Computing PageRank With MapReduce

Introduction to Information Retrieval
CS 221
Donald J. Patterson

Content adapted from Michael Nielsen
http://michaelnielsen.org/blog/using-mapreduce-to-compute-pagerank/

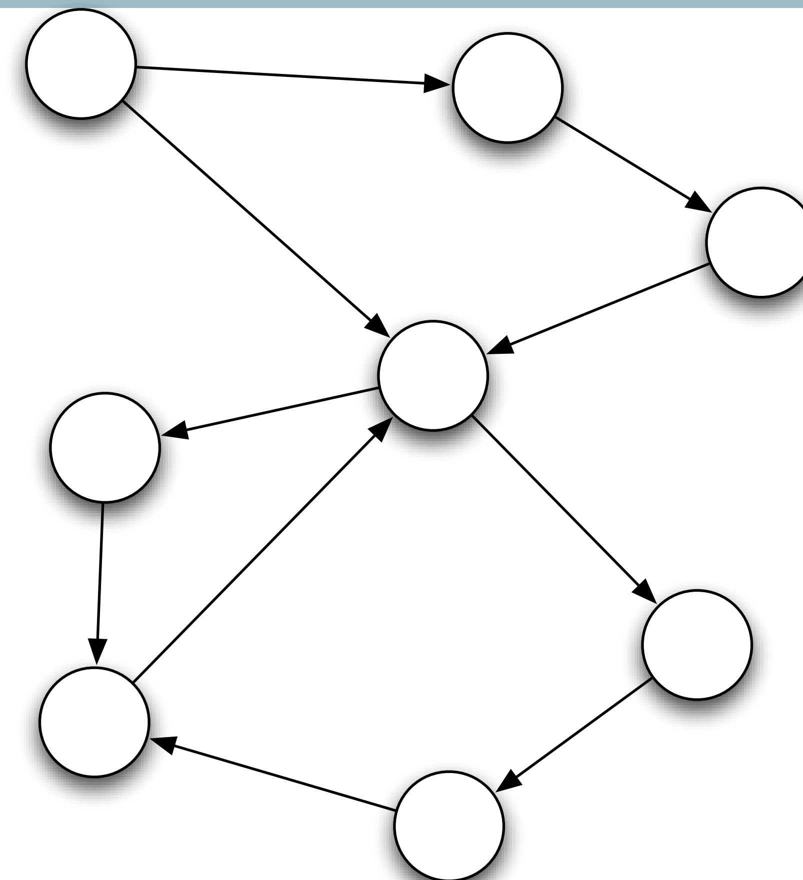
PageRank with MapReduce

- PageRank is iterative
- MapReduce is not
- This solution describes how to do one iteration of PageRank using MapReduce
- Multiple iterations would be required to converge

PageRank with MapReduce

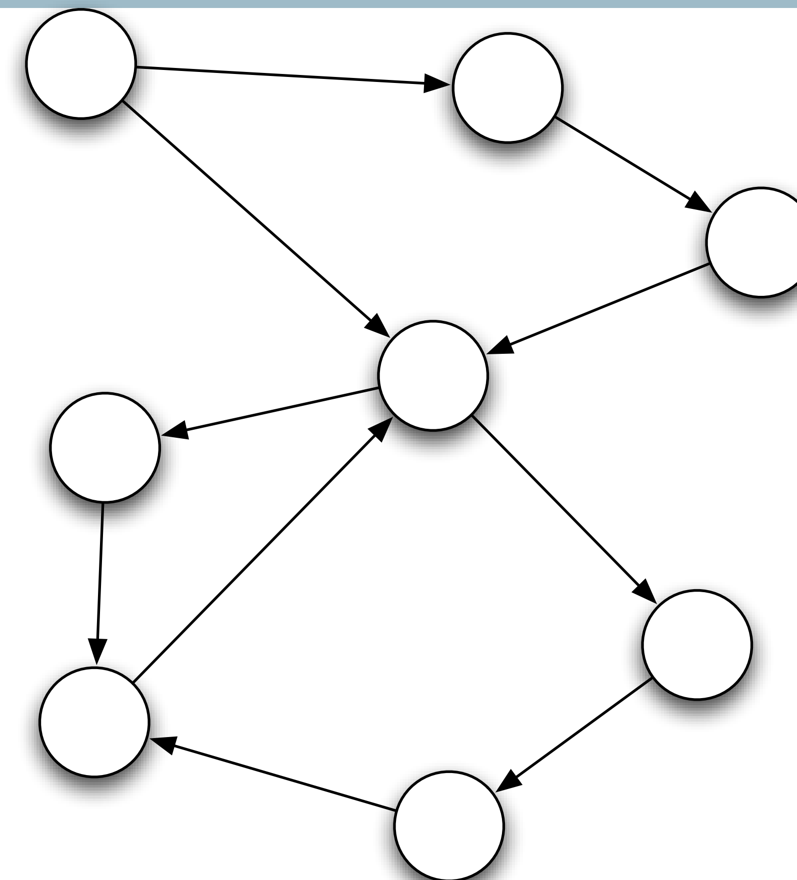
- Quick review of PageRank
  - PageRank determines which pages are well-connected
  - A connection is a social signal that a web page is important
  - A connection is a vote for importance
  - Connections take time to form
    - Not so good for real-time data
  - Mathematically this is a Markov Chain

PageRank with MapReduce



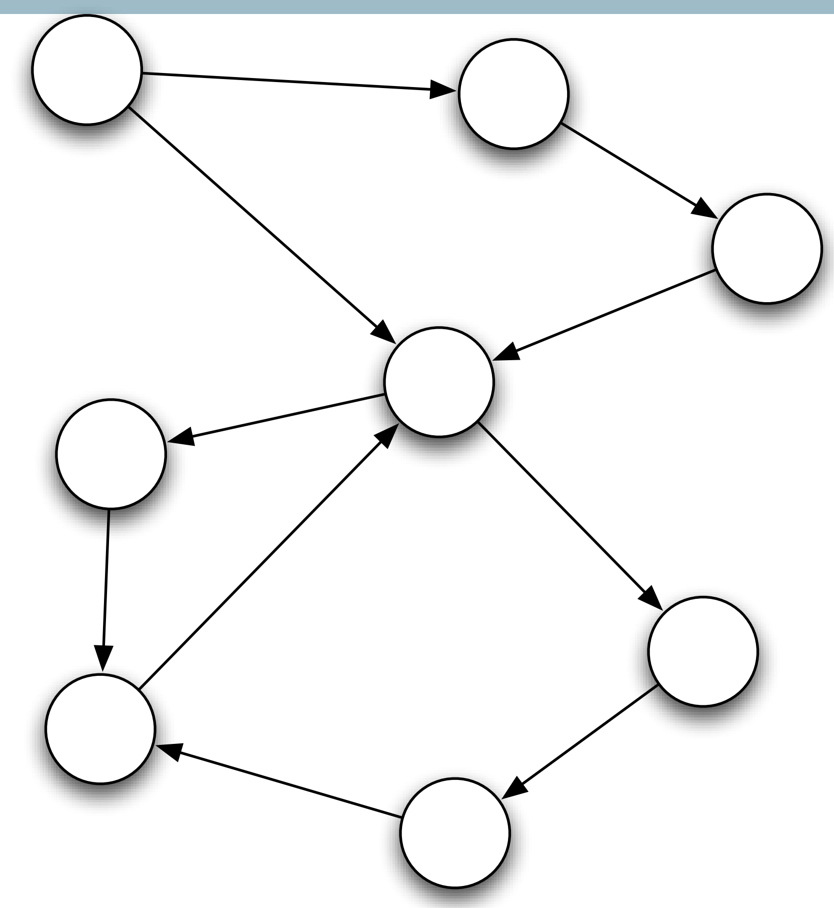
- Quick review of PageRank
  - A Markov Chain
    - Has a starting probability
    - Has a set of states
    - Has transition probabilities
  - The web forms a graph which can be treated like a Markov Chain
  - If the Markov Chain is ergodic, then PageRank converges

PageRank with MapReduce



- Quick review of PageRank
  - A Markov Chain
    - Has a starting probability P_0
    - Has a set of states N
    - Has transition probabilities A_{ij}
  - The web forms a graph which can be treated like a Markov Chain
  - If the Markov Chain is ergodic, then PageRank converges

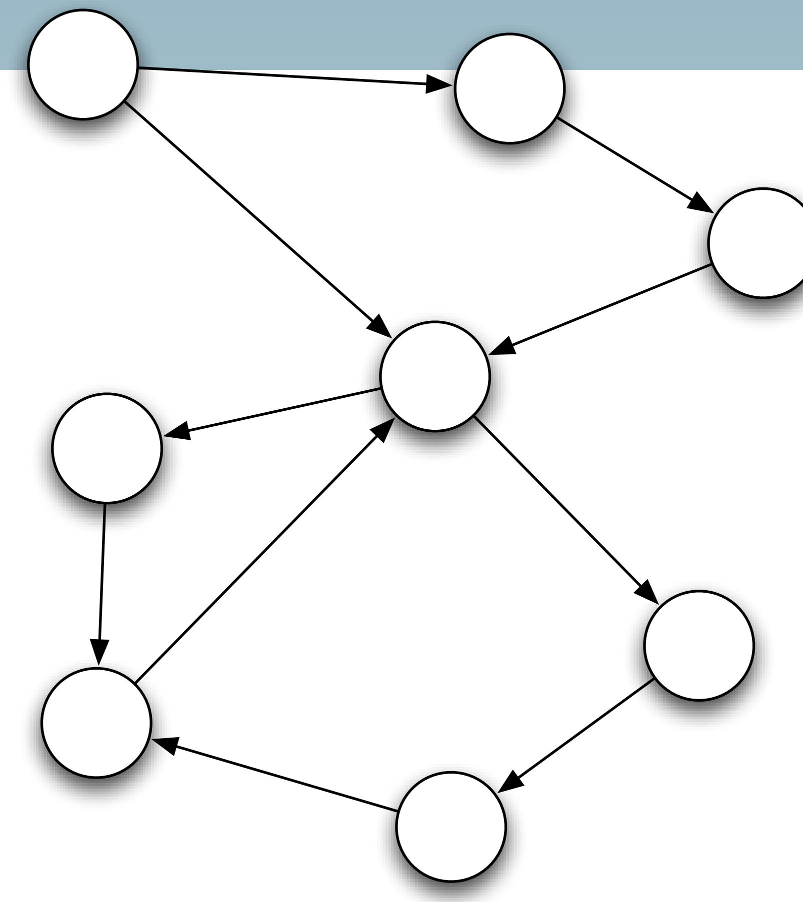
PageRank with MapReduce



$$P_1 = P_0 A$$

$$PageRank = \lim_{n \to \infty} (P_n)$$

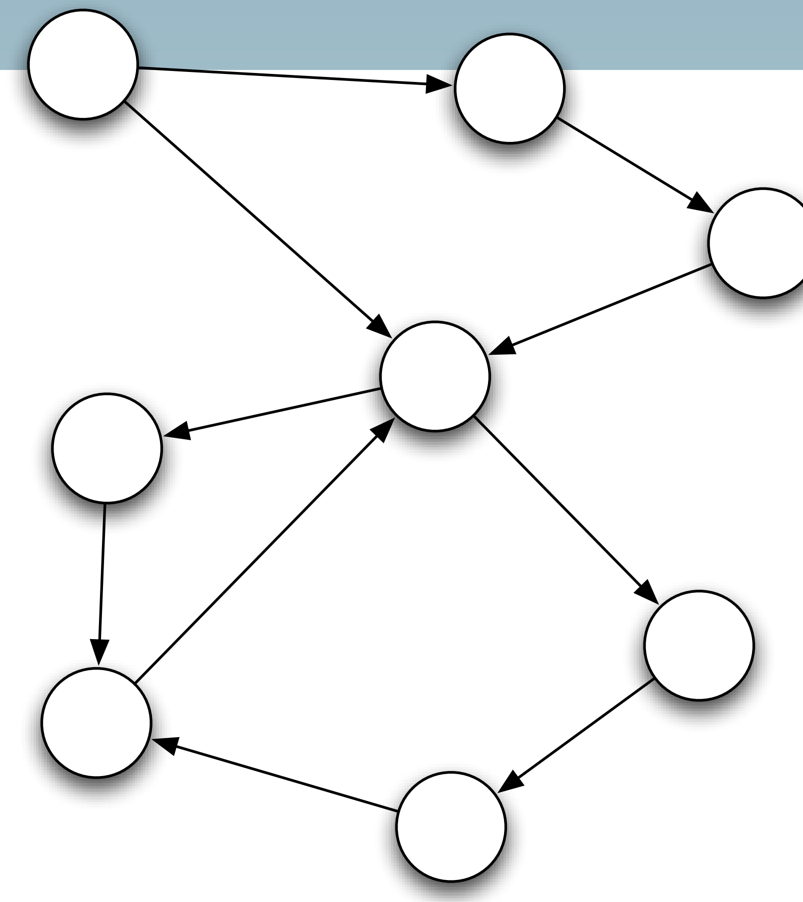
PageRank with MapReduce



- Assumptions
  - Initial probability is uniform
  - A transition is made up of
    - outlinks O
    - deadend teleports D
    - random teleports T
    - a mixing constant $0 <= \alpha <= 1$

$$A_{ij} = \alpha O + \alpha D + (1 - \alpha) T$$

PageRank with MapReduce



- Assumptions
  - Initial probability is uniform
  - A transition is made up of
    - outlinks O
    - deadend teleports D
    - random teleports T
    - a mixing constant $0 <= \alpha <= 1$

$$A_{ij} = \alpha O + \alpha D + (1 - \alpha)T$$

PageRank with MapReduce

- Map
  - Input is
    - key: page id, i
    - value: [p_i, set of outlinked pages O_i]
  - One output for every page $j \in (1..n)$
    - key: page id, j
    - value:
      - if $(O_i == \{\})$ $(\alpha f_D(i,j) + (1-\alpha) f_T(i,j)) p_i$
      - if $(j \in O_i)$ $(\alpha f_O(i,j) + (1-\alpha) f_T(i,j)) p_i$
      - if $(j \notin O_i)$ $(\alpha(0) + (1-\alpha) f_T(i,j)) p_i$

$$p_i(\alpha \frac{1}{|O_i|} + (1-\alpha)\frac{1}{n})$$

PageRank with MapReduce

- Outlink probability
  - uniform

$$f_O(i,j) = \frac{1}{|O_i|}$$

- When you hit a deadend
  - jump to a random page uniformly

$$f_D(i,j) = \frac{1}{n}$$

- When you teleport
  - teleport to a random page uniformly

$$f_T(i,j) = \frac{1}{n}$$

- More sophisticated extensions are imaginable

PageRank with MapReduce

- Reduce collects the probabilities and adds them
  - Input is
    - key: page id, i
    - value: probability of $j \rightarrow i$
  - Output is
    - key: page id, i
    - value: sum of all input probabilities

$$p_i = \sum_j p_j A_{ji}$$

PageRank with MapReduce

- Summary
  - Each step of PageRank computes one iteration of

$$P_{n+1} = P_n A$$

  - Each Map job handles the probability mass of one page being split across many pages
  - Each Reduce job collects the probabilities of one page coming from many pages