10. IR on the World Wide Web and Link Analysis

These notes are based, in part, on notes by Dr. Raymond J. Mooney at the University of Texas at Austin.
IR on the Web vs. Classic IR

• **Input:** publicly accessible Web

• **Goal:** retrieve **high quality** pages that are **relevant** to user’s **need**
  - static (text, audio, images, etc.)
  - dynamically generated (mostly database access)

• **What’s different about the Web:**
  - large volume
  - distributed data
  - Heterogeneity of the data
  - lack of stability
  - high duplication
  - high linkage
  - lack of quality standard
Search Engine Early History

- In 1990, Alan Emtage of McGill Univ. developed Archie (short for “archives”)
  - Assembled lists of files available on many FTP servers.
  - Allowed regex search of these file names.
- In 1993, Veronica and Jughead were developed to search names of text files available through Gopher servers.
- In 1993, early Web robots (spiders) were built to collect URL’s:
  - Wanderer
  - ALIWEB (Archie-Like Index of the WEB)
  - WWW Worm (indexed URL’s and titles for regex search)
- In 1994, Stanford grad students David Filo and Jerry Yang started manually collecting popular web sites into a topical hierarchy called Yahoo.
Search Engine Early History

• In early 1994, Brian Pinkerton developed WebCrawler as a class project at U Wash.
  – Eventually became part of Excite and AOL
• A few months later, Fuzzy Maudlin, a grad student at CMU developed Lycos
  – First to use a standard IR system
  – First to index a large set of pages
• In late 1995, DEC developed Altavista
  – Used a large farm of Alpha machines to quickly process large numbers of queries
  – Supported Boolean operators, phrases in queries.
• In 1998, Larry Page and Sergey Brin, Ph.D. students at Stanford, started Google
  – Main advance was use of link analysis to rank results partially based on authority.
Web Search

Web

Spider

Document corpus

Query String

IR System

Ranked Documents

1. Page1
2. Page2
3. Page3
Spiders (Robots/Bots/Crawlers)

- Start with a comprehensive set of root URL’s from which to start the search.
- Follow all links on these pages recursively to find additional pages.
- Index all novel found pages in an inverted index as they are encountered.
- May allow users to directly submit pages to be indexed (and crawled from).
Search Strategies - BFS

Breadth-first Search
Search Strategies - DFS

Depth-first Search
Search Strategy Trade-Off’s

• Breadth-first search (BFS) strategy explores uniformly outward from the root page but requires memory of all nodes on the previous level (exponential in depth). Standard spidering method.

• Depth-first search (DFS) requires memory of only depth times branching-factor (linear in depth) but gets “lost” pursuing a single thread.

• Both strategies implementable using a queue of links (URL’s).
Avoiding Page Duplication

- Must detect when revisiting a page that has already been spidered (web is a graph not a tree).
- Must efficiently index visited pages to allow rapid recognition test.
  - Tree indexing (e.g. trie)
  - Hashtable
- Index page using URL as a key.
  - Must canonicalize URL’s (e.g. delete ending “/”)
  - Not detect duplicated or mirrored pages.
- Index page using textual content as a key.
  - Requires first downloading page.
**Spidering Algorithm**

Initialize queue (Q) with initial set of known URL’s.
Until Q empty or page or time limit exhausted:
   Pop URL, L, from front of Q.
   If L is not an HTML page (.gif, .jpeg, .ps, .pdf, .ppt…)
      continue loop.
   If already visited L, continue loop.
   Download page, P, for L.
   If cannot download P (e.g. 404 error, robot excluded)
      continue loop.
   Index P (e.g. add to inverted index or store cached copy).
   Parse P to obtain list of new links N.
   Append N to the end of Q.
Queueing Strategy

• How new links added to the queue determines search strategy.

• FIFO (append to end of Q)
  – gives Breadth-First Search.

• LIFO (add to front of Q)
  – gives Depth-First Search.

• Heuristically ordering the Q gives a “focused crawler” that directs its search towards “interesting” pages.
  – May be able to use standard AI search algorithms such as Best-first search, A*, etc.
Restricting Spidering

• Restrict spider to a particular site.
  – Remove links to other sites from Q.

• Restrict spider to a particular directory.
  – Remove links not in the specified directory.

• Obey page-owner restrictions
  – robot exclusion protocol
Multi-Threaded Spidering

- Bottleneck is network delay in downloading individual pages.
- Best to have multiple threads running in parallel each requesting a page from a different host.
- Distribute URL’s to threads to guarantee equitable distribution of requests across different hosts to maximize through-put and avoid overloading any single server.
- Early Google spider had multiple coordinated crawlers with about 300 threads each, together able to download over 100 pages per second.
Directed/Focused Spidering

• Sort queue to explore more “interesting” pages first.

• Two styles of focus:
  – Topic-Directed
  – Link-Directed
Topic-Directed Spidering

• Assume desired topic description or sample pages of interest are given.

• Sort queue of links by the similarity (e.g. cosine metric) of their source pages and/or anchor text to this topic description.

• Preferentially explores pages related to a specific topic.
Link-Directed Spidering

- Monitor links and keep track of **in-degree** and **out-degree** of each page encountered.

- Sort queue to prefer popular pages with many incoming links (authorities).

- Sort queue to prefer summary pages with many out-going links (hubs).
Keeping Spidered Pages Up to Date

- Web is very dynamic: many new pages, updated pages, deleted pages, etc.

- Periodically check spidered pages for updates and deletions:
  - Just look at header info (e.g. META tags on last update) to determine if page has changed, only reload entire page if needed.

- Track how often each page is updated and preferentially return to pages which are historically more dynamic.

- Preferentially update pages that are accessed more often to optimize freshness of more popular pages.
Quality and the WWW
The Case for Connectivity Analysis

• Basic Idea: mine hyperlink information on the Web
• Assumptions:
  – links often connect related pages
  – a link between pages is a “recommendation”
• Approaches
  – classic IR: co-citation analysis (a.k.a. “bibliometrics”)
  – connectivity-based ranking (e.g., Google)
  – HITS - hypertext induced topic search
Co-Citation Analysis

• Has been around since the 50’s (Small, Garfield, White & McCain)
• Used to identify core sets of
  – authors, journals, articles for particular fields of study
• Main Idea: Measure similarity of page A and B by:
  – the number of documents cited by both A and B.
  – The number of documents that cite both A and B.
Co-citation analysis  (From Garfield 98)

The Global Map of Science, based on co-citation clustering:

Size of the circle represents number of papers published in the area;

Distance between circles represents the level of co-citation between the fields;

By zooming in, deeper levels in the hierarchy can be exposed.
Citations vs. Links

- Web links are a bit different than citations:
  - Many links are navigational.
  - Many pages with high in-degree are portals not content providers.
  - Not all links are endorsements.
  - Company websites don’t point to their competitors.
  - Citations to relevant literature is enforced by peer-review.
Authorities and Hubs

• **Authorities** are pages that are recognized as providing significant, trustworthy, and useful information on a topic.
  
  – *In-degree* (number of pointers to a page) is one simple measure of authority.
  
  – However in-degree treats all links as equal. Should links from pages that are themselves authoritative count more?

• **Hubs** are index pages that provide lots of useful links to relevant content pages (topic authorities).
HITS

• Algorithm developed by Kleinberg in 1998.
• Attempts to computationally determine hubs and authorities on a particular topic through analysis of a relevant subgraph of the web.
• Based on mutually recursive facts:
  – Hubs point to lots of authorities.
  – Authorities are pointed to by lots of hubs.
Hubs and Authorities

- Together they tend to form a bipartite graph:
HITS Algorithm

• Computes hubs and authorities for a particular topic specified by a normal query.
• First determines a set of relevant pages for the query called the *base* set $S$.
• Analyze the link structure of the web subgraph defined by $S$ to find authority and hub pages in this set.
Constructing a Base Subgraph

• For a specific query $Q$, let the set of documents returned by a standard search engine (e.g. VSR) be called the root set $R$.
• Initialize $S$ to $R$.
• Add to $S$ all pages pointed to by any page in $R$.
• Add to $S$ all pages that point to any page in $R$. 

![Diagram of S and R sets]
Base Limitations

• To limit computational expense:
  – Limit number of root pages to the top 200 pages retrieved for the query.
  – Limit number of “back-pointer” pages to a random set of at most 50 pages returned by a “reverse link” query.

• To eliminate purely navigational links:
  – Eliminate links between two pages on the same host.

• To eliminate “non-authority-conveying” links:
  – Allow only $m$ ($m \approx 4–8$) pages from a given host as pointers to any individual page.
Authorities and In-Degree

• Even within the base set $S$ for a given query, the nodes with highest in-degree are not necessarily authorities (may just be generally popular pages like Yahoo or Amazon).

• True authority pages are pointed to by a number of hubs (i.e. pages that point to lots of authorities).
Iterative Algorithm

• Use an iterative algorithm to slowly converge on a mutually reinforcing set of hubs and authorities.

• Maintain for each page \( p \in S \):
  – Authority score: \( a_p \) (vector \( a \))
  – Hub score: \( h_p \) (vector \( h \))

• Initialize all \( a_p = h_p = 1 \)

• Maintain normalized scores:

\[
\sum_{p \in S} \left( a_p \right)^2 = 1 \quad \sum_{p \in S} \left( h_p \right)^2 = 1
\]
HITS Update Rules

• Authorities are pointed to by lots of good hubs:

\[ a_p = \sum_{q:q \rightarrow p} h_q \]

• Hubs point to lots of good authorities:

\[ h_p = \sum_{q:p \rightarrow q} a_q \]
Illustrated Update Rules

\[ a_4 = h_1 + h_2 + h_3 \]

\[ h_4 = a_5 + a_6 + a_7 \]
HITS Iterative Algorithm

Initialize for all $p \in S$: $a_p = h_p = 1$

For $i = 1$ to $k$:

For all $p \in S$: $a_p = \sum_{q:q \rightarrow p} h_q$ \textit{(update auth. scores)}

For all $p \in S$: $h_p = \sum_{q:p \rightarrow q} a_q$ \textit{(update hub scores)}

For all $p \in S$: $a_p = a_p/c$ \textit{(normalize a)}

For all $p \in S$: $h_p = h_p/c$ \textit{(normalize h)}
HITS Example

First Iteration

Normalize: divide each vector by its norm (square root of the sum of the squares)

D    A    C    B    E
A:  [0.0, 0.0, 2.0, 2.0, 1.0]

D    A    C    B    E
H:  [4.0, 5.0, 0.0, 0.0, 0.0]

D    A    C    B    E
Norm A:  [0.0, 0.0, 0.67, 0.67, 0.33]

D    A    C    B    E
Norm H:  [0.62, 0.78, 0.0, 0.0, 0.0]
Convergence

• Algorithm converges to a *fix-point* if iterated indefinitely.

• Define $A$ to be the adjacency matrix for the subgraph defined by $S$.
  \[- A_{ij} = 1 \text{ for } i \in S, j \in S \text{ iff } i \rightarrow j \]

• Authority vector, $a$, converges to the principal eigenvector of $A^T A$

• Hub vector, $h$, converges to the principal eigenvector of $A A^T$

• In practice, 20 iterations produces fairly stable results.
HITS Results

- Authorities for query: “Java”
  - java.sun.com
  - comp.lang.java FAQ
- Authorities for query “search engine”
  - Yahoo.com
  - Excite.com
  - Lycos.com
  - Altavista.com
- Authorities for query “Gates”
  - Microsoft.com
  - roadahead.com

In most cases, the final authorities were not in the initial root set generated using Altavista. Authorities were brought in from linked and reverse-linked pages and then HITS computed their high authority score.
HITS: Other Applications

• Finding Similar Pages Using Link Structure
  – Given a page, $P$, let $R$ (the root set) be $t$ (e.g. 200) pages that point to $P$.
  – Grow a base set $S$ from $R$.
  – Run HITS on $S$.
  – Return the best authorities in $S$ as the best similar-pages for $P$.
  – Finds authorities in the “link neighbor-hood” of $P$.

Similar Pages to “honda.com”:
- toyota.com
- ford.com
- bmwusa.com
- saturncars.com
- nissanmotors.com
- audi.com
- volvocars.com
HITS: Other Applications

• HITS for Clustering
  – An ambiguous query can result in the principal eigenvector only covering one of the possible meanings.
  – Non-principal eigenvectors may contain hubs & authorities for other meanings.
  – Example: “jaguar”:
    - Atari video game (principal eigenvector)
    - NFL Football team (2nd non-princ. eigenvector)
    - Automobile (3rd non-princ. eigenvector)
  – An application of *Principle Component Analysis* (PCA)
PageRank

- Does not attempt to capture the distinction between hubs and authorities.
- Ranks pages just by authority.
- Applied to the entire Web rather than a local neighborhood of pages surrounding the results of a query.
Initial PageRank Idea

- Just measuring in-degree (citation count) doesn’t account for the authority of the source of a link.

- Initial page rank equation for page $p$:

$$R(p) = c \sum_{q:q \to p} \frac{R(q)}{N_q}$$

- $N_q$ is the total number of out-links from page $q$.
- A page, $q$, “gives” an equal fraction of its authority to all the pages it points to (e.g. $p$).
- $c$ is a normalizing constant set so that the rank of all pages always sums to 1.
Initial PageRank Idea

• Can view it as a process of PageRank “flowing” from pages to the pages they cite.
Initial PageRank Algorithm

• Iterate rank-flowing process until convergence:

Let $S$ be the total set of pages.

Initialize $\forall p \in S: R(p) = 1/|S|$

Until ranks do not change (much) (convergence)

For each $p \in S$: $R'(p) = \sum_{q: q \rightarrow p} \frac{R(q)}{N_q}$

$c = 1 / \sum_{p \in S} R'(p)$

For each $p \in S$: $R(p) = cR'(p)$ (normalize)
Sample Stable Fixpoint
Problem with Initial Idea

• A group of pages that only point to themselves but are pointed to by other pages act as a “rank sink” and absorb all the rank in the system.

Rank flows into cycle and can’t get out
Introduce a “rank source” $E$ that continually replenishes the rank of each page, $p$, by a fixed amount $E(p)$.

$$R(p) = \alpha \left( \sum_{q: q \rightarrow p} \frac{R(q)}{N_q} + E(p) \right)$$
PageRank Algorithm

Let $S$ be the total set of pages.

Let $\forall p \in S: E(p) = \alpha/|S|$ (for some $0<\alpha<1$, e.g. 0.15)

Initialize $\forall p \in S: R(p) = 1/|S|$

Until ranks do not change (much) (convergence)

For each $p \in S$:

$$R'(p) = \left[ (1-\alpha) \sum_{q:q\rightarrow p} \frac{R(q)}{N_q} \right] + E(p)$$

$$c = 1/\sum_{p \in S} R'(p)$$

For each $p \in S$: $R(p) = cR'(p)$ (normalize)
PageRank Example

\[ \alpha = 0.3 \]

Initial \( R \): \([0.33, 0.33, 0.33]\)

First Iteration Only:

\[
\begin{align*}
R'(C) & : \frac{R(A)}{2} + \frac{R(B)}{1} + \frac{0.3}{3} \\
R'(B) & : \frac{R(A)}{2} + \frac{0.3}{3} \\
R'(A) & : \frac{0.3}{3}
\end{align*}
\]

\[
\begin{bmatrix}
A & C & B \\
\end{bmatrix}
\]

\( R' \): \([0.1, 0.595, 0.27]\)

Normalization factor:

\[
1/\left[ R'(A) + R'(B) + R'(C) \right] = 1/0.965
\]

\[
\begin{bmatrix}
A & C & B \\
\end{bmatrix}
\]\n
\( R \): \([0.104, 0.617, 0.28]\)
Random Surfer Model

- PageRank can be seen as modeling a “random surfer” that starts on a random page and then at each point:
  - With probability $E(p)$ randomly jumps to page $p$.
  - Otherwise, randomly follows a link on the current page.

- $R(p)$ models the probability that this random surfer will be on page $p$ at any given time.

- “E jumps” are needed to prevent the random surfer from getting “trapped” in web sinks with no outgoing links.
Speed of Convergence

• Early experiments on Google used 322 million links.
• PageRank algorithm converged (within small tolerance) in about 52 iterations.
• Number of iterations required for convergence is empirically $O(\log n)$ (where $n$ is the number of links).
• Therefore calculation is quite efficient.
Google Ranking

• Complete Google ranking includes (based on university publications prior to commercialization).
  – Vector-space similarity component.
  – Keyword proximity component.
  – HTML-tag weight component (e.g. title preference).
  – PageRank component.

• Details of current commercial ranking functions are trade secrets.
Personalized PageRank

• PageRank can be biased (personalized) by changing $E$ to a non-uniform distribution.
• Restrict “random jumps” to a set of specified relevant pages.
• For example, let $E(p) = 0$ except for one’s own home page, for which $E(p) = \alpha$
• This results in a bias towards pages that are closer in the web graph to your own homepage.
• Similar personalization can be achieved by setting $E(p)$ for only pages $p$ that are part of the user’s profile.
PageRank-Biased Spidering

- Use PageRank to direct (focus) a spider on “important” pages.

- Compute page-rank using the current set of crawled pages.

- Order the spider’s search queue based on current estimated PageRank.
Link Analysis Conclusions

• Link analysis uses information about the structure of the web graph to aid search.

• It is one of the major innovations in web search.

• It is the primary reason for Google’s success.