22 Page Rank

At its simplest, a search engine is an *inverted index*. For each search term (say `pie`) there is a pointer to a list of webpages about `pie`. If the search has two terms, if they are a popular pairing (say `apple pie`) then there may be a dedicated list. If not, then we need to take both lists and return their intersection.

But, a search engine only returns 10 items (and some higher on the page than others). Which ones on the list should it return?

22.1 Webpage Similarity

The first answer to this question, and the one that built companies like Altavista, Lycos, Infoseek, was to define a webpage *similarity* function. Given a key word, webpages would automatically be ranked by their similarity to that keyword. The more similar webpages would be ranked higher by the search engine.

As soon as there were popular search engines, webpages tried to figure out how to optimize for them. Sometimes, search engines actually helped with this (header tags in html).

A simple example of a similarity function is *cosine similarity* between the keywords searched and the set of words on the webpage.

- How could this be fooled? .... (term spam)

Repeat the word `pie` 1000 times at the bottom of the page. Change the color of the text to the background color of the page, make it really small.

OK. So the search engines change there similarity function. Needs to be include `pie`, but not repeat it too many times. Maybe needs to include other words which typically occur with the word `pie`, such as `apple`, `pan`, `hot`, `delicious`, `recipe`.

- How could this be fooled?

(Find the top several ranked pages, copy their source at the bottom of your page.)

Crawlers. How did the search engines find these webpages in the first place? There were programs called crawlers that followed the trail of links between pages. Start at `www.pie.com`, collect data on that page, then follow a link to another webpage, and repeat. Many of these were run in parallel from many starting points ... and still are.

OK. So a key word is also important for a webpage if there is a hyperlink from another webpage with the word `pie` as the word you click on to get to this webpage.

- How could this be fooled?

(Create a bunch of dummy webpages ("spam farm") that only link to your webpage).

It was a constant battle, not just for speed, but also for modeling. A true "wild west" anything goes culture!

Indexes. A competing approach was an *index*, for example Yahoo! or LookSmart. They personally collected and organized links to webpages they thought were important (or webpages that they were paid to list). These sites were more reliable (except sometimes for paid links), but not quite as easy to use. LookSmart died, but Yahoo! is still in business partially under this model, but after creating a bunch of their own content; the (personalized) Yahoo! front page is still one of the most visited sites on the web.

22.2 PageRank

Then along came PageRank, and the game changed entirely. It was based on simple idea (and a clever execution):

IDEA 1: *Pages are important if they are linked to by other important pages.*

This sounds like a chicken and egg problem, but seems very natural.

Say page p_1 has text `T1` and links to p_2. Also page p_2 has text `T2`. If the search term `pie` is in `T1` $\cap$ `T2` then this makes p_1 a good choice. Even better if `pie` is in the hyperlink pointing from p_2 to p_1.

But if p_1 is hard to get to, it is not important, then this is still not that important of a piece of information. But this can still be tricked by "spam farms".

IDEA 2: *A page is important if a "random surfer" were likely to find it.*

The internet is a big directed graph $G = (V, E)$. Each $v \in V$ is a webpage. Each directed edged $e = (v_i, v_j)$ is a link from page v_1 to page v_2.

A random surfer starts on some page, clicks a random link on that page, and then goes to the next page. This continues (just like a crawler).

This process defines a Markov chain! Converged to distribution $q_* = P^* q$ gives the important $q_*[v]$ of a webpage $v \in V$. That is, the importance of a webpage v is defined as $q_*[v]$.

Now, the ranking of page v is defined as

$$S(v, \texttt{term}) = \textsf{magic}(q_*[v], \textsf{text}(v), \textsf{text}(e(v', v), q_*[v'])).$$

So there is still some proprietary magic here, but it involves the importance of the page $q_*[v]$, the text on the page $\textsf{text}(v)$, and the hyperlink text pointing to the page $(e(v', v)$ weighted by the importance $q_*[v']$ of the page v' pointing to v.

For instance, if both Yahoo! index and LookSmart index (which may have high importance) point to your webpage with a link that says `pie`, then this would probably have increased your rank on Google with the search term `pie`.

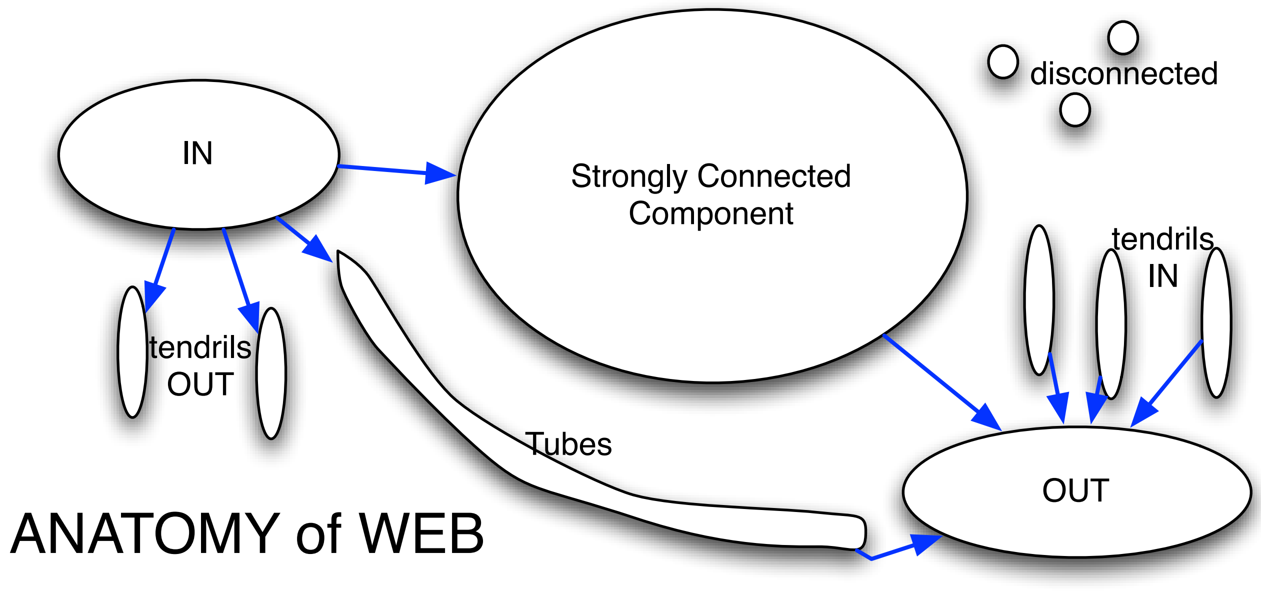
Computing q_*. So how do we compute q_*?
Running many random walks (e.g. MCMC) like web crawlers is inefficient. Those are optimized for coverage, and will take a while to converge. MCMC samples from the Markov chain, but does not do a good job of touching everything.

Should we compute $P^* = P^n$ for some large n (and then simply run $q_* = P^* q$ for any q)?
No, this takes a lot of space! There are already m (millions and millions) of webpages, and P^* is $n \times n$ and dense. P is sparse, but P^7 is already dense (its a small world, you can get from almost any webpage to any other in 7 steps).

So we compute q_i and then update it to $q_{i+1} = Pq_i$, and repeat for some large enough i (usually 50 is good, especially if we have a good starting spot, the q_* from last run).

22.3 Anatomy of Web

So is this enough? Is the web graph *ergodic*? ... No.
It may not be connected. It has transient nodes (lonely webpages that nobody links to). It is unlikely to be cyclic (one self loop prevents this).



- The web graph has one pretty strongly connected component (SCC). Here all pages can surf to all other pages.
- The IN component can get to itself, and to the SCC.
- The OUT component can get to itself, and can be reached by the SCC.
- There *might* be some tubes directly from IN to OUT that bypass SCC.
- There are tendrils that point to OUT, but cannot get back.
- There are tendrils that are pointed to from IN, but then don't go anywhere.
- There are disconnected parts (but relatively few in comparison).

Is this ergodic? No.

All the probability accumulated in OUT. It is like the "Hotel California," you can check-in, but you can never leave. These parts of the web are known (in some circles) as *spider traps*, since the web crawlers (get it? something that "crawls" the "web" is a spider), get stuck there.

So what is the solution (and your solution if you are browsing randomly and reach a page with no out edges?). ... Jump to a random page.

Taxation (or Teleportation). In PageRank, each transition step has a β probability of jumping to a random page (typically $\beta = 0.10$ – you go about 10 webpages deep, and then jump someplace random). The graph is now *totally connected, has no transient nodes, and cannot be cyclic*, it is ergodic. Also as a side benefit, it has no spider traps, and mixes much faster.

However, now P is dense. But we can actually let Q be a $m \times m$ matrix with $1/m$ in each entry. Then set $q_{i+1} = ((1-\beta)P + \beta Q)q_i$, which can be implemented efficiently since Q can be stored implicitly.

22.3.1 Spam Farms

But PageRank is not impervious to spam farms. It is possible to create a large number of webpages (comment sections on blogs – they only need crawlers to use the link, not people, etc).
How can we use these webpages to fool PageRank?

- Create a few *target* pages (which we want Google to rank highly).
- Have all *corrupted* pages send links to target pages (or proxy page, then target.)
- Have few targets link to each other.

The corrupted pages accumulated taxation (or other traffic), and direct this to the target pages. Once it reaches the target pages, it only leaves β fraction of the time.

So how does Google defeat spam farms?
Search for structure, then *blackball* it (don't rank it). But the structure can be modified a bit and still work. It is still an on going battle.

TrustRank. Some pages are more or less trustworthy:

- MORE: Wikipedia, .edu domains, .mil and .gov domains. Main Amazon pages (without comments), and VERY high PageRank pages.
- LESS: blogs, twitter, any pages with comments

Pages with more *trust weight* get teleported to more frequently, and taxed less heavily.

We can also try to classify pages as spam as follows. Create regular page rank $r(v)$ and a TrustRank $t(v)$ for a page v. Then if $s(v) = (r(v) - t(v))/r(v)$ is small or negative, then is NOT spam. If s is large, then likely spam.

Recently (2015) Google proposed using a measure of ***truth*** *to rank webpages. What could go wrong?*

Personalized PageRank. Idea: each user j has preference of which pages they find important, by a unit basis vector h_j. This can be estimated based on past search results, or can be set to the user. These can then be jumped to more frequently on distribution step, denoted as

$$q_{i+1} = ((1-\beta)P + \beta h_j)q_i.$$

Let the converged to vector $q_*^{(h_j)}$ be the personalized PageRank vector.

However, doing this for each user is very expensive. PageRank was great in that it denoted the *global importance* of webpages, and then could be combined with rankings based on the query. So it just needed to be computed once for everyone. So how can we speed this up?

- we restrict h_j to be from a hub set H of vectors (e.g. trusted basic `www.cs.utah.edu`, not `www.cs.utah.edu/~jeffp/teaching/cs5140.html`).

- Construct an orthogonal basis over H (maybe a partial one with PCA over everyones $\{h_j\}_j$ vectors), to get a set of basis vectors $B = [b_1, b_2, \ldots, b_k]$ in the column space of H.

- Run personalized page rank with each

  $$q_{i+1} = ((1-\beta)P + \beta b_\ell)q_i.$$

  to get a personalized PageRank vector that the above Markov Chain converges to $q_{*,\ell}$. There are actually some tricks to compute all of these at once.

- Then each individual's person h_j can be written as $h_j = \sum_{\ell=1}^k \alpha_\ell^{(j)} b_\ell$ using the basis B. We can then report

  $$q_*^{(h_j)} = \sum_{\ell=1}^{k} \alpha_\ell^{(j)} q_{*,\ell}.$$

  Some pretty cool theory shows this gives the right result (up to the accuracy of the PCA if it was used on H). *There maybe some subtleties of the reconstruction I am leaving out due to the approximation based on the use of H. (See (Jeh+Widom @ WWW 2003) for more info)*