15-499:Algorithms and Applications

Indexing and Searching IV
- Link Analysis
- Near duplicate removal

Indexing and Searching Outline

Introduction: model, query types
Inverted Indices: Compression, Lexicon, Merging
Vector Models: Weights, cosine distance
Latent Semantic Indexing:
Link Analysis: PageRank (Google), HITS
Near Duplicate Removal:

Link Analysis

The goal is to rank pages.

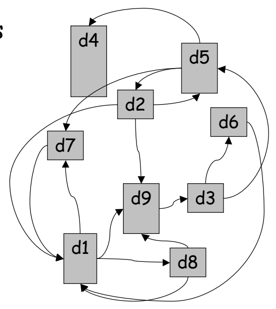
We want to take advantage of the link structure to do this.

Two main approaches:
- **Static**: we will use the links to calculate a ranking of the pages offline (Google)
- **Dynamic**: we will use the links in the results of a search to dynamically determine a ranking (Clever – hubs and authorities)

The Link Graph

View documents as graph nodes and the hyperlinks between documents as directed edges.

Can give weights on edges (links) based on
1. position in the document
2. weight of anchor term
3. # of occurrences of link
4. …



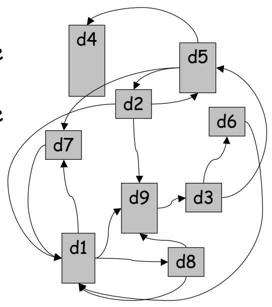
The Link Matrix

An adjacency matrix of the link graph.

Each row corresponds to the Out-Links of a vertex

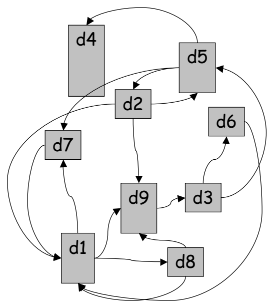
Each row corresponds to the In-links of a vertex

Often normalize columns to sum to 1 so the dot-product with self is 1.



The Link Matrix

0	0	0	0	0	0	1	1	1
1	0	0	0	1	0	0	0	1
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0

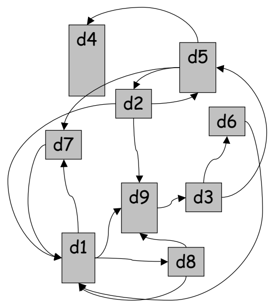


Google: Page Rank (1)

Goal: to give a total order (rank) on the "importance" of all pages they index

Possible Method: count the number of in-links.

Problems?



Google: Page Rank (2)

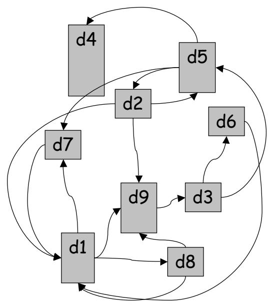
Refined Method: weigh the source by its importance.

Seems like this is a self recursive definition.

Imagine the following:

Jane Surfer, has a lot of time to waste, and randomly selects an outgoing link each time she gets to a page.

She counts how often she visits each page.



Google: Page Rank (3)

Remaining problem: Gives too high of a rank to the Yahoo's of the world, and too low to clusters with low connectivity into the cluster.

Solution: With some probability jump to a random page instead a random outgoing link.

But Simulating Jane's process will take a long time. Can we calculate the result faster?

Stationary probability of a Markov Chain.

Transition probability matrix is:

$$T = \varepsilon U + (1-\varepsilon)A$$

U is the uniform matrix, and A is the normalized link matrix (each column sums to 1).

Google: Page Rank (4)

Want to find p, such that Tp = p

This corresponds to finding the principal eigenvector.

Methods:

- **Power method**: iterate $p_{i+1} = Tp_i$ until it settles
  pick p_0 randomly,
  decomposing p_0 into eigenvectors $a_1 e_1 + a_2 e_2 + \dots$
  we have $p_i = \lambda_1^i a_1 e_1 + \lambda_2^i a_2 e_2 + \dots$
- **Multilevel method**: solve on contracted graph, use as initial vector for power method on the expanded graph.
- **Lancoz method**: diagonalize, then iterate

All methods are quite expensive

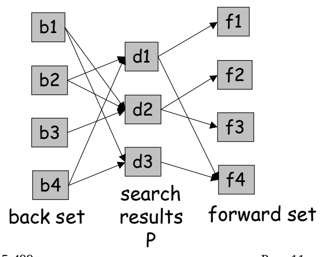
Hubs and Authorities (1)

Goal: To get a ranking for a particular query (instead of the whole web)

Assume: We have a search engine that can give a set of pages P that match a query

Find all pages that point to P **(back set)** and that P point to **(forward set)**.

Include b, d and f in a document set D.



Hubs and Authorities (2)

For a particular query we might want to find:

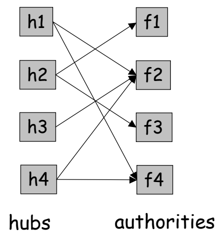
1. The "authorities" on the topic (where the actual information is kept)
2. The "hub" sites that point to the best information on the topic (typically some kind of link site)

Important authorities will have many pages that point to them

Important hubs will point to many pages

But: As with page-rank, just having a lot of pointers does not mean much

Hubs and Authorities (3)



vector h is a weight for each document in D for how good a hub it is

vector a is a weight for each document in D for how good an authority it is

T is the adjacency matrix

Now repeat:

$h = Ta$

$a = T^T h$

until it settles.

Hubs and Authorities (4)

What is this process doing?

$a_{i+1} = (TT^T)\, a_i$

$h_{i+1} = (T^T T)\, h_i$

Power method for eigenvectors of TT^T and $T^T T$

SVD(T) : first column of U and first row of V

Hubs and Authorities (5)

Problem (Topic Drift): Hubs and Authorities will tend to drift to the general, instead of specific.

e.g. you search for "Indexing Algorithms" which identifies all pages with those terms in it, and the back and forward pages.
Now you use the SVD to find Hubs + Authorities
These will most likely be pages on Algorithms in general rather than Indexing Algorithms.

Solution: weight the documents with "indexing" prominently in them, more heavily.

$a = TWh$, $h = T^T Wa$, where W is a diagonal matrix with weights on the diagonal.

Same as finding SVD of $T' = TW$

Extensions

What about second eigenvectors?

Mix terms and links.

Indexing and Searching Outline

Introduction: model, query types
Inverted Indices: Compression, Lexicon, Merging
Vector Models: Weights, cosine distance
Latent Semantic Indexing:
Link Analysis: PageRank (Google), HITS
➡ **Near Duplicate Removal:**

Syntacting Clustering of the Web

Problem: Many pages on the web are almost the same but not exactly the same.
- Mirror sites with local identifier and/or local links.
- Spam sites that try to improve their "pagerank"
- Plagerised sites
- Revisions of the same document
- Program documentation converted to HTML
- Legal documents

These near duplicates cause web indexing sites a huge headache.

Syntactic Clustering

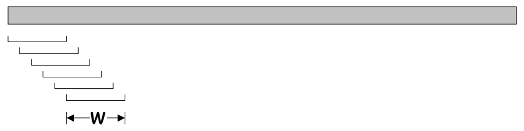
Goal: find pages for which either
1. Most of the content is the same
2. One is contained in the other (possibly with changes

Constraints:
- Need to do it with only a small amount of information per document: a **sketch**
- Comparison of documents should take time linear in the size of the sketch

Syntactic Clustering

Any ideas?

w-Shingling



View each shingle as a value.

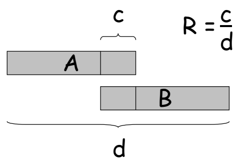
$S_w(A)$: the set of w-shingles for document A.

As shorthand, I will often drop the w subscript.

Resemblance and Containment

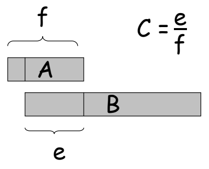
Resemblance:

$$R(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$



Containment:

$$C(A,B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$$



Calculating Resemblance/Containment

Method 1:

Keep all shingles and calculate union and intersection of the shingles directly.

Problem: can take |D|w space per document (larger than original document)

Other ideas?

- First 100 shingles?
- Shingles from every kth position?
- Shingles from random positions (preselected)? e.g. 23, 786, 1121, 1456, …

Using Min Valued Shingles

How about viewing each shingle as a number and selecting the 100 minimum valued shingles?

- e.g. append the ASCII codes

Possible problem?

Hash

How about a random hash?

i.e. for each shingle x take h(x), then pick the minimum k values.

$$Sk(A) = \min_k\{h(x) : x \in S(A)\}$$

Some Theory: Pairwise independent

Universal Hash functions: (Pairwise independent)

- *H* : a finite collection (family) of hash functions mapping U $\rightarrow$ {0…m}
- H is **universal** (or pairwise independent) if,
  - for $h \in H$ picked uniformly at random,
  - and for all $x_1, x_2 \in U$, $x_1 \neq x_2$

  $Pr(h(x_1) = h(x_2)) \leq 1/m$

The class of hash functions

- $h_{ab}(x) = ((a\,x + b) \bmod p) \bmod |M|$

is universal.

Some Theory: Minwise independent

Minwise independent permutations:

- S_n : a finite collection (family) of permutations mapping {1…n} $\rightarrow$ {1…n}
- H is **minwise independent** if,
  - for $\pi \in S_n$ picked uniformly at random,
  - and for X = {1…n}, and all $x \in X$

  $Pr(\min\{\pi(X)\} = \pi(x)) \leq 1/m$

It is actually hard to find a "compact" collection of hash functions that is minwise independent, but we can use an approximation.

Back to similarity

If S_n is minwise independent then:

$$\Pr\left(\min\{\pi(S(A))\} = \min\{\pi(S(B))\}\right) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|} = r(A,B)$$

This suggests we could just keep one minimum value as our "sketch", but our confidence would be low.

What we want for a sketch of size ***k*** is either

- use ***k*** π's,
- or keep the ***k*** minimum values for one π