Web Search & Analytics

CPS 296.3: Information Management and Mining

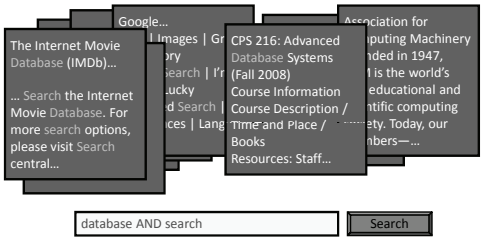
Jun Yang
Duke University
January 27, 2008

† Thanks to contents borrowed from
Rajaraman & Ullman (http://infolab.stanford.edu/~ullman/mining/mining.html)

Announcements

- Talk by Irfan Essa from Georgia Tech: ***Computation and Journalism: The Impact of Technology on Journalism, Information Quality, and Civic Literacy***
  - Right after this lecture; 4:30-6pm
  - Sanford Institute Room 03
  - References
- ➲I strongly encourage you to go—excellent source of project/research ideas

Web search



What are the documents containing both "database" and "search"?

Keywords × documents

All keywords	Document 1	Document 2	Document 3	…	Document *n*
"a"	1	1	1	…	1
"cat"	1	1	0	…	0
"database"	0	0	1	…	0
"dog"	0	1	0	…	1
"search"	0	0	1	…	0
…	…	…	…	…	…

All documents

1 means keyword appears in the document; 0 means otherwise

- Inverted lists: store the matrix by rows
- Signature files: store the matrix by columns

Inverted lists

- Store the matrix by rows
- For each keyword, store an inverted list
  - ⟨*keyword*, *doc-id-list*⟩
  - ⟨"database", {3, 7, 142, 857, …}⟩
  - ⟨"search", {3, 9, 192, 512, …}⟩
  - It helps to sort *doc-id-list* (why?)
- Vocabulary index on keywords
  - Tree- or hash-based
- How large is an inverted list index?

Using inverted lists

- Documents containing "database"
  - Use the vocabulary index to find the inverted list for "database"
  - Return documents in the inverted list
- Documents containing "database" AND "search"
  - Return documents in the intersection of the two inverted lists
- OR? NOT?
  - Union and difference, respectively

What are "all" the keywords?

- All sequences of letters (up to a given length)?
  - … that actually appear in documents!
- All words in English?
- Plus all phrases?
  - Alternative: approximate phrase search by proximity
- Minus all stop words
  - They appear in nearly every document, e.g., a, of, the, it
  - Not useful in search
- Combine words with common stems
  - Example: database, databases
  - They can be treated as the same for the purpose of search

Frequency and proximity

- Frequency
  - ⟨*keyword*, { ⟨*doc-id*, *number-of-occurrences*⟩, ⟨*doc-id*, *number-of-occurrences*⟩, … }⟩
- Proximity (and frequency)
  - ⟨*keyword*, { ⟨*doc-id*, ⟨*position-of-occurrence*$_1$, *position-of-occurrence*$_2$, …⟩, ⟨*doc-id*, ⟨*position-of-occurrence*$_1$, …⟩⟩, … }⟩
  - When doing AND, check for positions that are near

Signature files

- Store the matrix by columns and compress them
- For each document, store a w-bit signature
- Each word is hashed into a w-bit value, with only $s < w$ bits turned on
- Signature is computed by taking the bit-wise OR of the hash values of all words on the document

hash("database") = 0110
hash("dog") = 1100
hash("cat") = 0010

doc$_1$ contains "database": 0110
doc$_2$ contains "dog": 1100
doc$_3$ contains "cat" and "dog": 1110

Does *doc*$_3$ contain "database"?

➲Some false positives; no false negatives

Bit-sliced signature files

- Motivation
  - To check if a document contains a word, we only need to check the bits that are set in the word's hash value
  - So why bother retrieving all w bits of the signature?
- Instead of storing n signature files, store w bit slices
- Only check the slices that correspond to the set bits in the word's hash value
- Start from the sparse slices

doc	*Signature*
1	0 0 0 0 1 0 0 0
2	0 0 0 0 1 0 0 0
3	0 0 0 1 1 0 1 0
4	0 1 1 0 1 1 0 0
…	…
N	0 0 0 0 1 0 1 0

Slice 7 ··· Slice 0

Bit-sliced signature files

Starting to look like an inverted list again!

Inverted lists versus signatures

- Inverted lists better for most purposes (TODS, 1998)
- Problems of signature files
  - False positives
  - Hard to use because s, w, and the hash function need tuning to work well
  - Long documents will likely have mostly 1's in signatures
  - Common words will create mostly 1's for their slices
  - Difficult to extend with features such as frequency, proximity
- Saving grace of signature files
  - Sizes are tunable
  - Good for lots of search terms
  - Good for computing similarity of documents

Ranking result pages

- A single search may return many pages
  - A user will not look at all result pages
  - Complete result may be unnecessary

➲Result pages need to be ranked

- Based on content
  - Number of occurrences of the search terms
  - Similarity to the query text
- Based on link structure
  - Backlink count
  - PageRank
  - HITS
- And more…

Textual similarity

- Terms $\{t_1, \ldots, t_n\}$ and documents $D = \{d_1, d_2, \ldots\}$
- IDF (Inverse Document Frequency) of t_i:
  - $\mathrm{idf}_i = -\log(|\{d_j: t_i \in d_j\}| / |D|)$
- TF (Term Frequency) of t_i in d_j:
  - $\mathrm{tf}_{i,j}$ = (# of times t_i appears in d_j) / (total # of term occurrences in d_j)
- TF-IDF weight vector of d_j: $\mathbf{w}_j = \langle \mathrm{tf}_{1,j}\mathrm{idf}_1, \ldots, \mathrm{tf}_{n,j}\mathrm{idf}_n \rangle$
- Textual similarity between d_j and d_k can be measured by the normalized dot product of these vectors $(\mathbf{w}_j \cdot \mathbf{w}_j)/(\|\mathbf{w}_j\|_2 \cdot \|\mathbf{w}_k\|_2)$
  $= (\sum_i \mathrm{tf}_{i,j}\mathrm{tf}_{i,k}\mathrm{idf}_i^2) / (\mathrm{sqrt}(\sum_i \mathrm{tf}_{i,j}^2\mathrm{idf}_i^2)\ \mathrm{sqrt}(\sum_i \mathrm{tf}_{i,k}^2\mathrm{idf}_i^2))$
  - A query can be treated as a document

Why weigh by IDF?

- Without IDF weighting, the similarity measure would be dominated by the stop words
- "the" occurs frequently on the Web, so its occurrence on a particular page should be considered less significant
- "engine" occurs infrequently on the Web, so its occurrence on a particular page should be considered more significant

Ranking by content?

- Many pages containing search terms may be of poor quality or irrelevant
  - Example: a page with just a line "search engine"
- Many high-quality or relevant pages do not even contain the search terms
  - Example: Google homepage
- Page containing more occurrences of the search terms are ranked higher; spamming is easy

Term spamming techniques

- Repetition of one or few specific terms
  - E.g., cheap, viagra
- Dumping of a large number unrelated terms
  - E.g., copy the entire dictionary
- Weaving together legitimate content and spam terms (at random positions)
- Phrase stitching: gluing together sentences and phrases from different sources

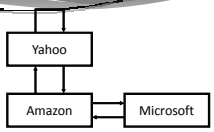
Backlink

- A page with more backlinks is ranked higher
- Intuition: Each backlink is a "vote" for the page's importance
- Based on local link structure; still easy to spam
  - Create lots of pages that point to a particular page

Google's PageRank

- Main idea: pages pointed by high-ranking pages are ranked higher
  - Definition is recursive by design
  - Based on global link structure; harder to spam
- Naïve PageRank
  - $F(p)$: set of pages that page p points to
  - $B(p)$: set of pages that point to p
  - $\mathrm{PageRank}(p) = \Sigma_{q \in B(p)} (\mathrm{PageRank}(q) / |F(q)|)$
  - ➲Each page gets a boost from every page pointing to it
  - ➲Each page distributes its importance evenly to pages that it points to

Example

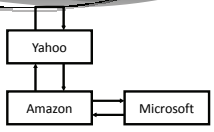


- $y = y/2 + a/2$
- $a = y/2 + m$
- $m = a/2$
- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint enforces uniqueness
  - $y + a + m = 1$
  - $y = 2/5$, $a = 2/5$, $m = 1/5$
- Gaussian elimination works for small examples, but we need a better method for large graphs

Matrix formulation

- Matrix $\mathbf{M}$ has one row/column for each page
  - If page j points to page i, $\mathbf{M}_{ij} = 1/|F(j)|$
    - Otherwise, $\mathbf{M}_{ij} = 0$
  - $\mathbf{M}$ is a left (column) stochastic matrix
    - I.e., each column sums up to 1
- Rank vector $\mathbf{r}$ has one entry per page
  - $r_i \geq 0$ is the importance score of page i
  - $\|\mathbf{r}\|_1 = \sum_i |r_i| = 1$
- Naïve PageRank: $\mathbf{r} = \mathbf{M}\mathbf{r}$
  - I.e., $\mathbf{r}$ is the principle eigenvector of $\mathbf{M}$, corresponding to the eigenvalue 1

Example revisited

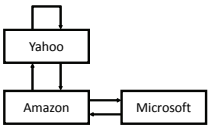


- $y = y/2 + a/2$
- $a = y/2 + m$
- $m = a/2$

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0 & 1 \\ 0 & 0.5 & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

Power iteration method

- Suppose there are N pages
- Initialize: $\mathbf{r}^{(0)} \leftarrow [1/N, \ldots, 1/N]^T$
- Iterate: $\mathbf{r}^{(k+1)} \leftarrow \mathbf{M}\mathbf{r}^{(k)}$
  - Stop when $\|\mathbf{r}^{(k+1)} - \mathbf{r}^{(k)}\| < \varepsilon$
- Example



$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0 & 1 \\ 0 & 0.5 & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

$$\begin{array}{cccccccc} y & & 1/3 & 1/3 & 5/12 & 3/8 & & 2/5 \\ a & = & 1/3 & 1/2 & 1/3 & 11/24 & \cdots & 2/5 \\ m & & 1/3 & 1/6 & 1/4 & 1/6 & & 1/5 \end{array}$$

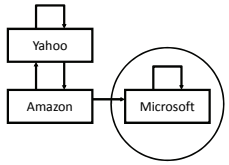
Random surfer model

- A random surfer
  - Starts with a random page
  - Randomly selects a link on the page to visit next
  - Never uses the "back" button
- $\mathrm{PageRank}(p)$ measures the probability that a random surfer visits page p
  - Rank vector $\mathbf{r}$ is a stationary distribution for the random walk
  - For graphs with certain properties, the stationary distribution is unique and eventually will be reached no matter what the initial probability distribution is

Spider traps

- A group of pages with no links out of the group
  - Will accumulate all importance of the Web

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$



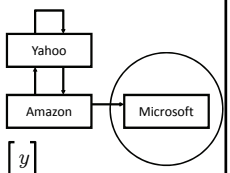
$$\begin{array}{cccccccc} y & & 1 & 1 & 3/4 & 5/8 & & 0 \\ a & = & 1 & 1/2 & 1/2 & 3/8 & \cdots & 0 \\ m & & 1 & 3/2 & 7/4 & 2 & & 3 \end{array}$$

Random teleports

- A surfer occasionally gets bored (with probability β) and jumps to a random page on the Web instead of following a random link on the current page
  - Surfer will get out of a spider trap in some time
  - Common values for β are between 0.8 and 0.9
- Revised PageRank formulation
  - $\mathbf{A} = \beta\mathbf{M} + (1-\beta)/N$ and $\mathbf{r} = \mathbf{A}\mathbf{r}$
- Equivalent to
  - Add a teleport link for each (i, j), with weight $(1-\beta)/N$
  - Reduce weight of each outlink from $1/|F(i)|$ to $\beta/|F(i)|$

Dead ends

- A page with no outgoing links
  - Will cause all importance to "leak" eventually



$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = (0.8 \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix} + 0.2\times 1/3) \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

Not even stochastic!

$$\begin{matrix} y \\ a \\ m \end{matrix} \quad = \quad \begin{matrix} 1 & 1 & 0.787 & 0.648 & \\ 1 & 0.6 & 0.547 & 0.430 & \cdots \\ 1 & 0.6 & 0.387 & 0.333 & \end{matrix} \quad \begin{matrix} 0 \\ 0 \\ 0 \end{matrix}$$

Dealing with dead ends

- Teleport
  - Follow random teleport links with probability 1 from dead ends
  - Adjust matrix accordingly
- Prune and propagate
  - Preprocess the graph to eliminate dead ends
    - Might require multiple passes!
  - Compute PageRank on reduced graph
  - Approximate values for dead ends by propagating values from reduced graph

Computing PageRank

- Key step is matrix-vector multiplication in power iteration: $\mathbf{r}^{(k+1)} \leftarrow \mathbf{A}\mathbf{r}^{(k)}$

  $= \beta\mathbf{M}\mathbf{r}^{(k)} + (1-\beta)/N \times \mathbf{1}(N,N)\ \mathbf{r}^{(k)}$

  $= \beta\mathbf{M}\mathbf{r}^{(k)} + (1-\beta)/N$
- Easy if we have enough memory to hold $\mathbf{A}$ and two copies of $\mathbf{r}$, but
  - Say $N = 10^9$ (1 billion)
  - Vectors: 8GB
  - Matrix: 4×10^{18}B = 4000PB

Sparse matrix encoding

- Store only the non-zero entries in $\mathbf{M}$
- Space $\propto N \times$ (# of outlinks per page)
  - Say $N = 10^9$ (1 billion), and 10 outlinks on average
  - Matrix: $4\times10\times10^9$B = 40GB
  - Still won't fit in memory, but will fit on disk

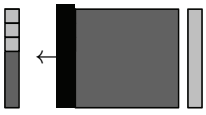
source node	degree	destination nodes
		= One column of $\mathbf{M}$

Basic algorithm (attempt 1)

- Assume enough memory to fit $\mathbf{r}$, plus some working memory

In iteration k+1:

- Read in all of $\mathbf{r}^{(k)}$
- Stream in $\mathbf{M}$ in row-major order, an entry at a time
- Stream out $\mathbf{r}^{(k+1)}$, an entry for each row of $\mathbf{M}$



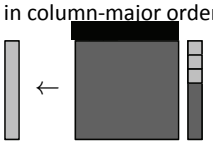
➲But $\mathbf{M}$ is stored in column-major order!

Basic algorithm

- Again, assume enough memory to fit $\mathbf{r}$, plus some working memory

In iteration k+1:

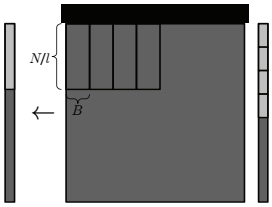
- Reserve space for $\mathbf{r}^{(k+1)}$
- Stream in $\mathbf{r}^{(k)}$, one entry for each column of $\mathbf{M}$
- Stream in $\mathbf{M}$ in column-major order, an entry at a time



➲But what if we can't even fit a vector in memory?

Block-stripe algorithm

- Assume one disk block holds B entries
- Break $\mathbf{r}^{(k+1)}$ into l chunks
  - Memory can hold one chunk plus some working memory
- Process $\mathbf{M}$ as vertical stripes ($N/l \times B$)
- For each $\mathbf{r}^{(k+1)}$ chunk
  - Stream in $\mathbf{r}^{(k)}$ one block at a time
  - Stream in the corresponding stripe of $\mathbf{M}$, one row at a time



l passes over $\mathbf{r}^{(k)}$
one pass over $\mathbf{M}$

Is this optimal?

Topic-specific PageRank

[Haveliwala, WWW '02]

- Instead of generic popularity, how about popularity within a topic?
  - E.g., computer science, health
- Bias the random walk
  - When teleporting, pick only from a set S containing just pages relevant to the topic
    - E.g., Open Directory (http://dmoz.org/) pages on topic
- A different rank vector $\mathbf{r}_s$ for each teleport set S
- $\mathbf{A}_{ij} = \beta\mathbf{M}_{ij} + (1-\beta)/|S|$ if $i \in S$ or $\beta\mathbf{M}_{ij}$ otherwise
  - $\mathbf{A}$ remains stochastic

TSPR discussion

- We have weighted all pages in S equally
  - Could also assign different weights to them
- Which topic ranking should we use?
  - Classify query into a topic
  - Let user pick from a menu
  - Use query context
    - Query launched from a page on a specific topic
    - Search history, e.g., "basketball" and then "jj"
  - Use user's context
    - My Yahoo! settings, bookmarks, purchase history, …

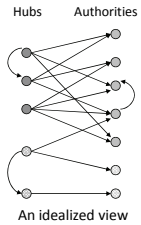
HITS (hubs & authorities)

- Given a collection of pages on some broad topic
  - E.g., Duke, Obama, or perhaps those returned by a text search
- Can we organize them in some manner?
  - Rank them, like in PageRank?
  - Identify hubs, authorities, cores?
- HITS: Hypertext-Induced Topic Selection
  - [Kleinberg, JACM '99]
  - Approximately the same time as PageRank

HITS model

Interesting pages fall into two classes

- Authorities are those containing useful info
  - Course home pages
  - Sites of auto manufacturers
- Hubs are those that point to authorities
  - Course bulletin
  - List of auto manufactures

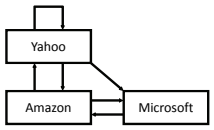


An idealized view

Mutually recursive definition

- A good hub links to many good authorities
- A good authority is linked from many good hubs
- A matrix $\mathbf{A}$ represents the connection graph $\mathbf{A}_{ij}$ = 1 if page i links to page j; 0 otherwise
  - $\mathbf{A}^T$, the transpose of $\mathbf{A}$, is similar to the PageRank matrix $\mathbf{M}$, though $\mathbf{A}^T$ has 1's while $\mathbf{M}$ has fractions

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



Definition cont'd

- A hub score and an authority score for each node
  - Represented by two vectors $\mathbf{h}$ and $\mathbf{a}$
- The hub score of a page $\propto$ the sum of the authority scores of the pages it links to
  - $\mathbf{h} = \lambda\, \mathbf{A}\, \mathbf{a}$
  - λ is a scale factor
- The authority score of a page $\propto$ the sum of the hub scores of the pages it is linked from
  - $\mathbf{a} = \mu\, \mathbf{A}^T \mathbf{h}$
  - μ is a scale factor

Iterative algorithm, again

- Initialize $\mathbf{h}$, $\mathbf{a}$ to all 1's
- $\mathbf{h} \leftarrow \mathbf{A}\, \mathbf{a}$
- Scale $\mathbf{h}$ so that its max entry is 1
- $\mathbf{a} \leftarrow \mathbf{A}^T \mathbf{h}$
- Scale $\mathbf{a}$ so that its max entry is 1
- Continue until $\mathbf{h}$, $\mathbf{a}$ converge

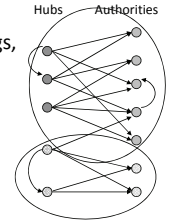
a(yahoo)	=	1	1	1	1	...	1
a(amazon)	=	1	1	4/5	0.75	...	0.732
a(msft)	=	1	1	1	1	...	1
h(yahoo)	=	1	1	1	1	...	1.000
h(amazon)	=	1	2/3	0.71	0.73	...	0.732
h(msft)	=	1	1/3	0.29	0.27	...	0.268

Eigenvectors, again

- Under reasonable assumptions about $\mathbf{A}$, the dual iterative algorithm converges to vectors $\mathbf{h}^*$ and $\mathbf{a}^*$, where
  - $\mathbf{h}^*$ is the principal eigenvector of the matrix $\mathbf{A}\, \mathbf{A}^T$
  - $\mathbf{a}^*$ is the principal eigenvector of the matrix $\mathbf{A}^T \mathbf{A}$

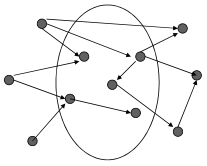
Secondary cores

- A single topic can have many bipartite cores
  - Corresponding to different meanings, or points of view
    - Abortion: pro-choice, pro-life
    - Jaguar: auto, Mac, NFL team, *panthera onca*
- To find the secondary cores
  - Once we find the primary core, remove its links
  - Repeat HITS on the residual graph to find the next bipartite core
  - Roughly, these cores correspond to non-primary eigenvectors of $\mathbf{A}\, \mathbf{A}^T$ and $\mathbf{A}^T \mathbf{A}$



HITS discussions

- Need a well-connected graph of pages for HITS to work well
  - ➲Extend the initial set
- PageRank vs. HITS
  - What is the value of an inlink from p to q?
  - In PageRank, the value depends on the links into p
  - In HITS, it depends on the values of other links out of p



Web spam

- Search is now the default gateway to the Web
  - Very high premium to appear on the first page of search results
- Spamming: deliberately actions solely in order to boost a page's rank that is incommensurate with its real value
  - SEO (Search Engine Optimization) industry might disagree!

➲Approximately 10-15% of Web pages are spam

Web spam taxonomy

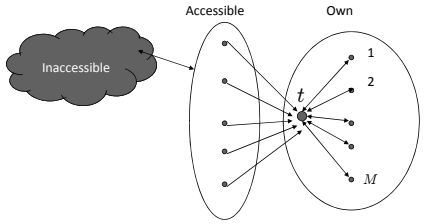
[Gyongyi & Garcia-Molina, VLDB '04]

- Boosting techniques: to achieve high rank
  - Term spamming (covered earlier)
  - Link spamming: create link structures that boost PageRank or hub/authority scores
- Hiding techniques: to hide the use of boosting
  - From humans and Web crawlers

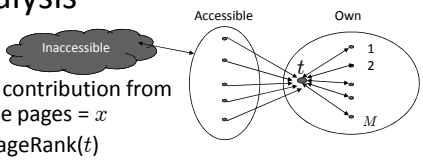
Link spam

- Three kinds of pages from a spammer's point of view
  - Inaccessible pages
  - Accessible pages
    - E.g., can post links on blog comments pages
  - Own pages: completely controlled by spammer
    - May span multiple domain names
- How to maximize the PageRank of a target page t
  - Get as many links from accessible pages as possible to t
  - Build a "link farm" to get the PageRank multiplier effect

A common type of link farm



Analysis



- Suppose contribution from accessible pages = x
- Let y = PageRank(t)
- PageRank of each farm page = $\beta y/M + (1-\beta)/N$
- $y = x + \beta M [\beta y/M + (1-\beta)/N] + (1-\beta)/N$
  $= x + \beta^2 y + \beta(1-\beta) M/N + \boxed{(1-\beta)/N}$ Small; ignore
- $y = x / (1-\beta^2) + c\, M/N$, where $c = \beta / (1+\beta)$
  - If $\beta = 0.85$, $1/(1-\beta^2) = 3.6$
    - ➲Multiplier effect for the acquired PageRank
  - By making M large, we can make y as large as we want

Detecting spam

- Term spamming
  - Analyze text using statistical methods, similar to email spam filtering
  - Also useful: detecting approximate duplicate pages
- Link spamming
  - TrustRank [Gyongyi et al., VLDB '04]
  - Spam mass estimation [Gyongyi et al., VLDB '06]

TrustRank idea

- Approximate isolation: it is rare for a good page to point a bad one
- Identify a set of "trusted pages"
- Propagate "trust" through links
- Use a threshold and mark all pages below it as spam

Trust propagation

- Trust attenuation
  - The degree of trust conferred by a trusted page decreases with distance
- Trust splitting
  - The larger the number of outlinks from a page, the less scrutiny the page author gives each outlink
- A simple model
  - TrustRank of page p: $t(p) = \beta \sum_{q \in B(p)} t(q) / |F(q)|$
    - $\beta \in (0, 1)$ is the attenuation factor
    - Note similarity to topic-specific PageRank; treat the initial set of trusted pages as teleport set

Picking the initial trusted set

- Conflict goals
  - Rely on human to inspect, so the set must be small
  - To ensure every good page gets an adequate TrustRank, need to make all good pages reachable from the set by short paths
- Pick the top k pages by PageRank?
- Pick the top k pages by inverse PageRank?
  - Pick the pages with many outlinks to pages with many outlinks (recursive)…
  - Construct G' by reversing all edges in Web graph G
  - PageRank in G' = inverse PageRank in G

Spam mass

- TrustRank started with good pages and propagated trust
- Complementary view: what fraction of a page's PageRank comes from spam pages?
- PageRank of p, $r(p)$, can be written as the sum of two components
  - $r^+(p)$: sum of contributions from "good" pages $\mathcal{V}^+$
  - $r^-(p)$: sum of contributions from "bad" pages $\mathcal{V}^-$
- Spam mass of $p = r^-(p)/r(p)$

Spam mass estimation

- $\mathcal{V}^+$ and $\mathcal{V}^-$ are not known exactly
- Use a large set of reasonably good (or bad) pages to compute an estimate for $r^+(p)$ (or $r^-(p)$)
  - Need not be as careful about quality of individual pages as with TrustRank
- Examples
  - White- and black-lists maintained by search engines
  - Approximate $\mathcal{V}^+$ with
    - .edu sites
    - .gov sites
    - .mil sites

Link spamming: a summary

- Still an open area of research
- Interesting read: [Najork, Zaragoza, Taylor, SIGIR '07]
  - Does people's understanding of the ranking algorithms make these algorithms less effective than before?