Web search

Web data management and distribution

Serge Abiteboul   Ioana Manolescu   Philippe Rigaux
Marie-Christine Rousset   Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

September 18, 2013

Outline

1 Web crawling
- Discovering new URLs
- Identifying duplicates
- Crawling architecture

2 Web Information Retrieval

3 Web Graph Mining

4 Conclusion

Web Crawlers

- crawlers, (Web) spiders, (Web) robots: autonomous user agents that retrieve pages from the Web
- Basics of crawling:
  1. Start from a given URL or set of URLs
  2. Retrieve and process the corresponding page
  3. Discover new URLs (cf. next slide)
  4. Repeat on each found URL
- No real termination condition (virtual unlimited number of Web pages!)
- Graph-browsing problem

  deep-first: not very adapted, possibility of being lost in robot traps

  breadth-first

  combination of both: breadth-first with limited-depth deep-first on each discovered website

Sources of new URLs

- From HTML pages:
  - hyperlinks `<a href="...">...</a>`
  - media `<img src="...">` `<embed src="...">` `<object data="...">`
  - frames `<frame src="...">` `<iframe src="...">`
  - JavaScript links `window.open("...")`
  - etc.
- Other hyperlinked content (e.g., PDF files)
- Non-hyperlinked URLs that appear anywhere on the Web (in HTML text, text files, etc.): use regular expressions to extract them
- Referrer URLs
- Sitemaps [sit08]

Scope of a crawler

- Web-scale
  - The Web is infinite! Avoid robot traps by putting depth or page number limits on each Web server
  - Focus on important pages [APC03] (cf. lecture on the Web graph)
- Web servers under a list of DNS domains: easy filtering of URLs
- A given topic: focused crawling techniques [CvdBD99, DCL+00] based on classifiers of Web page content and predictors of the interest of a link.
- The national Web (cf. public deposit, national libraries): what is this? [ACMS02]
- A given Web site: what is a Web site? [Sen05]

A word about hashing

Definition

A hash function is a deterministic mathematical function transforming objects (numbers, character strings, binary...) into fixed-size, seemingly random, numbers. The more random the transformation is, the better.

Example

Java hash function for the `String` class:

$$\sum_{i=0}^{n-1} s_i \times 31^{n-i-1} \bmod 2^{32}$$

where s_i is the (Unicode) code of character i of a string s.

Identification of duplicate Web pages

Problem

Identifying duplicates or near-duplicates on the Web to prevent multiple indexing

trivial duplicates: same resource at the same canonized URL:

```
http://example.com:80/toto
http://example.com/titi/../toto
```

exact duplicates: identification by hashing

near-duplicates: (timestamps, tip of the day, etc.) more complex!

Near-duplicate detection

Edit distance. Count the minimum number of basic modifications (additions or deletions of characters or words, etc.) to obtain a document from another one. Good measure of similarity, and can be computed in $O(mn)$ where m and n are the size of the documents. But: does not scale to a large collection of documents (unreasonable to compute the edit distance for every pair!).

Shingles. Idea: two documents similar if they mostly share the same succession of k-grams (succession of tokens of length k).

Example

I like to watch the sun set with my friend.
My friend and I like to watch the sun set.

S ={i like, like to, my friend, set with, sun set, the sun, to watch, watch the, with my}
T ={and i, friend and, i like, like to, my friend, sun set, the sun, to watch, watch the}

Hashing shingles to detect duplicates [BGMZ97]

- Similarity: Jaccard coefficient on the set of shingles:

$$J(S, T) = \frac{|S \cap T|}{|S \cup T|}$$

- Still costly to compute! But can be approximated as follows:
  1. Choose N different hash functions
  2. For each hash function h_i and each set of shingles $S_k = \{s_{k1} \dots s_{kn}\}$, store $\phi_{ik} = \min_j h_i(s_{kj})$
  3. Approximate $J(S_k, S_l)$ as the proportion of ϕ_{ik} and ϕ_{il} that are equal
- Possibly to repeat in a hierarchical way with super-shingles (we are only interested in very similar documents)

Crawling ethics

- Standard for robot exclusion: robots.txt at the root of a Web server [Kos94].

```
User-agent: *
Allow: /searchhistory/
Disallow: /search
```

- Per-page exclusion (*de facto* standard).

```
<meta name="ROBOTS" content="NOINDEX,NOFOLLOW">
```

- Per-link exclusion (*de facto* standard).

```
<a href="toto.html" rel="nofollow">Toto</a>
```

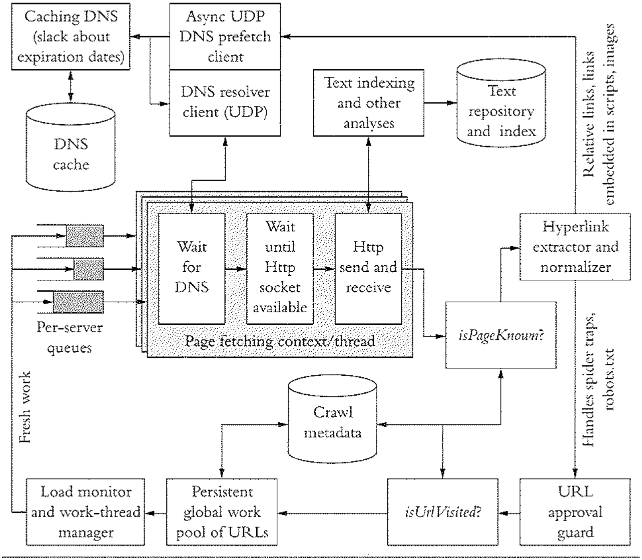
- Avoid Denial Of Service (DOS), wait 100ms/1s between two repeated requests to the same Web server

Parallel processing

Network delays, waits between requests:

- Per-server queue of URLs
- Parallel processing of requests to different hosts:
  - multi-threaded programming
  - asynchronous inputs and outputs (`select`, classes from `java.util.concurrent`): less overhead
- Use of keep-alive to reduce connexion overheads

General Architecture [Cha03]



Refreshing URLs

- Content on the Web changes
- Different change rates:

  online newspaper main page: every hour or so
  published article: virtually no change

- Continuous crawling, and identification of change rates for adaptive crawling:
  - `If-Last-Modified` HTTP feature (not reliable)
  - Identification of duplicates in successive request

Outline

1 Web crawling

2 Web Information Retrieval
- Text Preprocessing
- Inverted Index
- Answering Keyword Queries

3 Web Graph Mining

4 Conclusion

Information Retrieval, Search

Problem

How to index Web content so as to answer (keyword-based) queries efficiently?

Context: set of text documents

- d_1 The jaguar is a New World mammal of the Felidae family.
- d_2 Jaguar has designed four new engines.
- d_3 For Jaguar, Atari was keen to use a 68K family device.
- d_4 The Jacksonville Jaguars are a professional US football team.
- d_5 Mac OS X Jaguar is available at a price of US $199 for Apple's new "family pack".
- d_6 One such ruling family to incorporate the jaguar into their name is Jaguar Paw.
- d_7 It is a big cat.

Text Preprocessing

Initial text preprocessing steps

- Number of optional steps
- Highly depends on the application
- Highly depends on the document language (illustrated with English)

Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- Unambiguous scripts or letters: not very common!

한글
カタカナ
ދިވެހި
Għarbi
þorn

Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- Unambiguous scripts or letters: not very common!

  한글
  カタカナ
  ދިވެހި
  Għarbi
  þorn

  Respectively: Korean Hangul, Japanese Katakana, Maldivian Dhivehi, Maltese, Icelandic
- Extension of this: frequent characters, or, better, frequent *k*-grams
- Use standard machine learning techniques (classifiers)

Tokenization

Principle

Separate text into tokens (words)

Not so easy!

- In some languages (Chinese, Japanese), words not separated by whitespace
- Deal consistently with acronyms, elisions, numbers, units, URLs, emails, etc.
- Compound words: *hostname*, *host-name* and *host name*. Break into two tokens or regroup them as one token? In any case, lexicon and linguistic analysis needed! Even more so in other languages as German.

Usually, remove punctuation and normalize case at this point

Tokenization: Example

d_1 the$_1$ jaguar$_2$ is$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_{11}$

d_2 jaguar$_1$ has$_2$ designed$_3$ four$_4$ new$_5$ engines$_6$

d_3 for$_1$ jaguar$_2$ atari$_3$ was$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$

d_4 the$_1$ jacksonville$_2$ jaguars$_3$ are$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$

d_5 mac$_1$ os$_2$ x$_3$ jaguar$_4$ is$_5$ available$_6$ at$_7$ a$_8$ price$_9$ of$_{10}$ us$_{11}$ \$199$_{12}$ for$_{13}$ apple's$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$

d_6 one$_1$ such$_2$ ruling$_3$ family$_4$ to$_5$ incorporate$_6$ the$_7$ jaguar$_8$ into$_9$ their$_{10}$ name$_{11}$ is$_{12}$ jaguar$_{13}$ paw$_{14}$

d_7 it$_1$ is$_2$ a$_3$ big$_4$ cat$_5$

Stemming

Principle

Merge different forms of the same word, or of closely related words, into a single stem

- Not in all applications!
- Useful for retrieving documents containing *geese* when searching for *goose*
- Various degrees of stemming
- Possibility of building different indexes, with different stemming

Stemming schemes (1/2)

Morphological stemming.

- Remove bound morphemes from words:
  - plural markers
  - gender markers
  - tense or mood inflections
  - etc.
- Can be linguistically very complex, cf:
  Les poules du couvent couvent.
  [The hens of the monastery brood.]
- In English, somewhat easy:
  - Remove final -s, -'s, -ed, -ing, -er, -est
  - Take care of semiregular forms (e.g., -y/-ies)
  - Take care of irregular forms (mouse/mice)
- But still some ambiguities: cf stocking, rose

Stemming schemes (2/2)

Lexical stemming.

- Merge lexically related terms of various parts of speech, such as *policy*, *politics*, *political* or *politician*
- For English, Porter's stemming [Por80]; stem *university* and *universal* to *univers*: not perfect!
- Possibility of coupling this with lexicons to merge (near-)synonyms

Phonetic stemming.

- Merge phonetically related words: search despite spelling errors!
- For English, Soundex [US 07] stems *Robert* and *Rupert* to *R163*. Very coarse!

Stemming Example

- d_1 the$_1$ jaguar$_2$ be$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_{11}$
- d_2 jaguar$_1$ have$_2$ design$_3$ four$_4$ new$_5$ engine$_6$
- d_3 for$_1$ jaguar$_2$ atari$_3$ be$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$
- d_4 the$_1$ jacksonville$_2$ jaguar$_3$ be$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$
- d_5 mac$_1$ os$_2$ x$_3$ jaguar$_4$ be$_5$ available$_6$ at$_7$ a$_8$ price$_9$ of$_{10}$ us$_{11}$ \$199$_{12}$ for$_{13}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$
- d_6 one$_1$ such$_2$ rule$_3$ family$_4$ to$_5$ incorporate$_6$ the$_7$ jaguar$_8$ into$_9$ their$_{10}$ name$_{11}$ be$_{12}$ jaguar$_{13}$ paw$_{14}$
- d_7 it$_1$ be$_2$ a$_3$ big$_4$ cat$_5$

Stop Word Removal

Principle

Remove uninformative words from documents, in particular to lower the cost of storing the index

- determiners: *a*, *the*, *this*, etc.
- function verbs: *be*, *have*, *make*, etc.
- conjunctions: *that*, *and*, etc.
- etc.

Stop Word Removal Example

d_1 jaguar$_2$ new$_5$ world$_6$ mammal$_7$ felidae$_{10}$ family$_{11}$

d_2 jaguar$_1$ design$_3$ four$_4$ new$_5$ engine$_6$

d_3 jaguar$_2$ atari$_3$ keen$_5$ 68k$_9$ family$_{10}$ device$_{11}$

d_4 jacksonville$_2$ jaguar$_3$ professional$_6$ us$_7$ football$_8$ team$_9$

d_5 mac$_1$ os$_2$ x$_3$ jaguar$_4$ available$_6$ price$_9$ us$_{11}$ \$199$_{12}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$

d_6 one$_1$ such$_2$ rule$_3$ family$_4$ incorporate$_6$ jaguar$_8$ their$_{10}$ name$_{11}$ jaguar$_{13}$ paw$_{14}$

d_7 big$_4$ cat$_5$

Inverted Index construction

After all preprocessing, construction of an inverted index:

- Index of all terms, with the list of documents where this term occurs
- Small scale: disk storage, with memory mapping (cf. `mmap`) techniques; secondary index for offset of each term in main index
- Large scale: distributed on a cluster of machines; hashing gives the machine responsible for a given term
- Updating the index is costly, so only batch operations (not one-by-one addition of term occurrences)

Inverted Index Example

family	d_1, d_3, d_5, d_6
football	d_4
jaguar	$d_1, d_2, d_3, d_4, d_5, d_6$
new	d_1, d_2, d_5
rule	d_6
us	d_4, d_5
world	d_1
...	

Note:

- the length of an inverted (posting) list is highly variable – scanning short lists first is an important optimization.
- *entries* are homogeneous: this gives much room for compression.

Storing positions in the index

- phrase queries, NEAR operator: need to keep position information in the index
- just add it in the document list!

family	$d_1\,/\,11,\ d_3\,/\,10,\ d_5\,/\,16,\ d_6\,/\,4$
football	$d_4\,/\,8$
jaguar	$d_1\,/\,2,\ d_2\,/\,1,\ d_3\,/\,2,\ d_4\,/\,3,\ d_5\,/\,4,\ d_6\,/\,8+13$
new	$d_1\,/\,5,\ d_2\,/\,5,\ d_5\,/\,15$
rule	$d_6\,/\,3$
us	$d_4\,/\,7,\ d_5\,/\,11$
world	$d_1\,/\,6$
...	

$\Rightarrow$ so far, ok for Boolean queries: find the documents that contain a set of keywords; reject the other.

TF-IDF Weighting

The inverted is extended by adding Term Frequency—Inverse Document Frequency weighting

$$\text{tfidf}(t,d) = \frac{n_{t,d}}{\sum_{t'} n_{t',d}} \cdot \log \frac{|D|}{\left|\{d' \in D \,\middle|\, n_{t,d'} > 0\}\right|}$$

$n_{t,d}$	number of occurrences of t in d
D	set of all documents

Documents (along with weight) are stored in decreasing weight order in the index

TF-IDF Weighting Example

family	$d_1\,/\,11\,/\,.13$, $d_3\,/\,10\,/\,.13$, $d_6\,/\,4\,/\,.08$, $d_5\,/\,16\,/\,.07$
football	$d_4\,/\,8\,/\,.47$
jaguar	$d_1\,/\,2\,/\,.04$, $d_2\,/\,1\,/\,.04$, $d_3\,/\,2\,/\,.04$, $d_4\,/\,3\,/\,.04$, $d_6\,/\,8+13\,/\,.04$, $d_5\,/\,4\,/\,.02$
new	$d_2\,/\,5\,/\,.24$, $d_1\,/\,5\,/\,.20$, $d_5\,/\,15\,/\,.10$
rule	$d_6\,/\,3\,/\,.28$
us	$d_4\,/\,7\,/\,.30$, $d_5\,/\,11\,/\,.15$
world	$d_1\,/\,6\,/\,.47$
...	

Exercise: take an entry, and check that the tf/idf value is indeed correct (take documents after stop-word removal).

Answering Boolean Queries

- Single keyword query: just consult the index and return the documents in index order.
- Boolean multi-keyword query

  (*jaguar* AND *new* AND NOT *family*) OR *cat*

  Same way! Retrieve document lists from all keywords and apply adequate set operations:

AND	intersection
OR	union
AND NOT	difference

- Global score: some function of the individual weight (e.g., addition for conjunctive queries)
- Position queries: consult the index, and filter by appropriate condition

Answering Top-k Queries

$$t_1 \text{ AND } \dots \text{ AND } t_n$$

Problem

Find the top-k results (for some given k) to the query, without retrieving all documents matching it.

Notations:

$s(t,d)$	weight of t in d (e.g., tfidf)
$g(s_1,\dots,s_n)$	monotonous function that computes the global score (e.g., addition)

The Threshold Algorithm

1. Let R be the empty list, and $m = +\infty$.
2. For each $1 \leq i \leq n$:
    1. Retrieve the document $d^{(i)}$ containing term t_i that has the next largest $s(t_i, d^{(i)})$.
    2. Compute its global score $g_{d^{(i)}} = g(s(t_1, d^{(i)}), \ldots, s(t_n, d^{(i)}))$ by retrieving all $s(t_j, d^{(i)})$ with $j \neq i$.
    3. If R contains less than k documents, or if $g_{d^{(i)}}$ is greater than the minimum of the score of documents in R, add $d^{(i)}$ to R.
3. Let $m = g(s(t_1, d^{(1)}), s(t_2, d^{(2)}), \ldots, s(t_n, d^{(n)}))$.
4. If R contains more than k documents, and the minimum of the score of the documents in R is greater than or equal to m, return R.
5. Redo step 2.

The TA, by example

q = "*new* OR *family*", and $k = 3$. We use inverted lists sorted on the weight.

family $d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$

new $d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$

...

Initially, $R = \varnothing$ and $\tau = +\infty$.

1. $d^{(1)}$ is the first entry in L_{family}, one finds $s(\text{new}, \text{d}_1) = .20$; the global score for d_1 is $.13 + .20 = .33$.
2. Next, $i = 2$, and one finds that the global score for d_2 is .24.
3. The algorithm quits the loop on i with $R = \langle [d_1, .33], [d_2, .24] \rangle$ and $\tau = .13 + .24 = .37$.
4. We proceed with the loop again, taking d_3 with score .13 and d_5 with score .17. $[d_5, .17]$ is added to R (at the end) and τ is now $.10 + .13 = .23$. A last loop concludes that the next candidate is d_6, with a global score of .08. Then we are done.

Outline

1 Web crawling

2 Web Information Retrieval

3 Web Graph Mining
- PageRank
- HITS
- Spamdexing

4 Conclusion

The Web Graph

The World Wide Web seen as a (directed) graph:

Vertices: Web pages

Edges: hyperlinks

Same for other interlinked environments:

- dictionaries
- encyclopedias
- scientific publications
- social networks

The transition matrix

$$\begin{cases} g_{ij} = 0 & \text{if there is no link between page } i \text{ and } j; \\ g_{ij} = \frac{1}{n_i} & \text{otherwise, with } n_i \text{ the number of outgoing links of page } i. \end{cases}$$

$$G = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

PageRank (Google's Ranking [BP98])

Idea

Important pages are pages pointed to by important pages.

PageRank simulates a random walk by iterately computing the PR of each page, represented as a vector v.

Initially, v is set using a uniform distribution ($v[i] = \frac{1}{|v|}$).

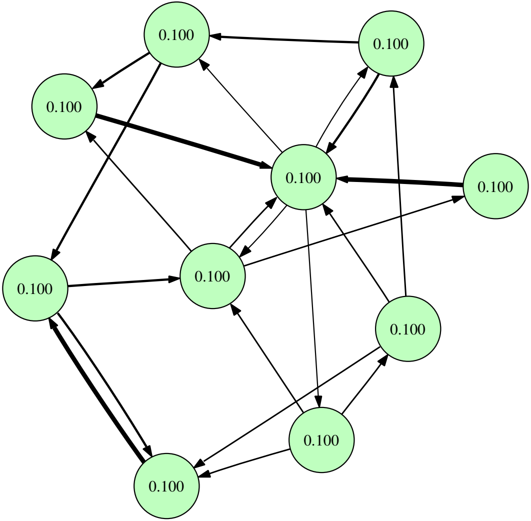
Definition (Tentative)

Probability that the surfer following the random walk in G has arrived on page i at some distant given point in the future.

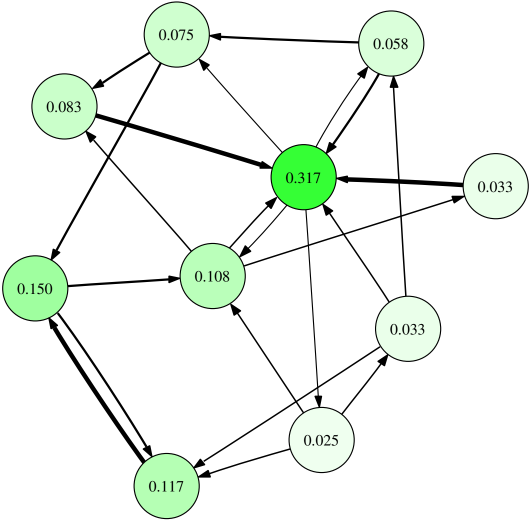
$$\mathrm{pr}(i) = \left(\lim_{k \to +\infty} (G^T)^k v \right)_i$$

where v is some initial column vector.

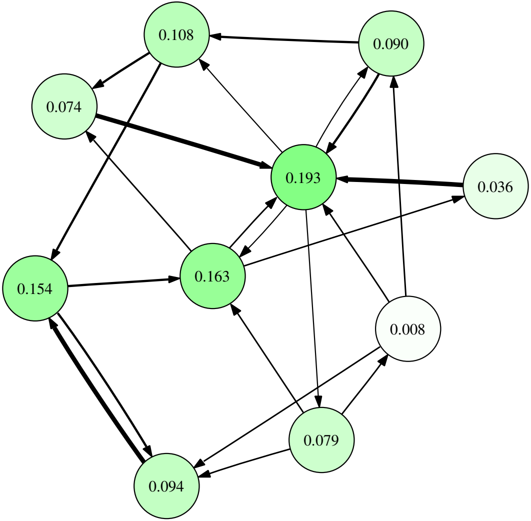
PageRank Iterative Computation



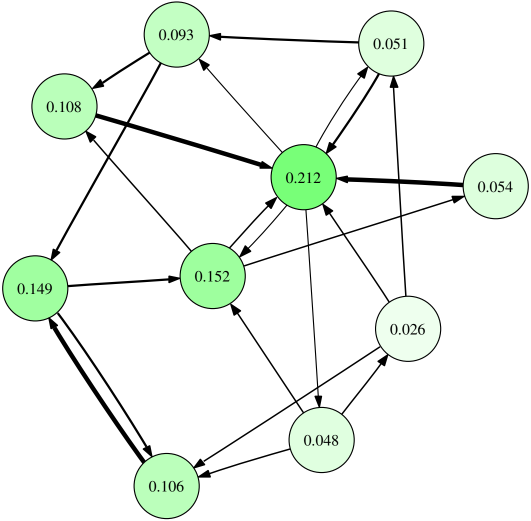
PageRank Iterative Computation



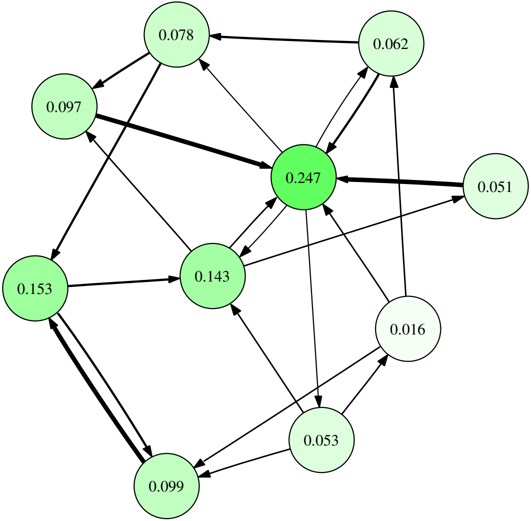
PageRank Iterative Computation



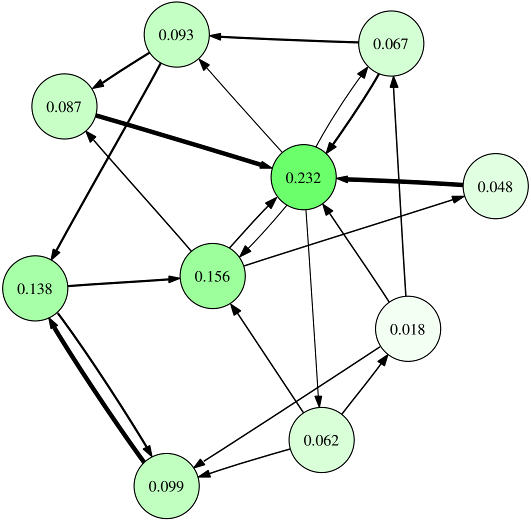
PageRank Iterative Computation



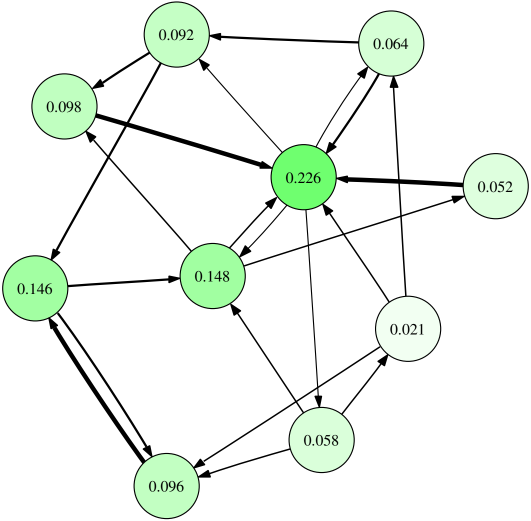
PageRank Iterative Computation



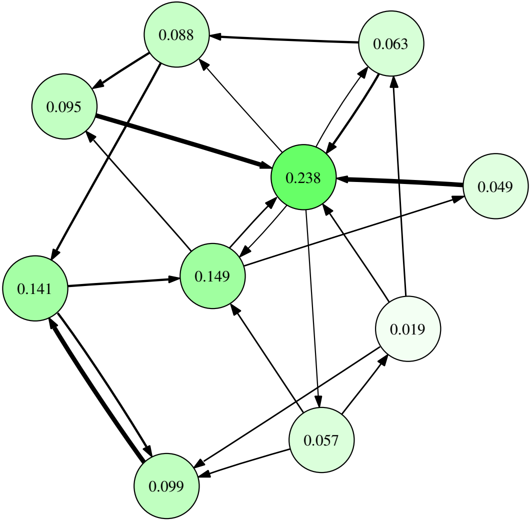
PageRank Iterative Computation



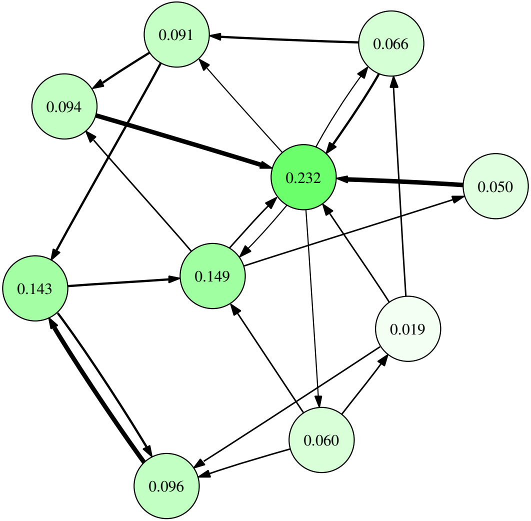
PageRank Iterative Computation



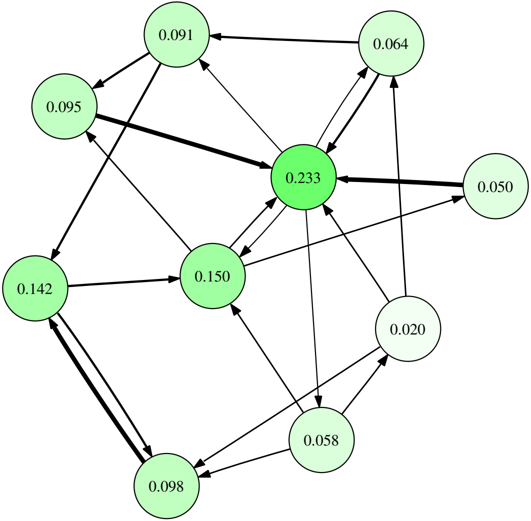
PageRank Iterative Computation



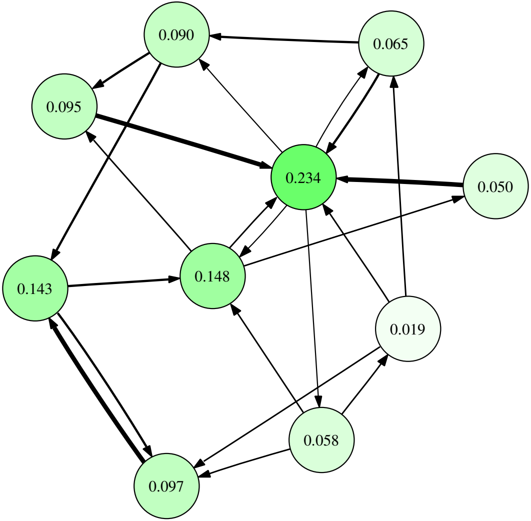
PageRank Iterative Computation



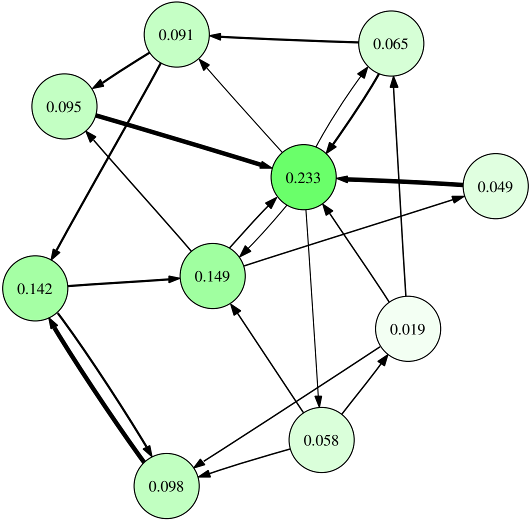
PageRank Iterative Computation



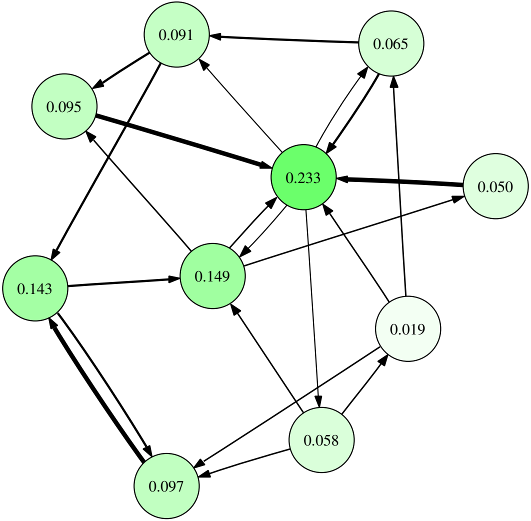
PageRank Iterative Computation



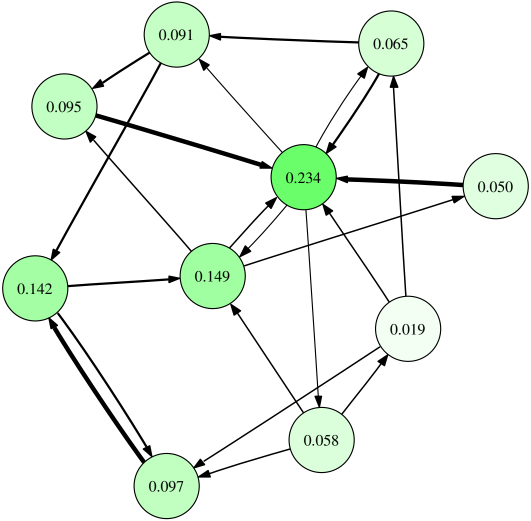
PageRank Iterative Computation



PageRank Iterative Computation



PageRank Iterative Computation



PageRank With Damping

May not always converge, or convergence may not be unique.

To fix this, the random surfer can at each step randomly jump to any page of the Web with some probability d ($1 - d$: damping factor).

$$\mathrm{pr}(i) = \left(\lim_{k \to +\infty} ((1-d)G^T + dU)^k v \right)_i$$

where U is the matrix with all $\frac{1}{N}$ values with N the number of vertices.

Using PageRank to Score Query Results

- PageRank: global score, independent of the query
- Can be used to raise the weight of important pages:

$$\text{weight}(t, d) = \text{tfidf}(t, d) \times \text{pr}(d),$$

- This can be directly incorporated in the index.

HITS (Kleinberg, [Kle99])

Idea

Two kinds of important pages: hubs and authorities. Hubs are pages that point to good authorities, whereas authorities are pages that are pointed to by good hubs.

G' transition matrix (with 0 and 1 values) of a subgraph of the Web. We use the following iterative process (starting with a and h vectors of norm 1):

$$\begin{cases} a := \frac{1}{\|G'^T h\|} G'^T h \\ h := \frac{1}{\|G' a\|} G' a \end{cases}$$

Converges under some technical assumptions to authority and hub scores.

Using HITS to Order Web Query Results

1. Retrieve the set D of Web pages matching a keyword query.
2. Retrieve the set D^* of Web pages obtained from D by adding all linked pages, as well as all pages linking to pages of D.
3. Build from D^* the corresponding subgraph G' of the Web graph.
4. Compute iteratively hubs and authority scores.
5. Sort documents from D by authority scores.

Less efficient than PageRank, because local scores.

Spamdexing

Definition

Fraudulent techniques that are used by unscrupulous webmasters to artificially raise the visibility of their website to users of search engines

Purpose: attracting visitors to websites to make profit.

Unceasing war between spamdexers and search engines

Spamdexing: Lying about the Content

Technique

Put unrelated terms in:

- meta-information (`<meta name="description">`, `<meta name="keywords">`)
- text content hidden to the user with JavaScript, CSS, or HTML presentational elements

Countertechnique

- Ignore meta-information
- Try and detect invisible text

Link Farm Attacks

Technique

Huge number of hosts on the Internet used for the sole purpose of referencing each other, without any content in themselves, to raise the importance of a given website or set of websites.

Countertechnique

- Detection of websites with empty or duplicate content
- Use of heuristics to discover subgraphs that look like link farms

Link Pollution

Technique

Pollute user-editable websites (blogs, wikis) or exploit security bugs to add artificial links to websites, in order to raise its importance.

Countertechnique

`rel="nofollow"` attribute to <a> links not validated by a page's owner

Outline

1. Web crawling
2. Web Information Retrieval
3. Web Graph Mining
4. Conclusion

What you should remember

- The inverted index model for efficient answers of keyword-based queries.
- The threshold algorithm for retrieving top-k results.
- PageRank and its iterative computation.

References

- Specifications
  - HTML 4.01, `http://www.w3.org/TR/REC-html40/`
  - HTTP/1.1, `http://tools.ietf.org/html/rfc2616`
  - *Robot Exclusion Protocol*, `http://www.robotstxt.org/orig.html`
- A book

  Mining the Web: Discovering Knowledge from Hypertext Data, Soumen Chakrabarti, Morgan Kaufmann

Bibliography I

Serge Abiteboul, Grégory Cobena, Julien Masanès, and Gerald Sedrati.
A first experience in archiving the French Web.
In *Proc. ECDL*, Roma, Italie, September 2002.

Serge Abiteboul, Mihai Preda, and Grégory Cobena.
Adaptive on-line page importance computation.
In *Proc. Intl. World Wide Web Conference (WWW)*, 2003.

Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig.
Syntactic clustering of the Web.
Computer Networks, 29(8-13):1157–1166, 1997.

Sergey Brin and Lawrence Page.
The anatomy of a large-scale hypertextual Web search engine.
Computer Networks, 30(1–7):107–117, April 1998.

Bibliography II

Soumen Chakrabarti.
Mining the Web: Discovering Knowledge from Hypertext Data.
Morgan Kaufmann, 2003.

Soumen Chakrabarti, Martin van den Berg, and Byron Dom.
Focused crawling: A new approach to topic-specific Web resource discovery.
Computer Networks, 31(11–16):1623–1640, 1999.

Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori.
Focused crawling using context graphs.
In *Proc. VLDB*, Cairo, Egypt, September 2000.

Jon M. Kleinberg.
Authoritative Sources in a Hyperlinked Environment.
Journal of the ACM, 46(5):604–632, 1999.

Bibliography III

Martijn Koster.
A standard for robot exclusion.
`http://www.robotstxt.org/orig.html`, June 1994.

Martin F. Porter.
An algorithm for suffix stripping.
Program, 14(3):130–137, July 1980.

Pierre Senellart.
Identifying Websites with flow simulation.
In *Proc. ICWE*, pages 124–129, Sydney, Australia, July 2005.

sitemaps.org.
Sitemaps XML format.
`http://www.sitemaps.org/protocol.php`, February 2008.

Bibliography IV

US National Archives and Records Administration.
The Soundex indexing system.
http://www.archives.gov/genealogy/census/soundex.html,
May 2007.