More About PageRank

Combatting Web Spam
Dealing with Non-Main-Memory Web Graphs
SimRank

Jeffrey D. Ullman
Stanford University
Web Spam

Term Spamming
Link Spamming
What Is Web Spam?

- **Spamming** = any deliberate action intended solely to boost a Web page’s position in search-engine results.
- **Web Spam** = Web pages that are the result of spamming.
- SEO industry might disagree!
  - **SEO** = search engine optimization
Web Spam Taxonomy

- **Boosting** techniques.
  - Techniques for making a Web page appear to be a good response to a search query.

- **Hiding** techniques.
  - Techniques to hide the use of boosting from humans and Web crawlers.
Boosting

- **Term spamming.**
  - Manipulating the text of web pages in order to appear relevant to queries.

- **Link spamming.**
  - Creating link structures that boost PageRank.
Repetition of terms, e.g., “Viagra,” in order to subvert TF.IDF-based rankings.

Dumping = adding large numbers of words to your page.

- **Example**: run the search query you would like your page to match, and add copies of the top 10 pages.
- **Example**: add a dictionary, so you match every search query.

**Key hiding technique**: words are hidden by giving them the same color as the background.
Link Spam

Design of a Spam Farm
TrustRank
Spam Mass
Link Spam

- PageRank prevents spammers from using term spam to fool a search engine.
  - While spammers can still use the techniques, they cannot get a high-enough PageRank to be in the top 10.
- Spammers now attempt to fool PageRank with *link spam* by creating structures on the Web, called *spam farms*, that increase the PageRank of undeserving pages.
Three kinds of Web pages from a spammer’s point of view:

1. *Own pages.*
   - Completely controlled by spammer.
2. *Accessible pages.*
   - E.g., Web-log comment pages: spammer can post links to his pages.
     - “I totally agree with you. Here’s what I wrote about the subject at www.MySpamPage.com.”
3. *Inaccessible pages.*
   - Everything else.
Spammer’s goal:

- Maximize the PageRank of target page $t$.

Technique:

1. Get as many links as possible from accessible pages to target page $t$.
   - **Note**: if there are none at all, then search engines will not even be aware of the existence of page $t$.

2. Construct a spam farm to get a PageRank-multiplier effect.
**Goal**: boost PageRank of page $t$.
Here is one of the most common and effective organizations for a spam farm.

Note links are 2-way. Page $t$ links to all $M$ pages and they link back.
Suppose rank from accessible pages = \( x \) (known). PageRank of target page = \( y \) (unknown). Taxation rate = \( 1 - \beta \).

Rank of each “farm” page = \( \frac{\beta y}{M} + \frac{(1-\beta)}{N} \).

\( \text{From } t; \ M = \text{number of farm pages} \)

\( \text{Share of “tax”; } \ N = \text{size of the Web. Total PageRank } = 1. \)
Analysis – (2)

\[ y = x + \beta M \left[ \frac{\beta y}{M} + \frac{(1-\beta)}{N} \right] + \frac{(1-\beta)}{N} \]
\[ y = x + \beta^2 y + \beta (1-\beta) M/N \]
\[ y = x/(1-\beta^2) + cM/N \text{ where } c = \frac{\beta}{(1+\beta)} \]

PageRank of each “farm” page

Tax share for \( t \). Very small; ignore.
\[ y = \frac{x}{1-\beta^2} + \frac{cM}{N} \] where \( c = \frac{\beta}{1+\beta} \).

For \( \beta = 0.85 \), \( 1/(1-\beta^2) = 3.6 \).

- Multiplier effect for “acquired” page rank.
- By making \( M \) large, we can make \( y \) almost as large as we want.

Question for Thought:
What if \( \beta = 1 \) (i.e., no tax)?
If you design your spam farm just as was described, Google will notice it and drop it from the Web.

More complex designs might be undetected, although SEO innovations are tracked by Google et al.

Fortunately, there are other techniques for combatting spam that do not rely on direct detection of spam farms.
Detecting Link Spam

- Topic-specific PageRank, with a set of “trusted” pages as the teleport set is called TrustRank.
- \( \text{Spam Mass} = \frac{\text{PageRank} - \text{TrustRank}}{\text{PageRank}} \).
  - High spam mass means most of your PageRank comes from untrusted sources – you may be link-spam.
Two conflicting considerations:

- Human may have to inspect each trusted page, so this set should be as small as possible.
- Must ensure every “good page” gets adequate TrustRank, so all good pages should be reachable from the trusted set by short paths.
  - Implies that the trusted set must be geographically diverse, hence large.
Approaches to Picking the Trusted Set

1. Pick the top $k$ pages by PageRank.
   - It is almost impossible to get a spam page to the very top of the PageRank order.

2. Pick the home pages of universities.
   - Domains like .edu are controlled.
   - Notice that both these approaches avoid the requirement for human intervention.
Efficiency Considerations for PageRank

Multiplication of Huge Vector and Matrix
Representing Blocks of a Stochastic Matrix
The Problem

- Google computes the PageRank of a trillion pages (**at least**!).
- The PageRank vector of double-precision reals requires 8 terabytes.
  - And another 8 terabytes for the next estimate of PageRank.
The matrix of the Web has two special properties:

1. It is very sparse: the average Web page has about 10 out-links.
2. Each column has a single value – 1 divided by the number of out-links – that appears wherever that column is not 0.
Trick: for each column, store \( n \) = the number of out-links and a list of the rows with nonzero values (which must be \( 1/n \)).

Thus, the matrix of the Web requires at least

\[
(4 \times 1 + 8 \times 10) \times 10^{12} = 84 \text{ terabytes}.
\]

Integer \( n \)

Average 10 links/column,
8 bytes per row number.
The Solution: Striping

- Divide the current and next PageRank vectors into \( k \) *stripes* of equal size.
  - Each stripe is the components in some consecutive rows.
- Divide the matrix into squares whose sides are the same length as one of the stripes.
- Pick \( k \) large enough to fit a stripe of each vector and in main memory at the same time.
  - *Note*: We also need a block of the matrix, but that can be piped through main memory and won’t use that much memory at any time.
At one time, we need $w_i$, $v_j$, and (part of) $M_{ij}$ in memory.

Vary $v$ slowest: $w_1 = M_{11} v_1; w_2 = M_{21} v_1; w_3 = M_{31} v_1; w_1 += M_{12} v_2;$ $w_2 += M_{22} v_2; w_3 += M_{32} v_2; w_1 += M_{13} v_3; w_2 += M_{23} v_3; w_3 += M_{33} v_3$
Each column of a block is represented by:

1. The number \( n \) of nonzero elements in the entire column of the matrix (i.e., the total number of out-links for the corresponding Web page).

2. The list of rows of that block only that have nonzero values (which must be \( 1/n \)).

I.e., for each column, we store \( n \) with each of the \( k \) blocks and each out-link with whatever block has the row to which the link goes.
Total space to represent the matrix = \[(4 \times k + 8 \times 10) \times 10^{12} = 4k+80\] terabytes.

Integer \( n \) for a column is represented in each of \( k \) blocks.

Possible savings: if a block has all 0’s in a column, then \( n \) is not needed.

Average 10 links/column, 8 bytes per row number, spread over \( k \) blocks.
**Needed Modifications**

- We are not just multiplying a matrix and a vector.
- We need to multiply the result by a constant to reflect the “taxation.”
- We need to add a constant to each component of the result $w$.
- Neither of these changes are hard to do.
  - After computing each component $w_i$ of $w$, multiply by $\beta$ and then add $(1-\beta)/N$. 
The strategy described can be executed on a single machine.

But who would want to?

There is a simple MapReduce algorithm to perform matrix-vector multiplication.

- But since the matrix is sparse, better to treat it as a relational join.
Another approach is to use many jobs, each to multiply a row of matrix blocks by the entire $\mathbf{v}$.

Use main memory to hold the one stripe of $\mathbf{w}$ that will be produced.

Read one stripe of $\mathbf{v}$ into main memory at a time.

Read the block of $\mathbf{M}$ that needs to multiply the current stripe of $\mathbf{v}$, a tiny bit at a time.

Works as long as $k$ is large enough that stripes (but not blocks) fit in memory.

$\mathbf{M}$ read once; $\mathbf{v}$ read $k$ times, among all the jobs.

- OK, because $\mathbf{M}$ is much larger than $\mathbf{v}$. 
Animation: First Stripe

Main Memory for job i
Animation: Second Stripe

Main Memory for job $i$
Animation: j-th Stripe

Main Memory for job i
SimRank

Graphs of Entities and Connections
Finding Similar Entities by Random Walks
Unlike the similarity based on a distance measure that we discussed with regard to LSH, we may wish to look for entities that play similar roles in a complex network.

**Example:** Nodes represent students and classes; find students with similar interests, classes on similar subjects.
Example: Network

- Gus
- Ann
- Sue
- Joe

Courses:
- CS246
- CS229
- Ma55
**Approach: Pair Graphs**

- **Intuition:**
  1. An entity is similar to itself.
  2. If two entities A and B are similar, then that is some evidence that entities C and D connected to A and B, respectively, are similar.
Example: Pair Graph

Gus
CS246
Ann
CS229
Sue
Ma55
Joe

Three others

CS246, Ma55
Gus, Ann
Gus, Sue
Gus, Joe
Using Pair Graphs

- You can run Topic-Sensitive PageRank on such a graph, with the nodes representing single entities as the teleport set.
- Resulting PageRank of a node measures how similar the two entities are.
- A high tax rate may be appropriate, or else you conclude things like CS246 is similar to Hist101.
- **Problem**: Using node pairs squares the number of nodes.
  - Can be too large, even for university-sized data.
Another approach is to work from the original network.
- Treat undirected edges as arcs or links in both directions.
- Find the entities similar to a single entity, which becomes the sole member of the teleport set.

Example: “Who is similar to Sue?” on next slides.
Example: SimRank

1.000

Gus

Ann

Sue

Joe

CS246

CS229

Ma55
Example: SimRank

- Gus
- CS246
- Ann
- CS229
- Sue
- Ma55
- Joe

Weights:
- Gus to CS246: 0.400
- Ann to CS229: 0.400
- Sue to Ma55: 0.400
- Joe to Ma55: 0.200
Example: SimRank

- Gus
  - CS246
- Ann
  - CS229
  - Ma55
- Sue
- Joe

Weights:
- Gus to CS246: 0.107
- Ann to CS229: 0.080
- Sue to Ma55: 0.080
- Joe to Ma55: 0.267
Example: SimRank

- Gus
- CS246: 0.048
- Ann
- CS229: 0.336
- Sue
- CS229: 0.336
- Joe
- Ma55: 0.294
- CS246: 0.048
Example: SimRank

0.019
Gus

0.008
CS246

0.109
Ann

0.131
CS229

0.407
Sue

0.207
Joe

0.112
Ma55