Graph Data + MapReduce

Everything Data

CompSci 216 Spring 2015

DUKE
COMPUTER SCIENCE

Announcements (Wed. Apr. 1)

- **Homework #11** to be posted by tomorrow
- **Poll**: are enough groups ready to start mini-conference on Monday April 20?
  - Originally scheduled for Wednesday April 22 and Thursday April 30 (final slot)

"Importance" of nodes/edges

AKA *centrality*

Which web pages are the most important in a web graph?

Which friendships are the most important in a social network?

Web search

- Recall TF-IDF + cosine similarity
- Is it enough?
  - A relevant page may not contain all terms being searched
  - An irrelevant page may contain many!
    - Any measure based on content alone invites spam

*Structure of the **web graph** comes to rescue!*

- Nodes: pages
- Directed edges: links

Rank by in-degree

That is, the number of incoming links

- Think of each URL pointing to your page as a "vote" for its importance

Problem?

- Still easy to spam
  - Just create lots of pages linking to the one you want to promote!
  - Culprit: measure based on "local" link structure

PageRank

Pages pointed to by important pages should be more important

– Definition is recursive by design
– Based on *global* link structure; harder to spam

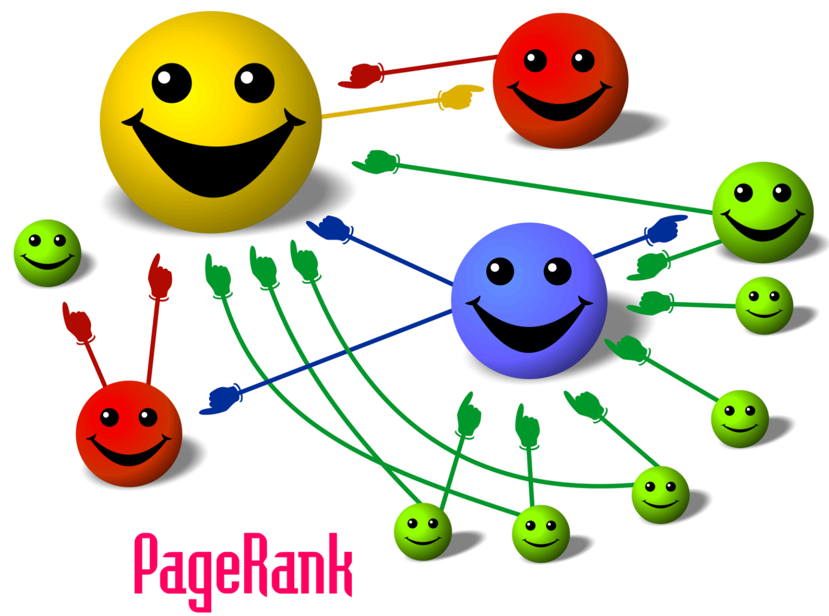
http://en.wikipedia.org/wiki/File:Google_page_brin.jpg

Naïve PageRank

- $F(p)$: set of pages that p points to
- $B(p)$: set of pages that point to p

$$PR(p) = \sum_{q \in B(p)} PR(q) \,/\, |F(q)|$$

  - Each page p gets a boost from every page q pointing to it
  - Each page q distributes its importance to pages it points to



http://en.wikipedia.org/wiki/File:PageRank-hi-res.png

Computing naïve PageRank

1. Initially, set all PageRank's to $1/N$
   - N is the total number of pages
2. For each page p, compute
   $$\sum_{q \in B(p)} \mathrm{PR}(q) \,/\, |F(q)|$$
3. Update all PageRank's
4. Go back to 2, unless values have converged

"Random surfer" model

- A random surfer
  - Starts with a random page
  - Randomly selects a link on the page to visit next
  - Never uses the "back" button

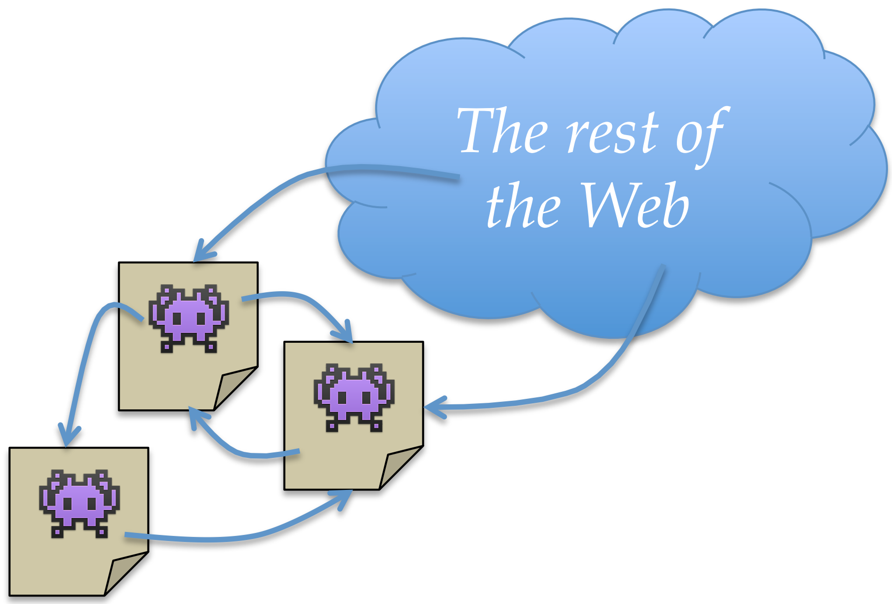
☞PageRank of *p* measures the probability that a random surfer visits page *p*

Image: http://www.zdnet.com/on-the-internet-now-everybody-knows-youre-not-a-dog-7000011439/

Problem: “dead end”

A page with no outgoing link—
all importance will eventually “leak” out

Problem: “spider trap”



A group of pages with
no links out of the group—
all importance will eventually be
“trapped” by them

Revised random surfer model

Instead of always following a link on the current page, flip a coin and “teleport” to a random page with some probability

What about dead ends?

At a dead end, what if the coin flip tells us not to teleport?

- Option 1: just teleport anyway
  - Make the dead end point to all pages
- Option 2: stay put
  - Make the dead end point to itself

Practical PageRank

$$PR(p) = (1 - d)/N + d \cdot \sum_{q \in B(p)} PR(q)/|F(q)|$$

- "Damping" factor d between 0 and 1
  - Typically between 0.8 to 0.9
  = Probability of following links
- Graph effectively becomes strongly connected—no dead ends or spider traps
- Computation is the same as naïve PageRank, except the formula for updating PageRank is revised accordingly

Personalized PageRank

Why should everybody rank pages the same way? Can we tailor PageRank toward individual preferences?

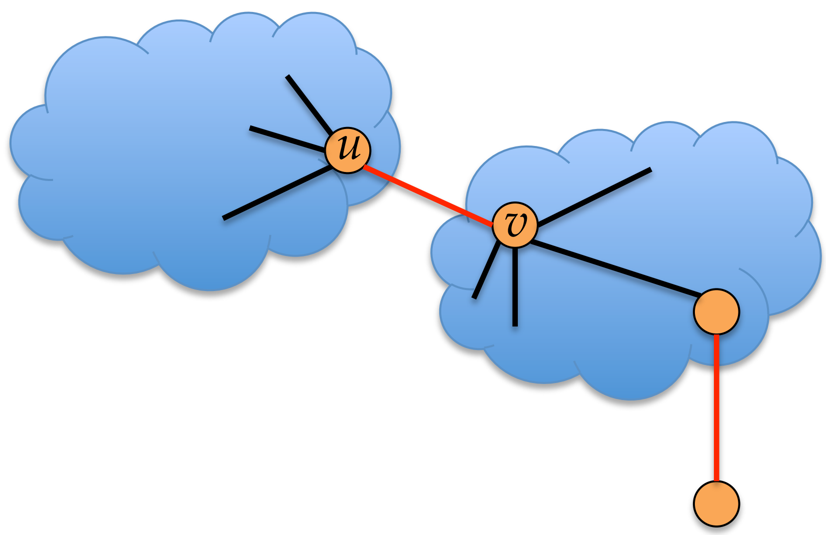
- Many methods exist, but they are all variants of the random surfer model, where the surfer teleports to different pages with different probabilities (personalized)

Most important links

http://www.bigvisionseo.com.au/wp-content/uploads/2013/05/Link-Building-Metrics.jpg

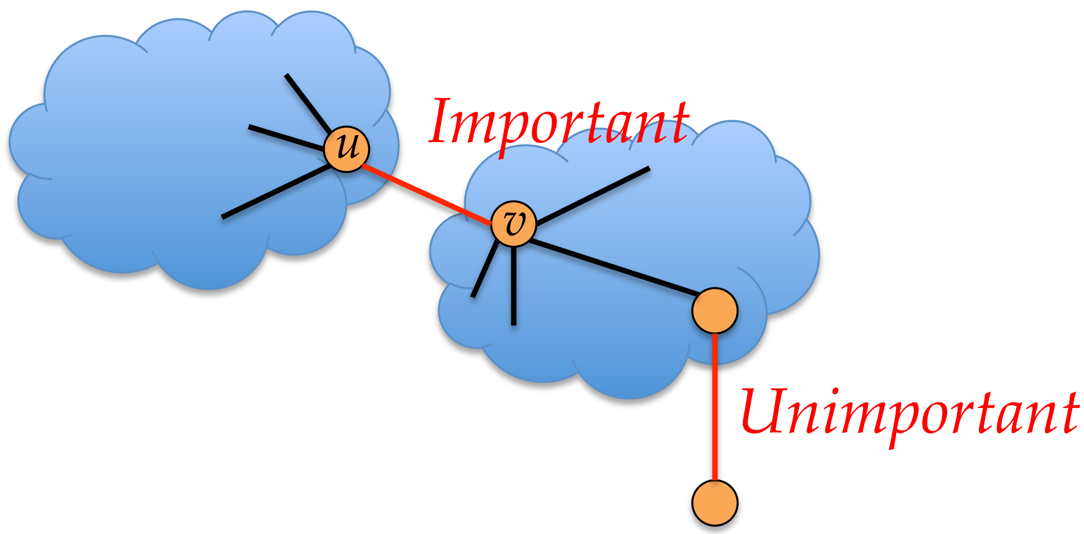
Bridge?

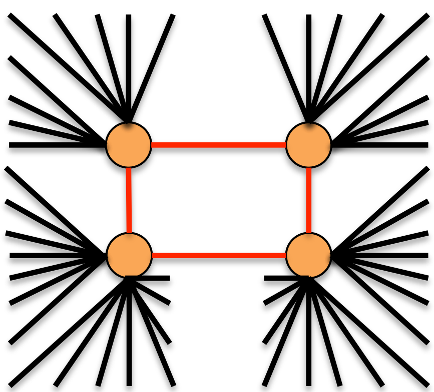
- That is, removing (u, v) will put u and v in separate connected components
  - Intuition: big impact on connectivity



But what about a bridge to a tiny island?

Edge between two hub nodes?



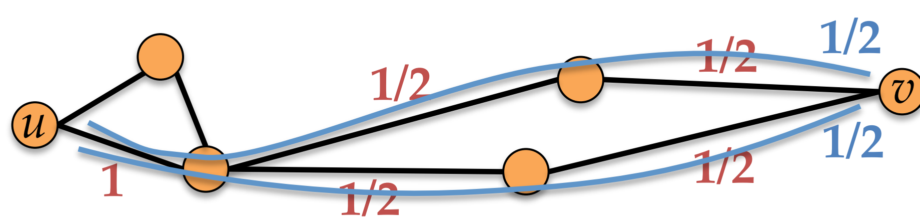


But in general:

- What qualifies as a hub?
- Still, which one do we remove?
- Will it really impact connectivity?

"Betweenness" measure

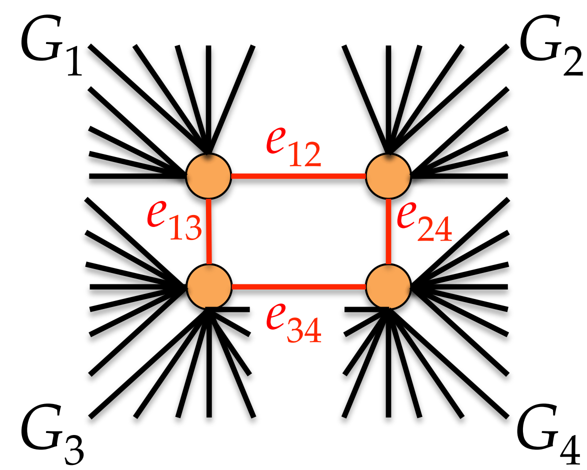
- Given nodes *u* and *v*, imagine pushing one unit of "flow" from *u* to *v*
- This flow divides itself evenly along all possible shortest paths from *u* to *v*



Only showing contribution from flow between (u, v) here

- ***Betweenness*** of an edge *e* = total amount of flow it carries (counting flows between all pairs of nodes along *e*)

Betweenness example



- Which one of these 4 edges has the highest betweenness?
  - All four edges carry flow on
    - Half of the shortest paths between G_1 and G_4 nodes
    - Half of the shortest paths between G_2 and G_3 nodes
  - In addition:
    - e_{12} carries flow on all shortest paths between G_1 and G_2 nodes
    - e_{13} carries flow on all shortest paths between G_1 and G_3 nodes
    - e_{24} carries flow on all shortest paths between G_2 and G_4 nodes
    - e_{34} carries flow on all shortest paths between G_3 and G_4 nodes
  - Suppose $|G_1|=|G_2|<|G_3|=|G_4|$; then e_{34} has the highest betweenness

Betweenness for partitioning

- Calculate betweenness for all edges
- Remove the edge with the highest betweenness
- Until the desired partitioning is obtained, repeat the above steps

Computationally expensive on big graphs; approximation or other methods often used instead

From theory to implementation

http://debane.org/franck/wp-content/uploads/2010/09/scalability.jpg

Large-scale PageRank

Compute in parallel with lots of machines, e.g., using

hadoop MapReduce

Overall approach (conceptual)

For each iteration:

- Input: ⟨*p*, (PR(*p*), *pages p points to*)⟩, …
- Map: for each page *p*, emit
  - ⟨*p*, *pages p points to*⟩
  - ⟨*q*, *contribution by p*⟩ for each *q* that *p* points to
- Reduce: for each page *p*
  - Compute PR(*p*) as the weighted sum of $1/N$ and total contributions
  - Emit ⟨*p*, (PR(*p*), *pages p points to*)⟩

The Devil is in the detail

How do we get N (total # pages)?

- A single reducer would be needed upfront just to set the initial value of $1/N$
- Trick: pretend all PR's get multiplied by N
  - $\mathrm{PRN}(p) = (1 - d) + d \cdot \sum_{q \in B(p)} \mathrm{PRN}(q)/|F(q)|$, where $\mathrm{PRN}(p) = \mathrm{PR}(p) \times N$
  - No longer probabilities, but still good for ranking
- Turns out we still need N (more on it later)
  - Just let map emit ⟨"N", 1⟩ for each page and let reduce sum them up

More details

Recall dead ends (pages with no outgoing links)

- Suppose we choose option 1—make them point to every page
  - Implementing it naïvely adds a lot of overhead
- Instead, sum up contributions from all dead ends, and apply this total to all pages
  - Each page then gets 1/*N* of the total
    - So we still need *N* here
  - Use a second MapReduce job in each iteration to apply contributions from dead ends

Even more details

How do we pair up each page with *N* and total contribution from dead ends?

- Need a "broadcast" primitive
  - Practical implementations of MapReduce often provide workarounds
    - E.g.: Hadoop has a distributed file system: broadcast = all tasks read the same file
  - In "pure" MapReduce, map can replicate input for each reduce key
    - You can invent keys to capture the desired degree of parallelism/replication

Summary

- Centrality measures
  - E.g., PageRank and betweenness
  - Others include *degree*, *closeness*, etc.
  - No one-size-fit-all; best choice depends on what you want
  - "Global" measures are more robust
- Scalability with MapReduce
  - Perhaps not the most natural/powerful model for graphs, but it works!