Data-Intensive Distributed Computing
CS 431/631 451/651 (Winter 2021)

Part 5: Analyzing Graphs (2/2)

Ali Abedi

Thanks to Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University)
These slides are available at https://www.student.cs.uwaterloo.ca/~cs451/
Structure of the Course

“Core” framework features and algorithm design

Analyzing Text
Analyzing Graphs
Analyzing Relational Data
Data Mining
Web as a Directed Graph

I'm a student at Univ. of X

My song lyrics

Classes

Univ. of X

I'm applying to college

USNews College Rankings

I teach at Univ. of X

USNews Featured Colleges

Networks

Networks class blog

Blog post about college rankings

Blog post about Company Z
Who to trust?

Query: University of Waterloo

uwwaterloo.ca

fakeuw.ca

University of waterloo University of waterloo University of waterloo University of waterloo University of waterloo University of waterloo University of waterloo University of waterloo University of waterloo

Ranked retrieval fails!
Web Search Challenge

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

PageRank:
The “Flow” Formulation
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

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.

$$r_j = r_j/3 + r_k/4$$
Define a “rank” $r_j$ for page $j$

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

d_i ... out-degree of node i

"Flow" equations:

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$
Solving the Flow Equations

- **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!**

Flow equations:
\[
\begin{align*}
  r_y &= r_y/2 + r_a/2 \\
  r_a &= r_y/2 + r_m \\
  r_m &= r_a/2
\end{align*}
\]

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

PageRank: How to solve?

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

- **Example:**

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

  Iteration 0, 1, 2, …
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:**

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

  Iteration 0, 1, 2, …

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>½</td>
<td>½</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>½</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>½</td>
<td>0</td>
</tr>
</tbody>
</table>

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

- **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

\[ r_j = \sum_{i \rightarrow j} \frac{r_i}{d_{out}(i)} \]
The Stationary Distribution

- Where is the surfer at time $t+1$?
  - Follows a link uniformly at random
    \[ p(t + 1) = M \cdot p(t) \]
  - Suppose the random walk reaches a state
    \[ p(t + 1) = M \cdot p(t) = p(t) \]
    then $p(t)$ is **stationary distribution** of a random walk

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 $t = 0$.
PageRank: The Google Formulation
PageRank: Three Questions

\[ r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i} \]

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

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

- Example:

\[
\begin{align*}
r_a &= 1 \ 0 \ 1 \ 0 \\
r_b &= 0 \ 1 \ 0 \ 1
\end{align*}
\]

Iteration 0, 1, 2, …
Does it converge to what we want?

\[ r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i} \]

- Example:
  
  \[ \begin{align*}
  r_a &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \\
  r_b &= \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}
  \end{align*} \]

Iteration 0, 1, 2, …
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”

- (2) Spider traps:
  (all out-links are within the group)
  - Random walker gets “stuck” in a trap
  - And eventually spider traps absorb all importance
**Problem: Spider Traps**

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

- **Example:**

  \[
  \begin{pmatrix}
  r_y \\
r_a \\
r_m
  \end{pmatrix} =
  \begin{pmatrix}
  1/3 & 1/3 & 1/3 \\
  2/6 & 1/6 & 3/6 \\
  3/12 & 2/12 & 3/12
  \end{pmatrix}
  \begin{pmatrix}
  5/24 \\
  3/24 \\
  7/12
  \end{pmatrix}
  \]

  Iteration 0, 1, 2, ...

  All the PageRank score gets "trapped" in node \( m \).

  \[
  \begin{array}{ccc}
  y & a & m \\
  \frac{1}{2} & \frac{1}{2} & 0 \\
  \frac{1}{2} & 0 & 0 \\
  0 & \frac{1}{2} & 1 \\
  \end{array}
  \]

  \( r_y = r_y/2 + r_a/2 \)

  \( r_a = r_y/2 \)

  \( r_m = r_a/2 + r_m \)

The Google solution for spider traps: At each time step, the random surfer has two options

- With prob. $\beta$, follow a link at random
- With prob. $1-\beta$, jump to some random page
- Common values for $\beta$ are in the range 0.8 to 0.9

Surfer will teleport out of spider trap within a few time steps
Problem: Dead Ends

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

- **Example:**

  \[
  \begin{pmatrix}
  r_y \\
  r_a \\
  r_m
  \end{pmatrix} =
  \begin{pmatrix}
  1/3 & 2/6 & 3/12 & 5/24 & 0 \\
  1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\
  1/3 & 1/6 & 1/12 & 2/24 & 0
  \end{pmatrix}
  \]

  Iteration 0, 1, 2, ...

  Here the PageRank “leaks” out since the matrix is not stochastic.

  \[
  \begin{array}{ccc}
  y & a & m \\
  \frac{1}{2} & \frac{1}{2} & 0 \\
  \frac{1}{2} & 0 & 0 \\
  0 & \frac{1}{2} & 0 \\
  \end{array}
  \]

  \[
  r_y = r_y/2 + r_a/2 \\
  r_a = r_y/2 \\
  r_m = r_a/2
  \]
**Solution: Always Teleport!**

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

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
Google’s solution that does it all:
At each step, random surfer has two options:
- With probability \( \beta \), follow a link at random
- With probability \( 1 - \beta \), jump to some random page

PageRank equation [Brin-Page, 98]

\[
    r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}
\]

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.

Random Teleports ($\beta = 0.8$)

\[
\begin{align*}
\begin{bmatrix}
\frac{7}{15} & \frac{7}{15} & \frac{13}{15} \\
\frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\
\frac{7}{15} & \frac{1}{15} & \frac{1}{15}
\end{bmatrix}
\end{align*}
\]

\[
\frac{1}{3} \quad 0.33 \quad 0.24 \quad 0.26 \quad \frac{7}{33}
\]

\[
a = \begin{bmatrix}
\frac{1}{3} & 0.20 & 0.20 & 0.18 & \ldots \\
\frac{1}{3} & 0.46 & 0.52 & 0.56 & \frac{21}{33}
\end{bmatrix}
\]

\[
\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}
\]

\[
\begin{bmatrix}
y & \frac{7}{15} & \frac{7}{15} & \frac{1}{15} \\
a & \frac{7}{15} & \frac{1}{15} & \frac{1}{15} \\
m & \frac{1}{15} & \frac{7}{15} & \frac{13}{15}
\end{bmatrix}
\]
PageRank MapReduce Implementation
Simplified PageRank

First, tackle the simple case:
- No random jump factor
- No dangling (dead end) nodes
PageRank in MapReduce

Map

Reduce
PageRank Pseudo-Code

class Mapper {
    def map(id: Long, n: Node) = {
        emit(id, n)
        p = n.PageRank / n.adjacenyList.length
        for (m <- n.adjacenyList) {
            emit(m, p)
        }
    }
}

class Reducer {
    def reduce(id: Long, objects: Iterable[Object]) = {
        var s = 0
        var n = null
        for (p <- objects) {
            if (isNode(p))
                n = p
            else
                s += p
        }
        n.PageRank = s
        emit(id, n)
    }
}
### PageRank vs. BFS

<table>
<thead>
<tr>
<th>PageRank</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>PR/N</td>
</tr>
<tr>
<td>Reduce</td>
<td>sum</td>
</tr>
</tbody>
</table>

A large class of graph algorithms involve:

- Local computations at each node
- Propagating results: “traversing” the graph
Complete PageRank

Two additional complexities
What is the proper treatment of dangling nodes?
How do we factor in the random jump factor?

Solution: second pass to redistribute “missing PageRank mass”
and account for random jumps

\[ r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N} \]

One final optimization: fold into a single MR job
Optimization: fold into one MapReduce job
PageRank Convergence

Alternative convergence criteria
Iterate until PageRank values don’t change
Iterate until PageRank rankings don’t change
Fixed number of iterations
Log Probs

PageRank values are really small...
Solution?

Product of probabilities = Addition of log probs

Addition of probabilities?

\[ a \oplus b = \begin{cases} 
  b + \log(1 + e^{a-b}) & a < b \\
  a + \log(1 + e^{b-a}) & a \geq b 
\end{cases} \]
Beyond PageRank

Variations of PageRank

- Weighted edges
- Personalized PageRank (A3/A4 😊)
Convergence?

Implementation Practicalities

HDFS

map

reduce

HDFS

map

HDFS

HDFS
MapReduce Sucks

Java verbosity
Hadoop task startup time
Stragglers
Needless graph shuffling
Checkpointing at each iteration

Spark to the rescue?
Let’s Spark!
MapReduce vs. Spark