NPFL103: Information Retrieval (12)

Web search, Crawling, Spam detection

Pavel Pecina

pecina@ufal.mff.cuni.cz

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University

Original slides are courtesy of Hinrich Schütze, University of Stuttgart.

Contents

Web search

Web IR

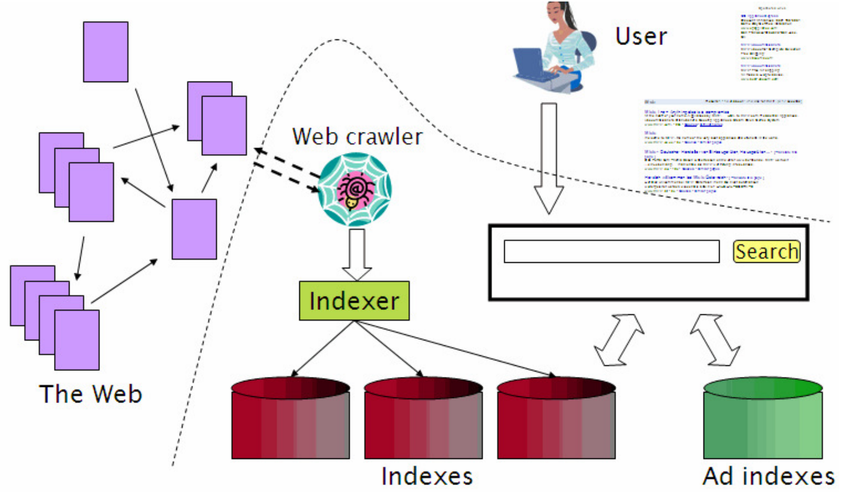
Web crawling

Duplicate detection

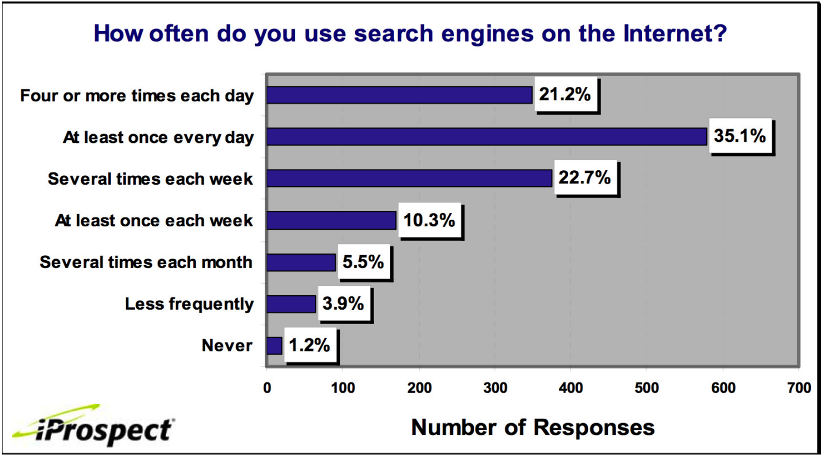
Spam detection

Web search

Web search overview



Search is a top activity on the web



Without search engines, the web wouldn't work

- Without search, content is hard to find.

→ Without search, there is no incentive to create content.
  - Why publish something if nobody will read it?
  - Why publish something if I don't get ad revenue from it?

- Somebody needs to pay for the web.
  - Servers, web infrastructure, content creation
  - A large part today is paid by search ads.
  - Search pays for the web.

IR on the web vs. IR in general

- On the web, search is not just a nice feature.
  - Search is a key enabler of the web: ...
  - ...financing, content creation, interest aggregation etc.

  → look at search ads
- The web is a chaotic und uncoordinated collection. → lots of duplicates – need to detect duplicates
- No control / restrictions on who can author content → lots of spam – need to detect spam
- The web is very large. → need to know how big it is

Brief history of the search engine (1)

- 1995–1997: Early keyword-based search engines
  - Altavista, Excite, Infoseek, Inktomi
- Second half of 1990s: Goto.com
  - Paid placement ranking
  - The highest bidder for a particular query gets the top rank.
  - The second highest bidder gets rank 2 etc.
  - This was the only match criterion!
  - ...if there were enough bidders.

Brief history of the search engine (2)

- Starting in 1998/1999: Google
  - Blew away most existing search engines at the time
  - Link-based ranking was perhaps the most important differentiator.
  - But there were other innovations: super-simple UI, tiered index, proximity search etc.
  - Initially: zero revenue!
- Beginning around 2001: Second generation search ads
  - Strict separation of search results and search ads
  - The main source of revenue today

Web IR

Web IR: Differences from traditional IR

- Links: The web is a hyperlinked document collection.
- Queries: Web queries are different, more varied and there are a lot of them. How many? $\approx 10^9$
- Users: Users are different, more varied and there are a lot of them. How many? $\approx 10^9$
- Documents: Documents are different, more varied and there are a lot of them. How many? $\approx 10^{11}$
- Context: Context is more important on the web than in many other IR applications.
- Ads and spam

Query distribution (1)

Most frequent queries on a large search engine on 2002.10.26.

1	sex	16	crack	31	juegos	46	Caramail
2	(artifact)	17	games	32	nude	47	msn
3	(artifact)	18	pussy	33	music	48	jennifer lopez
4	porno	19	cracks	34	musica	49	tits
5	mp3	20	lolita	35	anal	50	free porn
6	Halloween	21	britney spears	36	free6	51	cheats
7	sexo	22	ebay	37	avril lavigne	52	yahoo.com
8	chat	23	sexe	38	hotmail.com	53	eminem
9	porn	24	Pamela Anderson	39	winzip	54	Christina Aguilera
10	yahoo	25	warez	40	fuck	55	incest
11	KaZaA	26	divx	41	wallpaper	56	letras de canciones
12	xxx	27	gay	42	hotmail.com	57	hardcore
13	Hentai	28	harry potter	43	postales	58	weather
14	lyrics	29	playboy	44	shakira	59	wallpapers
15	hotmail	30	lolitas	45	traductor	60	lingerie

More than 1/3 of these are queries for adult content.

Exercise: Does this mean that most people are looking for adult content?

Query distribution (2)

- Queries have a power law distribution.
- Recall Zipf's law: a few very frequent words, a large number of very rare words
- Same here: a few very frequent queries, a large number of very rare queries
- Examples of rare queries: search for names, towns, books etc
- The proportion of adult queries is much lower than 1/3

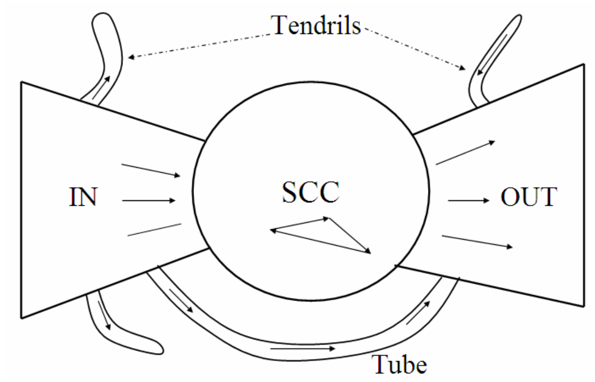
Types of queries / user needs in web search

- Informational user needs: I need information on something. "low hemoglobin"
- We called this "information need" earlier in the class.
- On the web, information needs are only a subclass of user needs.
- Other user needs: Navigational and transactional
- Navigational user needs: I want to go to this web site. "hotmail", "facebook", "United Airlines"
- Transactional user needs: I want to make a transaction.
  - Buy something: "MacBook Air"
  - Download something: "Acrobat Reader"
  - Chat with someone: "live soccer chat"
- Difficult problem: How can the search engine tell what the user need or intent for a particular query is?

Search in a hyperlinked collection

- Web search in most cases is interleaved with navigation ...
- ...i.e., with following links.
- Different from most other IR collections

Bowtie structure of the web



- Strongly connected component (SCC) in the center
- Lots of pages that get linked to, but don't link (OUT)
- Lots of pages that link to other pages, but don't get linked to (IN)
- Tendrils, tubes, islands

User intent: Answering the need behind the query

- What can we do to guess user intent?
- Guess user intent independent of context:
  - Spell correction
  - Precomputed "typing" of queries (next slide)
- Better: Guess user intent based on context:
  - Geographic context (slide after next)
  - Context of user in this session (e.g., previous query)
  - Context provided by personal profile (Yahoo/MSN do this, Google claims it doesn't)

Guessing of user intent by "typing" queries

- Calculation: 5+4
- Unit conversion: 1 kg in pounds
- Currency conversion: 1 euro in kronor
- Tracking number: 8167 2278 6764
- Flight info: LH 454
- Area code: 650
- Map: columbus oh
- Stock price: msft
- Albums/movies etc: coldplay

The spatial context: Geo-search

- Three relevant locations
  - Server (nytimes.com → New York)
  - Web page (nytimes.com article about Albania)
  - User (located in Palo Alto)
- Locating the user
  - IP address
  - Information provided by user (e.g., in user profile)
  - Mobile phone
- Geo-tagging: Parse text and identify the coordinates of the geographic entities
  - Example: East Palo Alto CA → Latitude: 37.47 N, Longitude: 122.14 W
  - Important NLP problem

How do we use context to modify query results?

- Result restriction: Don't consider inappropriate results
  - For user on google.fr only show .fr results, etc.
- Ranking modulation: use a rough generic ranking, rerank based on personal context
- Contextualization / personalization is an area of search with a lot of potential for improvement.

Users of web search

- Use short queries (average < 3)
- Rarely use operators
- Don't want to spend a lot of time on composing a query
- Only look at the first couple of results
- Want a simple UI, not a start page overloaded with graphics
- Extreme variability in terms of user needs, user expectations, experience, knowledge, ...
  - Industrial/developing world, English/Estonian, old/young, rich/poor, differences in culture and class
- One interface for hugely divergent needs

How do users evaluate search engines?

- Classic IR relevance (as measured by F) can also be used for web IR.
- Equally important: Trust, duplicate elimination, readability, loads fast, no pop-ups
- On the web, precision is more important than recall.
  - Precision at 1, precision at 10, precision on the first 2-3 pages
  - But there is a subset of queries where recall matters.

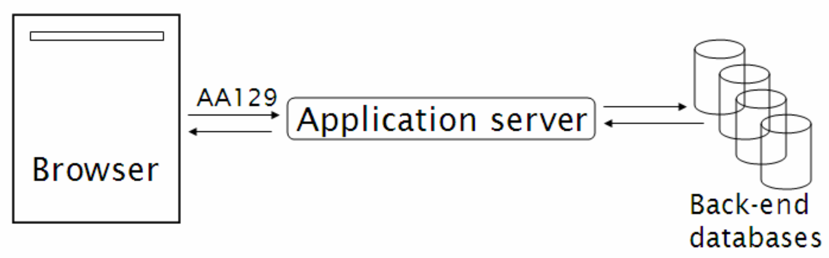
Web information needs that require high recall

- Has this idea been patented?
- Searching for info on a prospective financial advisor
- Searching for info on a prospective employee
- Searching for info on a date

Web documents: different from other IR collections

- Distributed content creation: no design, no coordination
  - "Democratization of publishing"
  - Result: extreme heterogeneity of documents on the web
- Unstructured (text, html), semistructured (html, xml), structured/relational (databases)
- Dynamically generated content

Dynamic content



- Dynamic pages are generated from scratch when the user requests them – usually from underlying data in a database.
- Example: current status of flight LH 454

Dynamic content (2)

- Most (truly) dynamic content is ignored by web spiders.
    - It's too much to index it all.
- Actually, a lot of "static" content is also assembled on the fly (asp, php etc.: headers, date, ads etc)

Multilinguality

- Documents in a large number of languages
- Queries in a large number of languages
- First cut: Don't return English results for a Japanese query
- However: Frequent mismatches query/document languages
- Many people can understand, but not query in a language
- Translation is important.
- Google example: "Beaujolais Nouveau -wine"

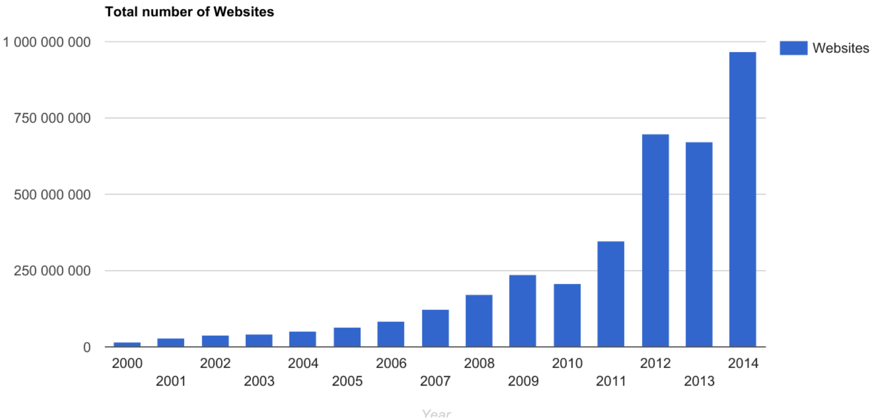
Duplicate documents

- Significant duplication – 30%–40% duplicates in some studies
- Duplicates in the search results were common in the early days of the web.
- Today's search engines eliminate duplicates very effectively.
- Key for high user satisfaction

Trust

- For many collections, it is easy to assess the trustworthiness of a document.
  - A collection of Reuters newswire articles
  - A collection of TASS (Telegraph Agency of the Soviet Union) newswire articles from the 1980s
  - Your Outlook email from the last three years
- Web documents are different: In many cases, we don't know how to evaluate the information.
- Hoaxes abound.

Growth of the web



source: internetlivestats.com

Web crawling

How hard can crawling be?

- Web search engines must crawl their documents.
- Getting the content of the documents is easier for many other IR systems.
  - E.g., indexing all files on your hard disk: just do a recursive descent on your file system
- Ok: for web IR, getting the content of the documents takes longer ...
- ...because of latency.
- But is that really a design/systems challenge?

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat:
    1. Take URL from queue
    2. Fetch and parse page
    3. Extract URLs from page
    4. Add URLs to queue
- Fundamental assumption: The web is well linked.

Exercise: What's wrong with this crawler?

```
urlqueue := (some carefully selected set of seed urls)
while urlqueue is not empty:
  myurl := urlqueue.getlastanddelete()
  mypage := myurl.fetch()
  fetchedurls.add(myurl)
  newurls := mypage.extracturls()
  for myurl in newurls:
    if myurl not in fetchedurls and not in urlqueue:
      urlqueue.add(myurl)
  addtoinvertedindex(mypage)
```

What's wrong with the simple crawler

- Scale: we need to distribute.
- We can't index everything: we need to subselect. How?
- Duplicates: need to integrate duplicate detection
- Spam and spider traps: need to integrate spam detection
- Politeness: we need to be "nice" and space out all requests for a site over a longer period (hours, days)
- Freshness: we need to recrawl periodically.
  - Because of the size of the web, we can do frequent recrawls only for a small subset.
  - Again, subselection problem or prioritization

Magnitude of the crawling problem

- To fetch 20,000,000,000 pages in one month ...
- ...we need to fetch almost 8000 pages per second!
- Actually: many more since many of the pages we attempt to crawl will be duplicates, unfetchable, spam etc.

What a crawler must do

Be polite

- Don't hit a site too often
- Only crawl pages you are allowed to crawl: robots.txt

Be robust

- Be immune to spider traps, duplicates, very large pages, very large websites, dynamic pages etc

Robots.txt

- Protocol for giving crawlers ("robots") limited access to a website, originally from 1994
- Examples:
  - User-agent: *
    Disallow: /yoursite/temp/
  - User-agent: searchengine
    Disallow: /
- Important: cache the robots.txt file of each site we are crawling

Example of a robots.txt (nih.gov)

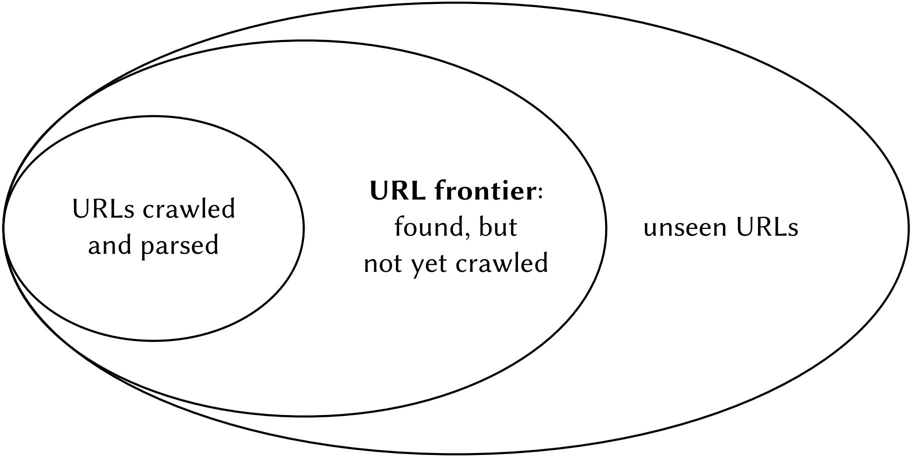
```
User-agent: PicoSearch/1.0
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/

User-agent: *
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/
Disallow: /ddir/
Disallow: /sdminutes/
```

What any crawler should do

- Be capable of distributed operation
- Be scalable: need to be able to increase crawl rate by adding more machines
- Fetch pages of higher quality first
- Continuous operation: get fresh version of already crawled pages

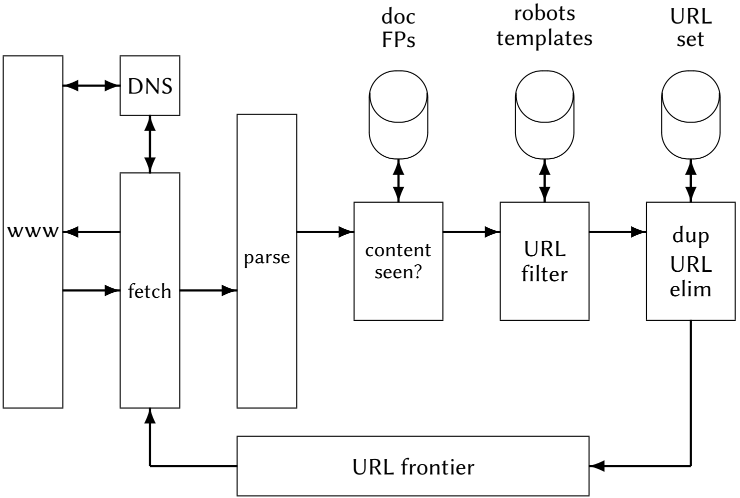
URL frontier



URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.
- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must keep all crawling threads busy

Basic crawl architecture



URL normalization

- Some URLs extracted from a document are relative URLs.
- E.g., at http://mit.edu, we may have aboutsite.html
  - This is the same as: http://mit.edu/aboutsite.html
- During parsing, we must normalize (expand) all relative URLs.

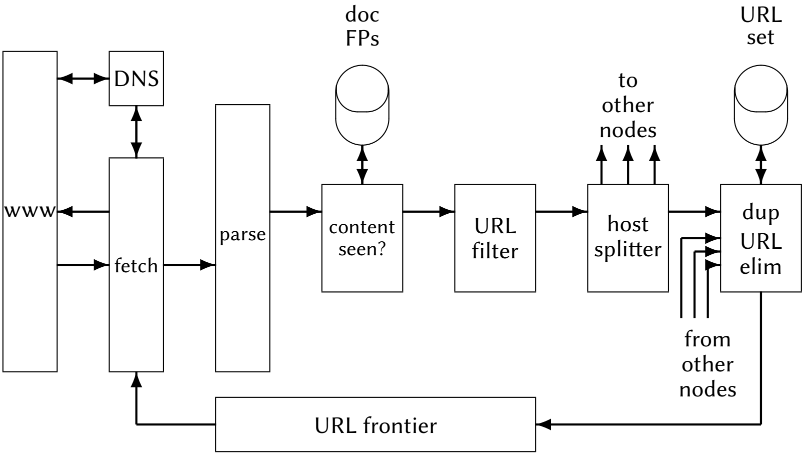
Content seen

- For each page fetched: check if the content is already in the index
- Check this using document fingerprints or shingles
- Skip documents whose content has already been indexed

Distributing the crawler

- Run multiple crawl threads, potentially at different nodes
  - Usually geographically distributed nodes
- Partition hosts being crawled into nodes

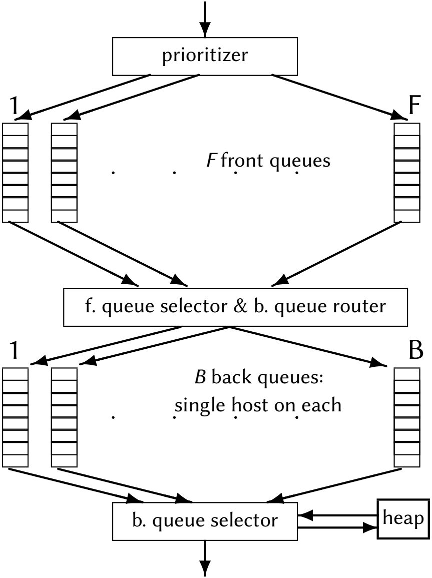
Distributed crawler



URL frontier: Two main considerations

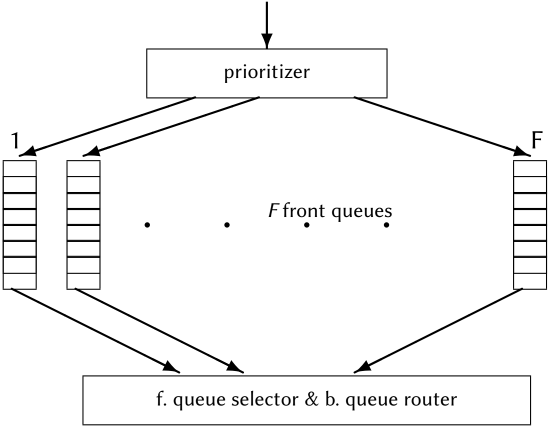
- Politeness: Don't hit a web server too frequently
  - E.g., insert a time gap between successive requests to the same server
- Freshness: Crawl some pages (e.g., news sites) more often than others
- Not an easy problem: simple priority queue fails.

Mercator URL frontier



- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.
- Each queue is FIFO.

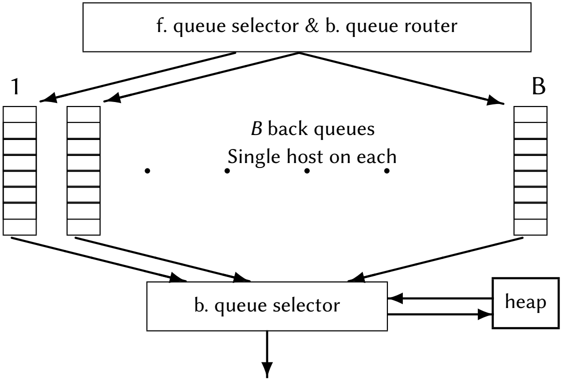
Mercator URL frontier: Front queues



- Prioritizer assigns to URL an integer priority between 1 and F.
- Then appends URL to corresponding queue
- Heuristics for assigning priority: refresh rate, PageRank etc

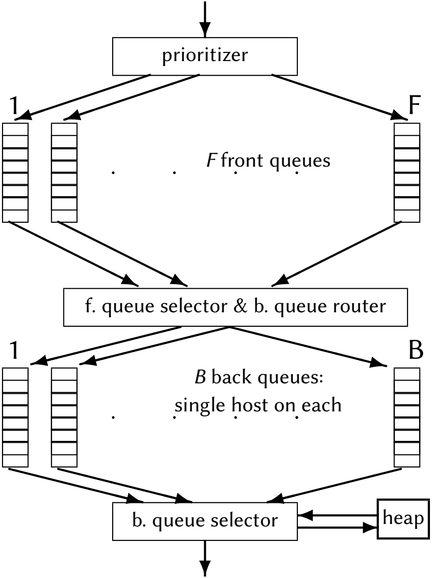
- Selection from front queues is initiated by back queues
- Pick a front queue from which to select next URL: Round robin, randomly, or more sophisticated variant

Mercator URL frontier: Back queues



- Invariant 1. Each back queue is kept non-empty while the crawl is in progress.
- Invariant 2. Each back queue only contains URLs from a single host.
- Maintain a table from hosts to back queues.
- In the heap:
- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again.

Mercator URL frontier



- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.
- Each queue is FIFO.

Spider trap

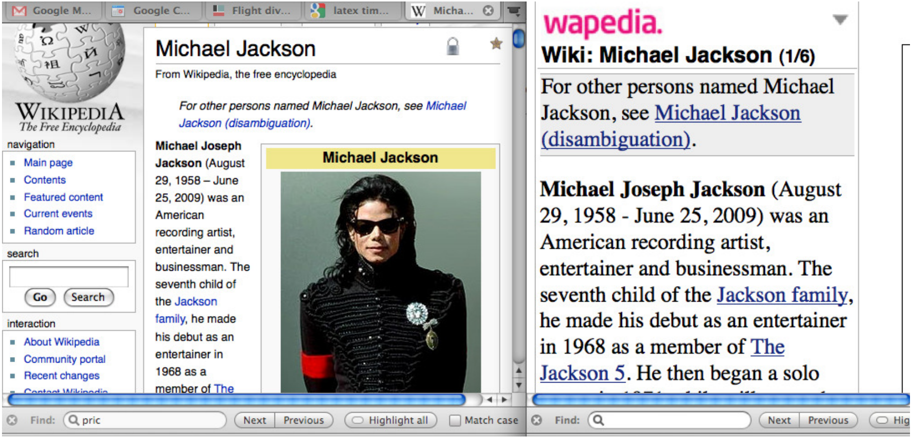
- Malicious server that generates an infinite sequence of linked pages
- Sophisticated spider traps generate pages that are not easily identified as dynamic.

Duplicate detection

Duplicate detection

- The web is full of duplicated content.
- More so than many other collections
- Exact duplicates
  - Easy to eliminate
  - E.g., use hash/fingerprint
- Near-duplicates
  - Abundant on the web
  - Difficult to eliminate
- For the user, it's annoying to get a search result with near-identical documents.
- Marginal relevance is zero: even a highly relevant document becomes nonrelevant if it appears below a (near-)duplicate.

Near-duplicates: Example



Detecting near-duplicates

- Compute similarity with an edit-distance measure
- We want "syntactic" (as opposed to semantic) similarity.
  - True semantic similarity (similarity in content) is too difficult to compute.
- We do not consider documents near-duplicates if they have the same content, but express it with different words.
- Use similarity threshold θ to make the call "is/isn't a near-duplicate".
- E.g., two documents are near-duplicates if similarity $> \theta = 80\%$.

Represent each document as set of **shingles**

- A shingle is simply a word n-gram.
- Shingles are used as features to measure syntactic similarity of documents.
- For example, for $n = 3$, "a rose is a rose is a rose" would be represented as this set of shingles:
  - { a-rose-is, rose-is-a, is-a-rose }
- We can map shingles to $1..2^m$ (e.g., $m = 64$) by fingerprinting.
- From now on: s_k refers to the shingle's fingerprint in $1..2^m$.
- We define the similarity of two documents as the Jaccard coefficient of their shingle sets.

Recall: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

$$\textsc{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

  $(A \neq \emptyset$ or $B \neq \emptyset)$
- $\textsc{jaccard}(A, A) = 1$
- $\textsc{jaccard}(A, B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

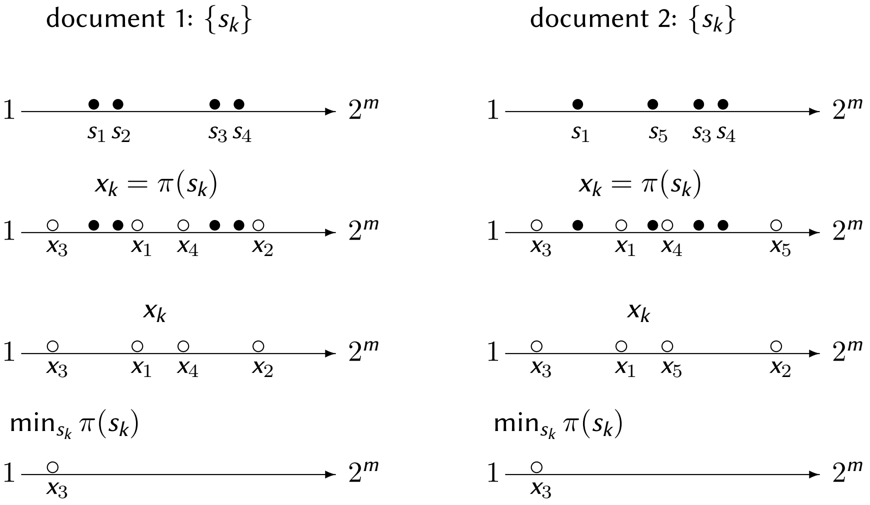
Jaccard coefficient: Example

- Three documents:
  d_1: "Jack London traveled to Oakland"
  d_2: "Jack London traveled to the city of Oakland"
  d_3: "Jack traveled from Oakland to London"
- Based on shingles of size 2 (2-grams or bigrams), what are the Jaccard coefficients $J(d_1, d_2)$ and $J(d_1, d_3)$?
- $J(d_1, d_2) = 3/8 = 0.375$
- $J(d_1, d_3) = 0$
- Note: very sensitive to dissimilarity

Represent each document as a **sketch**

- The number of shingles per document is large.
- To increase efficiency, we will use a sketch, a cleverly chosen subset of the shingles of a document.
- The size of a sketch is, say, $n = 200$ ...
- ...and is defined by a set of permutations $\pi_1 \dots \pi_{200}$.
- Each π_i is a random permutation on $1..2^m$
- The sketch of d is defined as:
  $< \min_{s \in d} \pi_1(s), \min_{s \in d} \pi_2(s), \dots, \min_{s \in d} \pi_{200}(s) >$
  (a vector of 200 numbers).

Permutation and minimum: Example



We use $\min_{s \in d_1} \pi(s) = \min_{s \in d_2} \pi(s)$ as a test for: are d_1 and d_2 near-duplicates? In this case: permutation π says: $d_1 \approx d_2$

Computing Jaccard for sketches

- Sketches: Each document is now a vector of $n = 200$ numbers.
- Much easier to deal with than the very high-dimensional space of shingles
- But how do we compute Jaccard?

Computing Jaccard for sketches (2)

- How do we compute Jaccard?
- Let U be the union of the set of shingles of d_1 and d_2 and I the intersection.
- There are $|U|!$ permutations on U.
- For $s' \in I$, for how many permutations π do we have $\arg\min_{s \in d_1} \pi(s) = s' = \arg\min_{s \in d_2} \pi(s)$?
- Answer: $(|U| - 1)!$
- There is a set of $(|U| - 1)!$ different permutations for each s in I. $\Rightarrow$ $|I|(|U| - 1)!$ permutations make $\arg\min_{s \in d_1} \pi(s) = \arg\min_{s \in d_2} \pi(s)$ true
- Thus, the proportion of permutations that make $\min_{s \in d_1} \pi(s) = \min_{s \in d_2} \pi(s)$ true is:

$$\frac{|I|(|U| - 1)!}{|U|!} = \frac{|I|}{|U|} = J(d_1, d_2)$$

Estimating Jaccard

- Thus, the proportion of successful permutations is the Jaccard coefficient.
  - Permutation π is successful iff $\min_{s \in d_1} \pi(s) = \min_{s \in d_2} \pi(s)$
- Picking a permutation at random and outputting 1 (successful) or 0 (unsuccessful) is a Bernoulli trial.
- Estimator of probability of success: proportion of successes in n Bernoulli trials. ($n = 200$)
- Our sketch is based on a random selection of permutations.
- Thus, to compute Jaccard, count the number k of successful permutations for $< d_1, d_2 >$ and divide by $n = 200$.
- $k/n = k/200$ estimates $J(d_1, d_2)$.

Implementation

- We use hash functions as an efficient type of permutation: $h_i : \{1..2^m\} \rightarrow \{1..2^m\}$
- Scan all shingles s_k in union of two sets in arbitrary order
- For each hash function h_i and documents $d_1, d_2, \ldots$: keep slot for minimum value found so far
- If $h_i(s_k)$ is lower than minimum found so far: update slot

Example

	d_1	d_2
s_1	1	0
s_2	0	1
s_3	1	1
s_4	1	0
s_5	0	1

$h(x) = x \bmod 5$

$g(x) = (2x + 1) \bmod 5$

$\min(h(d_1)) = 1 \neq 0 = \min(h(d_2))$

$\min(g(d_1)) = 2 \neq 0 = \min(g(d_2))$

$\hat{J}(d_1, d_2) = \frac{0+0}{2} = 0$

	d_1 slot		d_2 slot	
h		∞		∞
g		∞		∞
$h(1) = 1$	1	1	–	∞
$g(1) = 3$	3	3	–	∞
$h(2) = 2$	–	1	2	2
$g(2) = 0$	–	3	0	0
$h(3) = 3$	3	1	3	2
$g(3) = 2$	2	2	2	0
$h(4) = 4$	4	1	–	2
$g(4) = 4$	4	2	–	0
$h(5) = 0$	–	1	0	0
$g(5) = 1$	–	2	1	0

final sketches

Exercise

	d_1	d_2	d_3
s_1	0	1	1
s_2	1	0	1
s_3	0	1	0
s_4	1	0	0

$h(x) = 5x + 5 \mod 4$
$g(x) = (3x + 1) \mod 4$

Estimate $\hat{J}(d_1, d_2), \hat{J}(d_1, d_3), \hat{J}(d_2, d_3)$

Solution (1)

	d_1	d_2	d_3
s_1	0	1	1
s_2	1	0	1
s_3	0	1	0
s_4	1	0	0

$$h(x) = 5x + 5 \mod 4$$
$$g(x) = (3x + 1) \mod 4$$

	d_1 slot		d_2 slot		d_3 slot	
		∞		∞		∞
		∞		∞		∞
$h(1) = 2$	–	∞	2	2	2	2
$g(1) = 0$	–	∞	0	0	0	0
$h(2) = 3$	3	3	–	2	3	2
$g(2) = 3$	3	3	–	0	3	0
$h(3) = 0$	–	3	0	0	–	2
$g(3) = 2$	–	3	2	0	–	0
$h(4) = 1$	1	1	–	0	–	2
$g(4) = 1$	1	1	–	0	–	0

final sketches

Solution (2)

$$\hat{J}(d_1, d_2) = \frac{0+0}{2} = 0$$
$$\hat{J}(d_1, d_3) = \frac{0+0}{2} = 0$$
$$\hat{J}(d_2, d_3) = \frac{0+1}{2} = 1/2$$

Shingling: Summary

- Input: N documents
- Choose n-gram size for shingling, e.g., $n = 5$
- Pick 200 random permutations, represented as hash functions
- Compute N sketches: $200 \times N$ matrix shown on previous slide, one row per permutation, one column per document
- Compute $\frac{N \cdot (N-1)}{2}$ pairwise similarities
- Transitive closure of documents with similarity $> \theta$
- Index only one document from each equivalence class

Efficient near-duplicate detection

- Now we have an extremely efficient method for estimating a Jaccard coefficient for a single pair of two documents.
- But we still have to estimate $O(N^2)$ coefficients where N is the number of web pages.
- Still intractable
- One solution: locality sensitive hashing (LSH)
- Another solution: sorting (Henzinger 2006)

Spam detection

The goal of spamming on the web

- You have a page that will generate lots of revenue for you if people visit it.
- Therefore, you would like to direct visitors to this page.
- One way of doing this: get your page ranked highly in search results.
- Exercise: How can I get my page ranked highly?

Spam technique: Keyword stuffing / Hidden text

- Misleading meta-tags, excessive repetition
- Hidden text with colors, style sheet tricks etc.
- Used to be very effective, most search engines now catch these

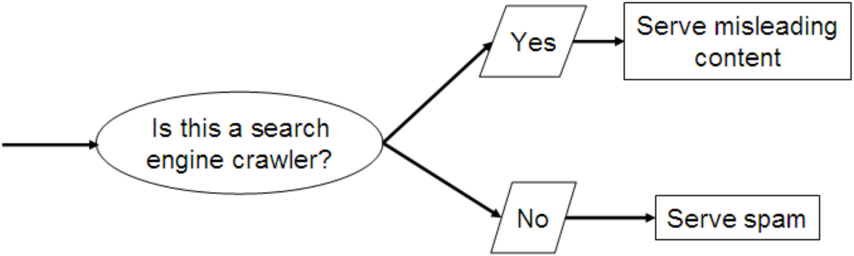
Spam technique: Doorway and lander pages

- Doorway page: optimized for a single keyword, redirects to the real target page
- Lander page: optimized for a single keyword or a misspelled domain name, designed to attract surfers who will then click on ads

Spam technique: Duplication

- Get good content from somewhere (steal it or produce it yourself)
- Publish a large number of slight variations of it
- For example, publish the answer to a question with the spelling variations

Spam technique: Cloaking



- Serve fake content to search engine spider
- So do we just penalize this always?
- No: legitimate uses (e.g., different content to US vs. European users)

Spam technique: Link spam

- Create lots of links pointing to the page you want to promote
- Put these links on pages with high (or at least non-zero) PageRank
  - Newly registered domains (domain flooding)
  - A set of pages that all point to each other to boost each other's PageRank (mutual admiration society)
  - Pay somebody to put your link on their highly ranked page
  - Leave comments that include the link on blogs

SEO: Search engine optimization

- Promoting a page in the search rankings is not necessarily spam.
- It can also be a legitimate business – which is called SEO.
- You can hire an SEO firm to get your page highly ranked.
- There are many legitimate reasons for doing this
  - For example, Google bombs like *Who is a failure?*
- And there are many legitimate ways of achieving this:
  - Restructure your content in a way that makes it easy to index
  - Talk with influential bloggers and have them link to your site
  - Add more interesting and original content

The war against spam

- Quality indicators
  - Links, statistically analyzed (PageRank etc)
  - Usage (users visiting a page)
  - No adult content (e.g., no pictures with flesh-tone)
  - Distribution and structure of text (e.g., no keyword stuffing)
- Combine all of these indicators and use machine learning
- Editorial intervention
  - Blacklists
  - Top queries audited
  - Complaints addressed
  - Suspect patterns detected

Webmaster guidelines

- Major search engines have guidelines for webmasters.
- These guidelines tell you what is legitimate SEO and what is spamming.
- Ignore these guidelines at your own risk
- Once a search engine identifies you as a spammer, all pages on your site may get low ranks (or disappear from the index entirely).
- There is often a fine line between spam and legitimate SEO.
- Scientific study of fighting spam on the web: *adversarial information retrieval*