PageRank and BlockRank

Presented by:
Yu-Chung Chen
2006-4-10
Content

- “The PageRank Citation Ranking: Bring Order to the Web”, Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, 1998
Outline

- What is PageRank
- Exploit web block structure, BlockRank
- Applications
What is PageRank

- Web Link Structure
  - Forward/Back links
- Authority Matter!
- Random Surfer model

Figure 1: A and B are Backlinks of C
In the Old Days

- All backlinks created equal
Link Structure

Figure 1: A and B are Backlinks of C

Figure 2: Simplified PageRank Calculation
Simple PageRank Definition

\[ R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \]

- \( Fu \): Set of links from \( u \)
- \( Bu \): Set of links to \( u \)
- \( Nu \): \(|Fu|\)
- \( c \): constant*
- \( R(u) \): Rank of \( u \)
Rank Sink

- The loop keeps accumulate rank, but never distribute any rank outside!
Escape Term

- Solution: Rank source
- \( E(u) \) is a vector over web pages (for example, uniform or favorite page) that corresponds to a source of rank
- \( E(u) \) is a user designed parameter

\[
R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} + cE(u)
\]
Random Surfer Model

- Probability distribution of a random walk on the web graphs
- $E(u)$ can be thought as the random surfer gets bored periodically and jumps to a different page and not kept in a loop forever
Markov Chain

- Discrete-time stochastic process
- Memory-less, based solely on present decision
- Random walks
  - Discrete-time stochastic process over a graph \( G=(V, E) \) with a transition probability matrix \( P \)
- Need to be aperiodic and irreducible*
  - Web graph is not strongly connected graph!
  - Add a new transition term to create a strongly connected transition graph

\[
\text{PageRank}(p) = \frac{d}{n} + (1 - d) \sum_{(q, p) \in E} \frac{\text{PageRank}(q)}{\text{outdegree}(q)}
\]
Markov Chain (cont.)

- According Markov theory, the PageRank\( (u) \) becomes the probability of being at ‘\( u \)’ page after a lot of clicks.
- \( R \) is the solution to:

\[
R = \begin{bmatrix}
\frac{(1-q)}{N} \\
\frac{(1-q)}{N} \\
\vdots \\
\frac{(1-q)}{N}
\end{bmatrix} + q
\begin{bmatrix}
\ell(p_1, p_1) & \ell(p_1, p_2) & \cdots & \ell(p_1, p_N) \\
\ell(p_2, p_1) & \cdots \\
\vdots \\
\ell(p_N, p_1) & \cdots & \ell(p_i, p_j) & \ell(p_N, p_N)
\end{bmatrix} R
\]

- Empirical results implies \( q = 0.85 \).
Matrix Notation

\[ R = c(A^T + E \times e^T)R \]

- Write to matrix form: \( R = cA^T R + cE \)
- \( R \) is the dominant eigenvector and \( c \) is the dominant eigenvalue of \((A + E \times e^T)\) because \( c \) is maximized
- Broken down to Eigenvalue problem, can be solved efficiently
  - Characteristic polynomial: not scalable
  - Power iterative method
Compute PageRank

\[ R_0 \leftarrow S \]

loop :

\[ R_{i+1} \leftarrow AR_i \]

\[ d \leftarrow \|R_i\|_1 - \|R_{i+1}\|_1 \]

\[ R_{i+1} \leftarrow R_{i+1} + dE \]

\[ \delta \leftarrow \|R_{i+1} - R_i\|_1 \]

while \( \delta > \epsilon \)
Implementation

- 24 million pages
- 75 million URLs
- Memory and disk storage
  - Mem: weight vector: 4 bytes float
  - Disk: Matrix A: linear disk access
- $75000000 \times 4 / 1000000 = 300$MB/75million URLs
  - Fit into memory or multiple passes
- 6 minutes/iteration per machine
Back to 1998...

- In 1998, it took 5 days to index on a 24 million page database.
Now...

- Today: Google cluster and Google File system
  - 719 racks, 63,272 machines, 126,544 CPUs
  - 126,544 Gb RAM, 5,062Tb of disk space
  - [http://www.tnl.net/blog/entry/How_many_Google_machines](http://www.tnl.net/blog/entry/How_many_Google_machines)
Convergence

- O(log|V|) due to rapidly mixing web graph G
- Good initial ranking -> quick convergence*
Personalized PageRank

- Rank source $E$ can be initialized:
  - Uniformly
    - All pages are treated the same, not good
      - Copyright, mailing list archives
  - Total weigh on a single page
    - Bad too
  - Everything in-between
    - About News, sports, etc
Issues - Quality

- Users are no random walkers
- Reinforcing effects/bias towards main pages/sites
- Linkage spam
- Manipulation by commercial interests
  - Cost to buy 1 link from an important page or a link from many non-important pages
  - Hilltop, only trust experts
Issues - Speed

- Argue: Time is insignificant compared to building full text index, but...
- Re-compute ranks every few months
  - Web changes faster!
- WWW conference 2003: Google becoming up to 5 times faster
  - BlockRank: 3X the current calculation speed!
  - Extrapolation
  - Adaptive PageRank
BlockRank

- Observations: web link graph is nested block structure
  - Pages under the same domain/host link to pages under the same domain/host
  - Internal links: 80% of all links per page
- Exploit this structure to speedup PageRank computation
- 3-stage algorithm
Block Structure

(a) IBM

(b) Stanford/Berkeley

(c) Stanford-50

(d) Stanford/Berkeley Host Graph
Experiment Setup & Observations

Table 2: Hyperlink statistics on LARGEWEB for the full graph (Full: 291M nodes, 1.137B links) and for the graph with dangling nodes removed (DNR: 64.7M nodes, 607M links).
3 Stage Algorithm

1. Local PageRanks of pages for each host are computed independently.
2. Calculate BlockRanks of hosts in Block Graph.
3. Local PageRanks are weighted by the ‘importance’ of the corresponding host.
0. Sort the web graph lexicographically as described in Section 3, exposing the nested block structure of the web.
1. Compute the local PageRank vector $\vec{l}_J$ for each block $J$.
   
   \[
   \text{foreach block } J \text{ do}
   \]
   
   \[
   \vec{l}_J = \text{pageRank}(G, \vec{z}_J, \vec{v}_J);
   \]
   
   \text{end}
2. Compute block transition matrix $B$ and Block-Ranks $\vec{b}$.
   
   \[
   B = L^T AS
   \]
   
   \[
   \vec{b} = \text{pageRank}(B, \vec{v}_k, \vec{v}_k)
   \]
3. Find an approximation $\vec{x}^{(0)}$ to the global PageRank vector $\vec{x}$ by weighting the local PageRanks of pages in block $J$ by the BlockRank of $J$.
   
   \[
   \vec{x}^{(0)} = LB
   \]
4. Use this approximation as a start vector for a standard PageRank iteration.
   
   \[
   \vec{x}^{(0)} = \text{pageRank}(G, \vec{x}^{(0)}, \vec{v})
   \]

Algorithm 3: BlockRank Algorithm
BlockRank Advantages

- Speedup due to caching effects*
  - Now CPU cache and Memory
- Converge quickly
- 1st step can be done completely parallel or distributed fashion
- Results of 1st step can be reused
Experiment Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Wallclock time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>180m 36s</td>
</tr>
<tr>
<td>Standard (using url-sorted links)</td>
<td>87m 44s</td>
</tr>
<tr>
<td>BlockRank (no pipelining)</td>
<td>81m 19s</td>
</tr>
<tr>
<td>BlockRank (w/ pipelining)</td>
<td>57m 06s</td>
</tr>
</tbody>
</table>

Table 6: Wallclock running times for 4 algorithms for computing PageRank with $c = 0.85$ to a residual of less than $10^{-3}$.

<table>
<thead>
<tr>
<th></th>
<th>PageRank</th>
<th>BlockRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANFORD/BERKELEY</td>
<td>50</td>
<td>27</td>
</tr>
<tr>
<td>LARGEWEB</td>
<td>28</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 7: Number of iterations needed to converge for standard PageRank and for BlockRank (to a tolerance of $10^{-4}$ for STANFORD/BERKELEY, and $10^{-3}$ for LARGEWEB).
Experiment Results (cont.)

Figure 5: Convergence rates for standard PageRank (solid line) vs. BlockRank (dotted line). The $x$-axis is the number of iterations, and the $y$-axis is the log of the $L_1$-residual. STANFORD/BERKELEY data set; $e = 0.85$. 
Applications - PageRank

- Estimate web traffic
- Backlink predictor
- Better search engine quality
- Check out Google.com!
Applications - BlockRank
PageRank/BlockRank Highlights

- PageRank is a global ranking based on the web’s graph structure
- PageRank uses backlink information
- PageRank can be thought as random surfer model
- BlockRank: exploit block structure to speedup and advantages
- Various applications
Thank you for your attention!

Questions?
More Implementations

- Unique integer ID for each URL
- Sort and Remove dangling Links
- Iterating until converge
- Add back dangling links and re-compute
Convergence

- $G(V,E)$ is an expander with factor $\alpha$ if for all subsets $S$: $|A_S| \geq \alpha|S|$
- Eigenvalue separation: largest eigenvalue is sufficiently larger than the second-largest eigenvalue
- Random walk converges fast to a limiting probability distribution on a set of nodes in the graph
Google File System

- Performance, scalability, reliability and availability
- It’s normal to have hardware component failures
- Huge number of huge files
- Mutations
- Constraint specific file system
Google File System (cont.)

- Master: Handle meta-data
- Chunk server: Hold chunked data
  - 64MB per chunk
- Clients: Access to tera-bytes of data
Google File System (cont.)

- Reduce master workload
  - Reduce interaction with master
- Keep metadata in memory
- Availability!
- Replication!
  - Multiple replicated data chunks
  - Master state replication, and shadow master
- Fast recovery
References

- http://www-db.stanford.edu/~backrub/google.html
- http://www.tnl.net/blog/entry/How_many_Google_machines
- http://en.wikipedia.org/wiki/Markov_chain