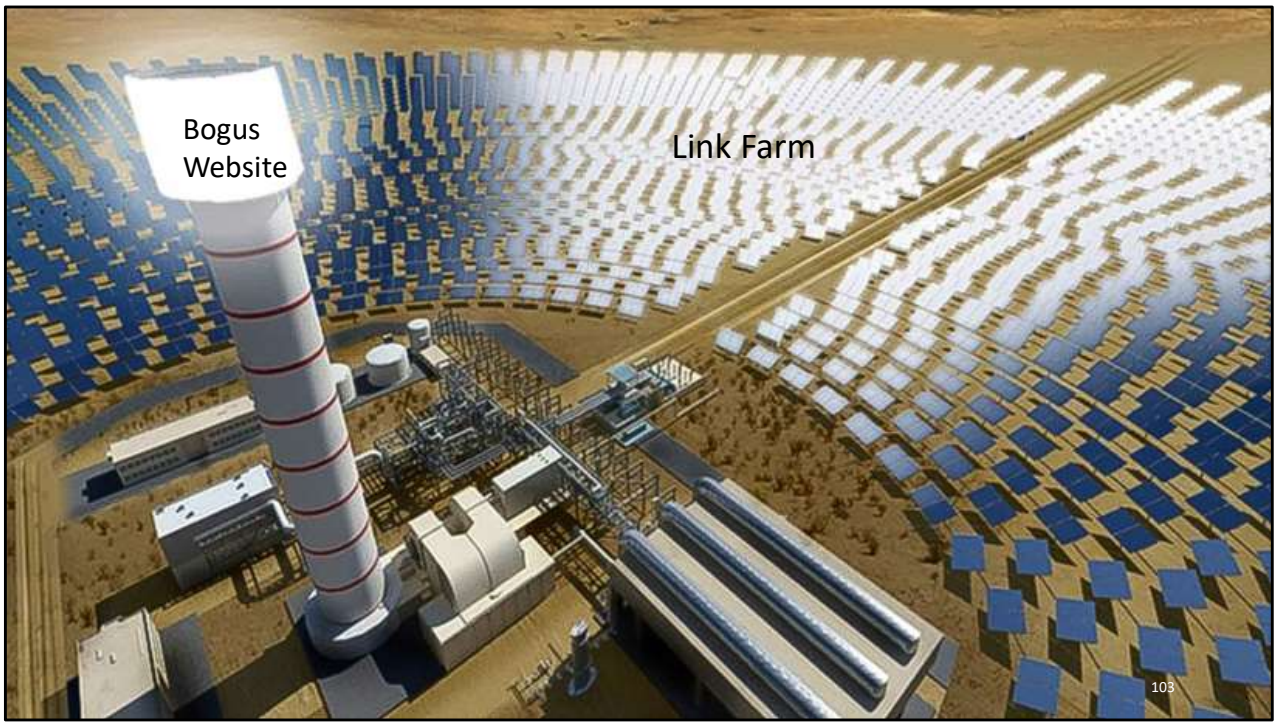
Spam Farming Techniques

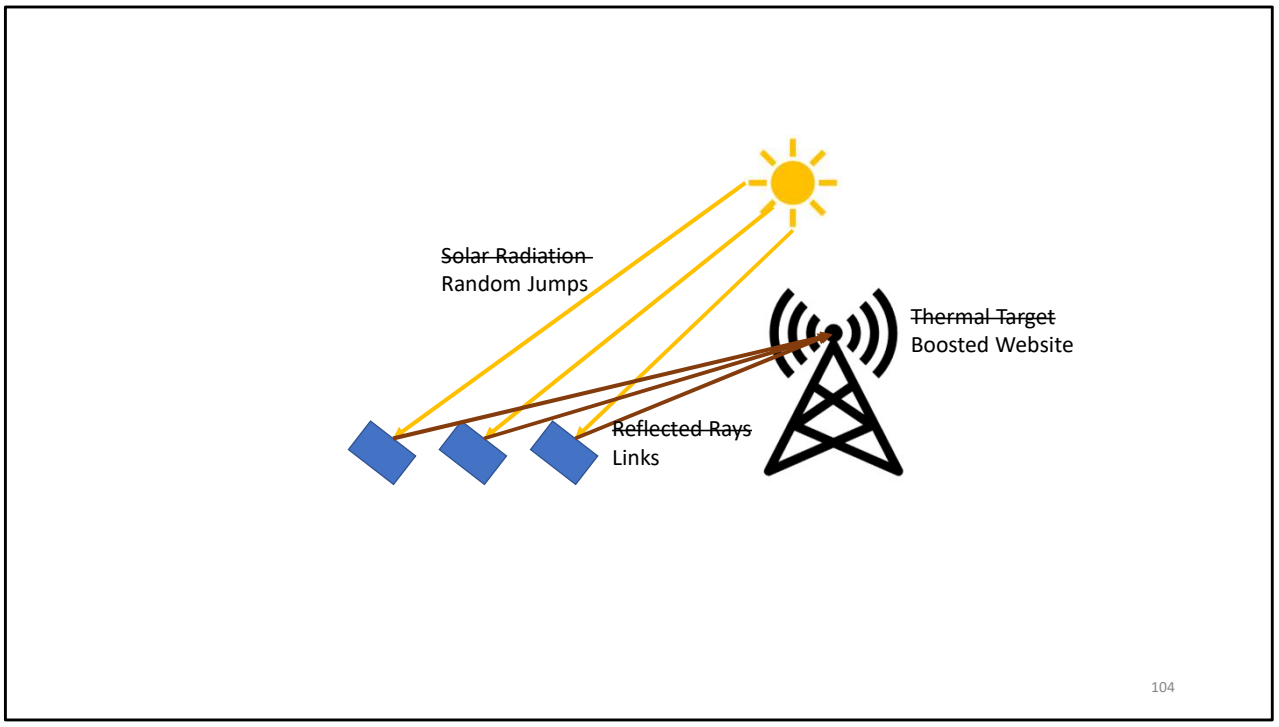
"Spider traps" accumulate rank.

- Random jumps prevent them from accumulating ALL rank, but it's still boosted by the topology

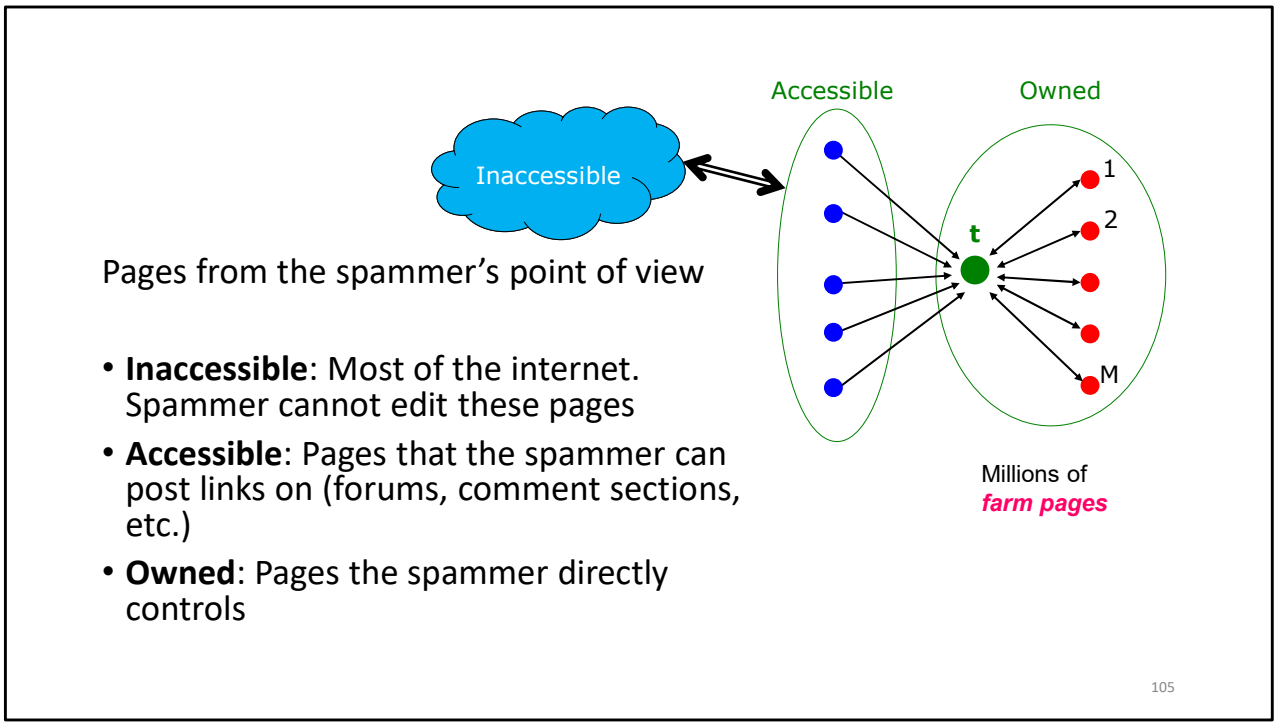
Technique:

- Page you want to promote has millions of hidden links to farm pages
  - They all accumulate the random-jump weight
- Farm pages all link back to the page you want to promote
  - They send all their rank back to the page being boosted


Bogus
Website
Link Farm
103



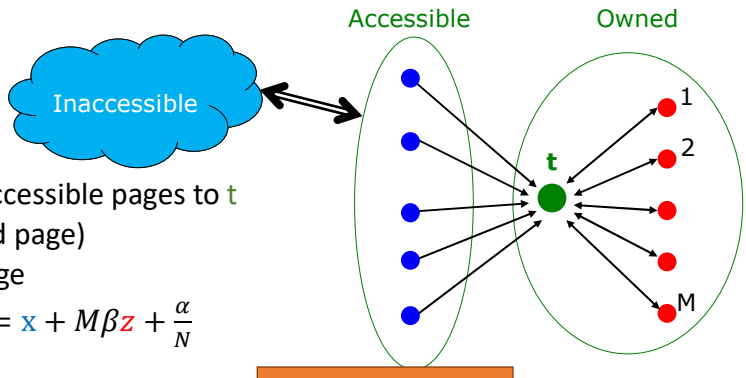
This is just a joke because it reminded me of concentrated solar thermal generators



Random jumps mean that the spider trap gets incoming rank even with no external links. The more websites you can insert links into, the better.

Also the joke solar diagram misses that the target has to link back to the millions of farm pages. That ensures that its accumulated rank stays within the trap. (Websites do of course link to external sites, but even a few dozen links will be minor when divided by millions of total links)

MATH!



x: total contribution of accessible pages to t
y: page rank of t (boosted page)
z: page rank of a farm page

$z = \frac{\beta y}{M} + \frac{\alpha}{N}$ $\quad y = x + M\beta z + \frac{\alpha}{N}$

$$y = x + M\beta\left[\frac{\beta y}{M} + \frac{\alpha}{N}\right] + \frac{\alpha}{N}$$

This term is very small, so ignored

Millions of *farm pages*

3.6x for β = 0.85
Spider trap amplifies incoming links

$$y = \frac{x}{1-\beta^2} + \frac{\beta M}{(1+\beta)N}$$

For β = 0.85, 0.45M/N
Grows linearly with M

106

Random jumps mean that the spider trap gets incoming rank even with no external links. The more websites you can insert links into, the better.

Also the joke solar diagram misses that the target has to link back to the millions of farm pages. That ensures that its accumulated rank stays within the trap. (Websites do of course link to external sites, but even a few dozen links will be minor when divided by millions of total links)

Solution to Link Spam

- Ignore links tagged as “nofollow”
- Convince forums, news sites, etc. to insert “nofollow” to all links posted in comments

Added Benefit: A researcher (university website, high rank) can link to a page (to use as an example of term spam) and not boost its ranking

Makes this 0. Solved?

$$y = \frac{x}{1-\beta^2} + \frac{\beta M}{(1+\beta)N}$$

107

Not solved! The rank is directly proportional to M. The spammer can still boost their page rank arbitrarily high by increasing M (which is cheap, just add more dummy pages to the farm and link to them from t

108

How to Solve a Problem like Spam Farms

Thoughts? What can we do to prevent this sort of trickery?

Any trick to identify the “farm pages” will lead to cat and mouse.
Don’t allow small pages to contribute? They’ll make them large enough to count.
[Also now recipe pages are forced to include a multi-paragraph story or they’ll be null rated]

Can we somehow ensure random jumps do not lead to the farm pages?

Solution to Link Farms

What's the solution?

It's the topic of the Graph assignment!

In Personalized Page Rank, spam farms don't work. Why?

109

Why: Random jumps do not lead into the farm pages, so they're not accumulating very much mass.

What to use for “Source Nodes”

We should identify “trustworthy” pages

Easy to say…

What’s trustworthy?

Domains with strict entry requirements?

.edu, .gov, etc.

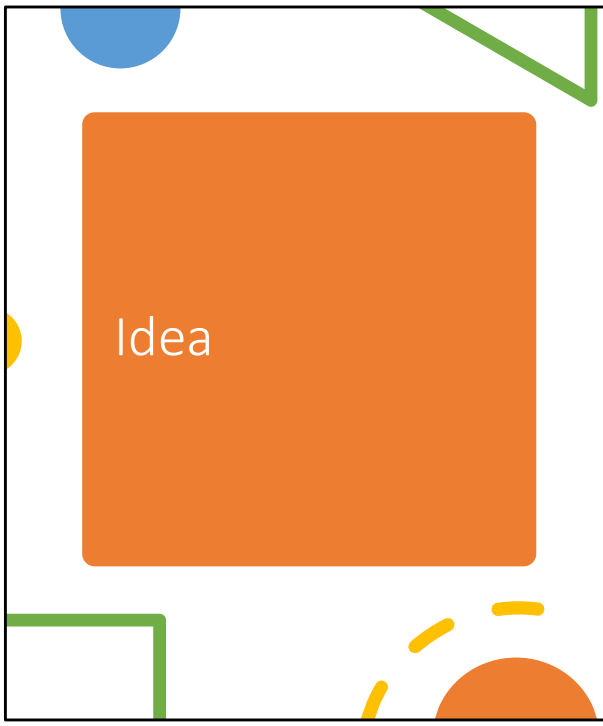
(UW doesn’t make the cut…)

Idea

Collect a sampling of webpages (seed pages)

Oracle (Human) sorts the trustworthy from the spam.

- Expensive
- Keep the set as small as possible



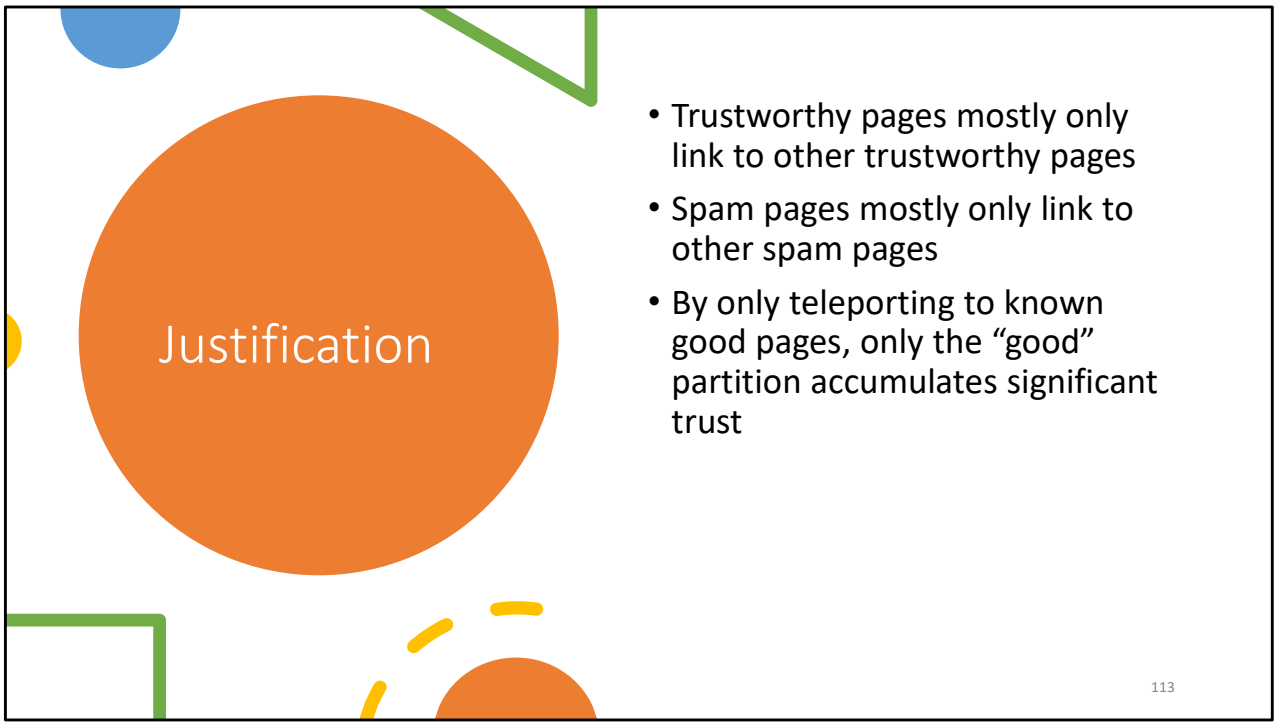
Use the “good” pages as the source nodes for personalized page rank

Small change: Each page in trust set is initialized to 1:

Trust sums to M instead of to 1

After iteration, all pages have a trust factor of between 0 and 1

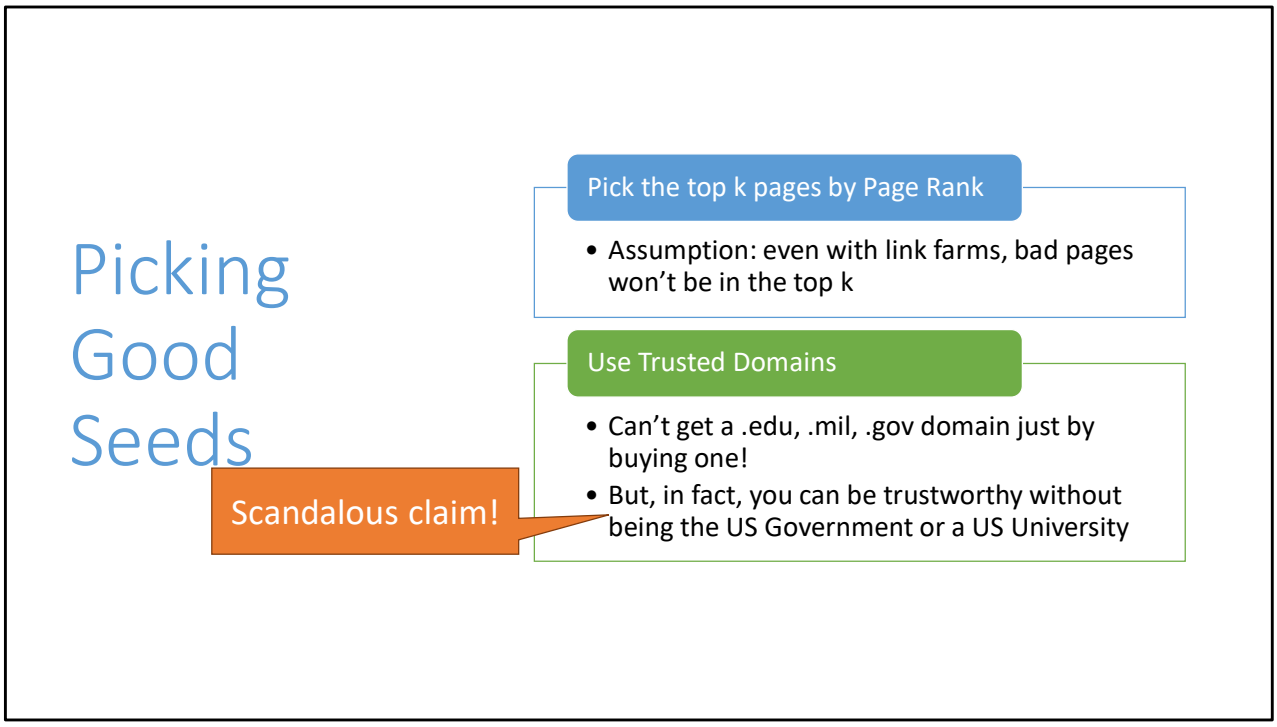
Pick a threshold and mark all pages below that as spam


Justification
• Trustworthy pages mostly only link to other trustworthy pages
• Spam pages mostly only link to other spam pages
• By only teleporting to known good pages, only the “good” partition accumulates significant trust
113

Conflicting Interests

- The more seed pages there are, the most time and effort needed to curate
- The fewer seed pages there are, the less trust there is in the system
  - Threshold will need to be lower, more spam pages slip through
- Need to pick seed pages that are highly likely to point to "most" of the other good pages
  - Your Trust is roughly proportional to your link distance from a "good" seed page.

Picking Good Seeds



So what's the solution?

Bootstrapping Trust

- If your seed set is small, will miss a lot of trustworthy sites
  - Alternate View: You will only catch a small number of spam farmers
- BUT: Anything that you find is probably pretty trustworthy
  - Candidates to be added to the trusted set.

1. Run PageRank
2. Select top pages as seed (and verify trustworthiness)
3. Run TrustRank
4. Set threshold low enough to avoid false positives
5. Remove spam pages from graph
6. Goto Step 1

Alternative – Spam Mass

r_p = PageRank of Page *p*

r_p^+ = PageRank of Page *p*, but random jumps only lead to **Trusted** pages

$r_p^- = r_p - r_p^+$ = Contribution of "low trust" pages to *p*'s rank

$S_p = \frac{r_p^-}{r_p}$ = Spam Mass (Fraction of *p*'s rank that's from "low trust" pages)

The higher your Spam Mass, the more likely you are to be spam

117

Question: Can this be exploited?
Can you target a business and make them look like spammers?

No: If you target Wikipedia with a spam farm, almost all of its rank is coming from elsewhere. You can't form the proper spider trap topology without Wikipedia pointing to all of your farm pages (and hopefully editors won't let you do that)

(The hidden reference here is that “Trojans” could be about the Trojan war, or about the sports team. Or condoms. Hard to say)

Topic-Specific Page Rank

Instead of the teleport set being all pages, it's all pages on a given topic T

Where do find this set?

DMOZ is dead, long live Curlie

- DMOZ?
- High Page-Rank pages, classified using ML (Topic Classification) ?

In other words, the same thing as Multi-Source Personalized Page Rank

Further Tweaks

The random-teleport set can be weighted without breaking anything

If a total of X mass teleports then

- Unweighted: Every Source Node gets X / M
- Weighted: Source Node S_i gets w_iX / W (Where W is the sum of all w_i)

You can use regular Page Rank as the weights.