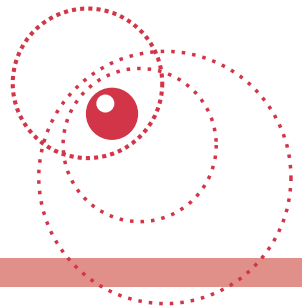


Pesto Web Mining

Web Crawling

7 October 2015

TELECOM ParisTech

- 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

- 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 (**?**)

Web-scale

- The Web is infinite! Avoid robot traps by putting depth or page number limits on each Web server
- Focus on important pages (**?**)

- Web servers under a list of DNS domains: easy filtering of URLs
- A given topic: focused crawling techniques (**??**) 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? (**?**)
- A given Web site: what is a Web site? (**?**)

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.

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!

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$\}$

- 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)

- Standard for robot exclusion: robots.txt at the root of a Web server (**?**).

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

- Per-page exclusion.

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

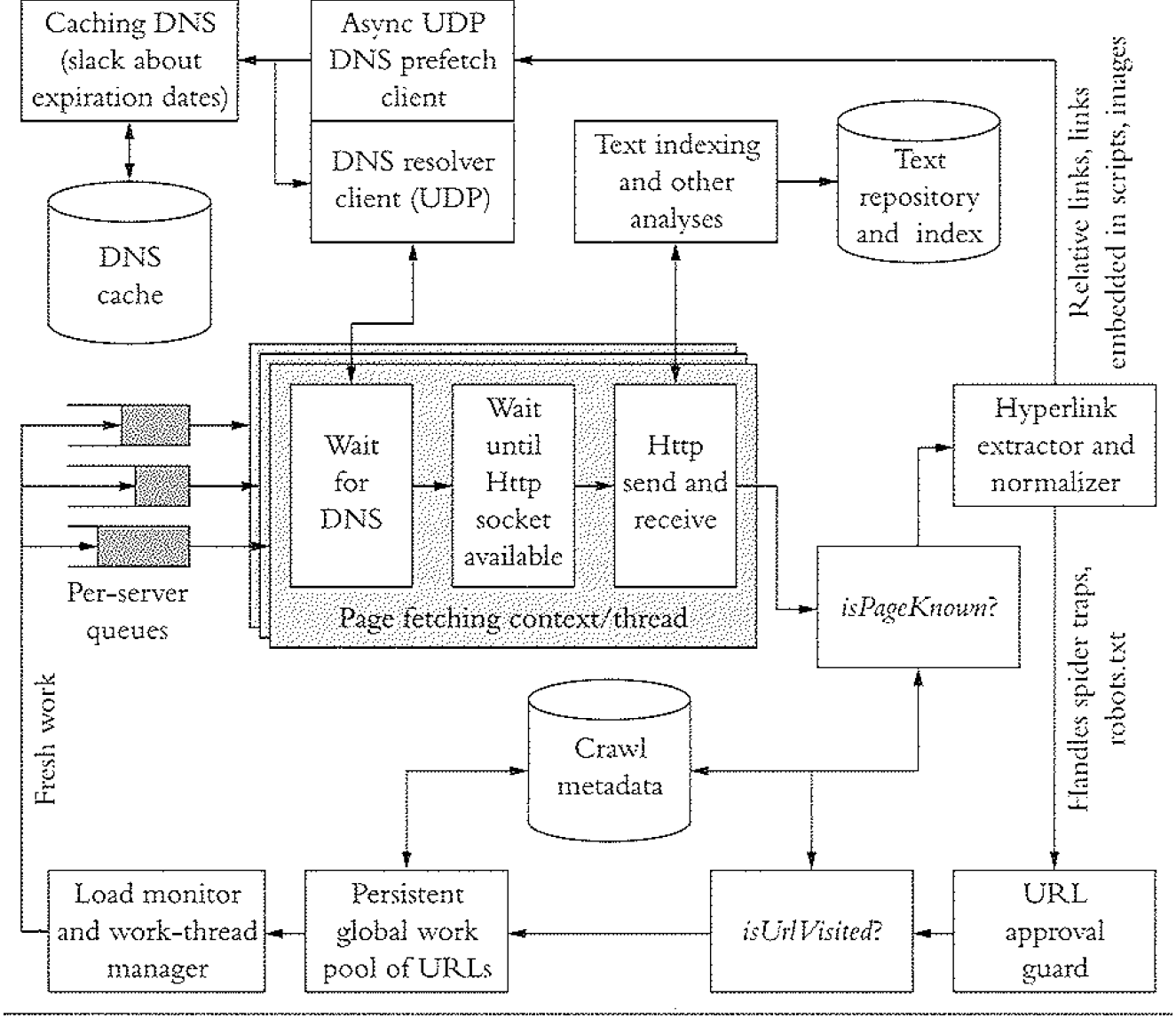
- Per-link exclusion.

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

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

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


Caching DNS (slack about expiration dates)
Async UDP DNS prefetch client
DNS resolver client (UDP)
DNS cache
Text indexing and other analyses
Text repository and index
Relative links, links embedded in scripts, images
Wait for DNS
Wait until Http socket available
Http send and receive
Page fetching context/thread
Per-server queues
Hyperlink extractor and normalizer
isPageKnown?
Handles spider traps, robots.txt
Fresh work
Crawl metadata
Load monitor and work-thread manager
Persistent global work pool of URLs
isUrlVisited?
URL approval guard

- 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: how to know the time of last modification of a Web page?

1. Check HTTP timestamp.
2. Check content timestamp.
3. Compare a hash of the page with a stored hash.
4. Non-significant differences (ads, fortunes, request timestamp):
   - only hash text content, or “useful” text content;
   - compare distribution of *n*-grams (shingling);
   - or even compute edit distance with previous version.

Adapting strategy to each different archived website?

TELECOM ParisTech

- Some modern Web sites only work when cookies are activated (session cookies), or when JavaScript code is interpreted
- Regular Web crawlers (wget, Heritrix, Apache Nutch) do not usually perform any cookie management and do not interpret JavaScript code
- Crawling of some Websites therefore require more advanced tools

Web scraping frameworks such as scrapy (Python) or WWW::Mechanize (Perl) simulate a Web browser interaction and cookie management (but no JS interpretation)

Headless browsers such as htmlunit simulate a Web browser, including simple JavaScript processing

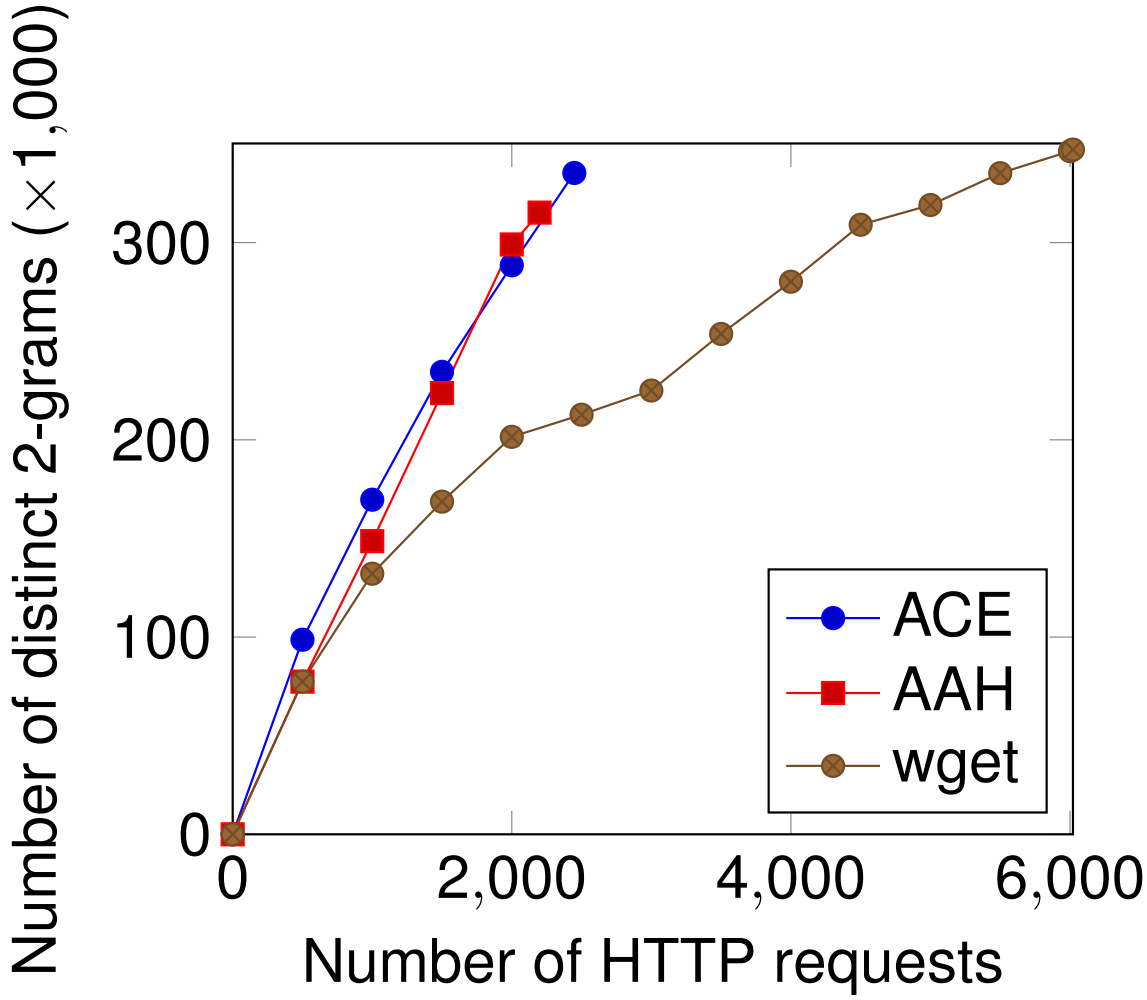
Browser instrumentors such as Selenium allow full instrumentation of a regular Web browser (Chrome, Firefox, Internet Explorer)

OXPath: a full-fledged navigation and extraction language for complex Web sites (**?**)

- Many Web sites (especially, Web forums, blogs) use one of a few content management systems (CMS)
  - Web sites that use the same CMS will be similarly structured, present a similar layout, etc.
  - Information is somewhat structured in CMSs: publication date, author, tags, forums, threads, etc.
  - Some structure differences may exist when Web sites use different versions, or different themes, of a CMS

- Traditional crawling approaches crawl Web sites independently of the nature of the sites and of their CMS
- When the CMS is known:
  - Potential for much more efficient crawling strategies (avoid pages with redundant information, uninformative pages, etc.)
  - Potential for automatic extraction of structured content
- Two ways of approaching the problem:
  - Have a handcrafted knowledge base of known CMSs, their characteristics, how to crawl and extract information (**??**) (AAH)
  - Automatically infer the best way to crawl a given CMS (**?**) (ACE)
- Need to be robust w.r.t. template change

- One main challenge in intelligent crawling and content extraction is to identify the CMS and then perform the best crawling strategy accordingly
- Detecting CMS using:
  1. URL patterns,
  2. HTTP metadata,
  3. textual content,
  4. XPath patterns, etc.
- These can be manually described (AAH), or automatically inferred (ACE)
- For instance the vBulletin Web forum content management system, that can be identified by searching for a reference to a `vbulletin_global.js` JavaScript script by using a simple `//script/@src` XPath expression.



TELECOM ParisTech

1	google.com
2	facebook.com
3	youtube.com
4	yahoo.com
5	baidu.com
6	wikipedia.org
7	live.com
8	twitter.com
9	qq.com
10	amazon.com
11	blogspot.com
12	linkedin.com
13	google.co.in
14	taobao.com
15	sina.com.cn
16	yahoo.co.jp
17	msn.com
18	wordpress.com
19	google.com.hk
20	t.co
21	google.de
22	ebay.com
23	google.co.jp
24	googleusercontent.com
25	google.co.uk
26	yandex.ru
27	163.com
28	weibo.com

(Alexa)

1 google.com
2 facebook.com
3 youtube.com
4 yahoo.com
5 baidu.com
6 wikipedia.org
7 live.com
8 twitter.com
9 qq.com
10 amazon.com
11 blogspot.com
12 linkedin.com
13 google.co.in
14 taobao.com
15 sina.com.cn
16 yahoo.co.jp
17 msn.com
18 wordpress.com
19 google.com.hk
20 t.co
21 google.de
22 ebay.com
23 google.co.jp
24 googleusercontent.com
25 google.co.uk
26 yandex.ru
27 163.com
28 weibo.com

(Alexa)

Social networking sites

1	google.com
2	facebook.com
3	youtube.com
4	yahoo.com
5	baidu.com
6	wikipedia.org
7	live.com
8	twitter.com
9	qq.com
10	amazon.com
11	blogspot.com
12	linkedin.com
13	google.co.in
14	taobao.com
15	sina.com.cn
16	yahoo.co.jp
17	msn.com
18	wordpress.com
19	google.com.hk
20	t.co
21	google.de
22	ebay.com
23	google.co.jp
24	googleusercontent.com
25	google.co.uk
26	yandex.ru
27	163.com
28	weibo.com

(Alexa)

Social networking sites

Sites with social networking features (friends, user-shared content, user profiles, etc.)

Huge numbers of users (2012):

Facebook	900 million
QQ	540 million
W. Live	330 million
Weibo	310 million
Google+	170 million
Twitter	140 million
LinkedIn	100 million

Huge numbers of users (2012):

Facebook	900 million
QQ	540 million
W. Live	330 million
Weibo	310 million
Google+	170 million
Twitter	140 million
LinkedIn	100 million

Huge volume of shared data:

250 million tweets per day on Twitter (3,000 per second on average!)...

... including statements by heads of states, revelations of political activists, etc.

- Theoretically possible to crawl social networking sites using a regular Web crawler
- Sometimes not possible:
  `https://www.facebook.com/robots.txt`
- Often very inefficient, considering politeness constraints
- Better solution: Use provided social networking APIs
  `https://dev.twitter.com/docs/api/1.1`
  `https://developers.facebook.com/docs/graph-api/reference/v2.1/`
  `https://developer.linkedin.com/apis`
  `https://developers.google.com/youtube/v3/`
- Also possible to buy access to the data, directly from the social network or from brokers such as `http://gnip.com/`

- Most social networking Web sites (and some other kinds of Web sites) provide APIs to effectively access their content
- Usually a RESTful API, occasionally SOAP-baed
- Usually require a token identifying the application using the API, sometimes a cryptographic signature as well
- May access the API as an authenticated user of the social network, or as an external party
- APIs seriously limit the rate of requests:
  `https://dev.twitter.com/docs/api/1.1/get/search/tweets`

- Mode of interaction with a Web service
- Follow the KISS (Keep it Simple, Stupid) principle
- Each request to the service is a simple HTTP GET method
- Base URL is the URL of the service
- Parameters of the service are sent as HTTP parameters (in the URL)
- HTTP response code indicates success or failure
- Response contains structured output, usually as JSON or XML
- No side effect, each request independent of previous ones

- Two main APIs:
  - REST APIs, including search, getting information about a user, a list, followers, etc. `https://dev.twitter.com/docs/api/1.1`
  - Streaming API, providing real-time result
- Very limited history available
- Search can be on keywords, language, geolocation (for a small portion of tweets)

- Often useful to combine results from different social networks
- Numerous libraries facilitating SN API accesses (twipy, Facebook4J, FourSquare VP C++ API...) incompatible with each other... Some efforts at generic APIs (OneAll, APIBlender (**?**))
- Example use case: No API to get all check-ins from FourSquare, but a number of check-ins are available on Twitter; given results of Twitter Search/Streaming, use FourSquare API to get information about check-in locations.

(Deep Web, Hidden Web, Invisible Web)

All the content on the Web that is not directly accessible through hyperlinks. In particular: HTML forms, Web services.

Size estimate: 500 times more content than on the surface Web! (**?**). Hundreds of thousands of deep Web databases (**?**)

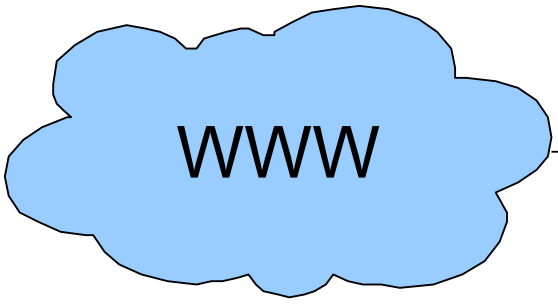
Example

- *Yellow Pages* and other directories;
- Library catalogs;
- Weather services;
- US Census Bureau data;
- etc.

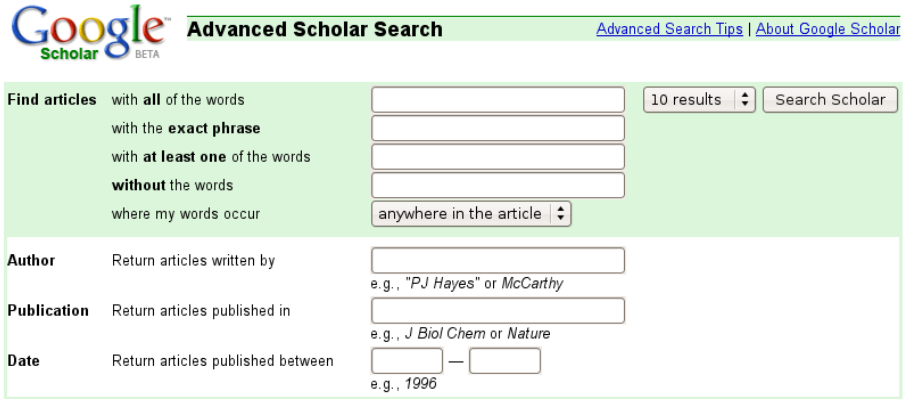
- Content of the deep Web hidden to classical Web search engines (they just follow links)
- But very valuable and high quality!
- Even services allowing access through the surface Web (e.g., e-commerce) have more semantics when accessed from the deep Web
- How to benefit from this information?
- How to analyze, extract and model this information?

Focus here: Automatic, unsupervised, methods, for a given domain of interest

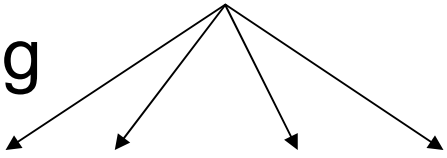
Extensional Approach

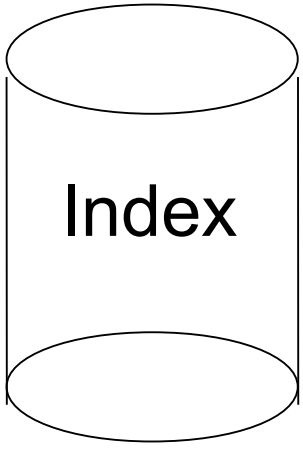


discovery

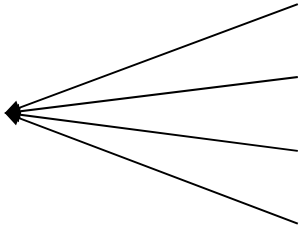


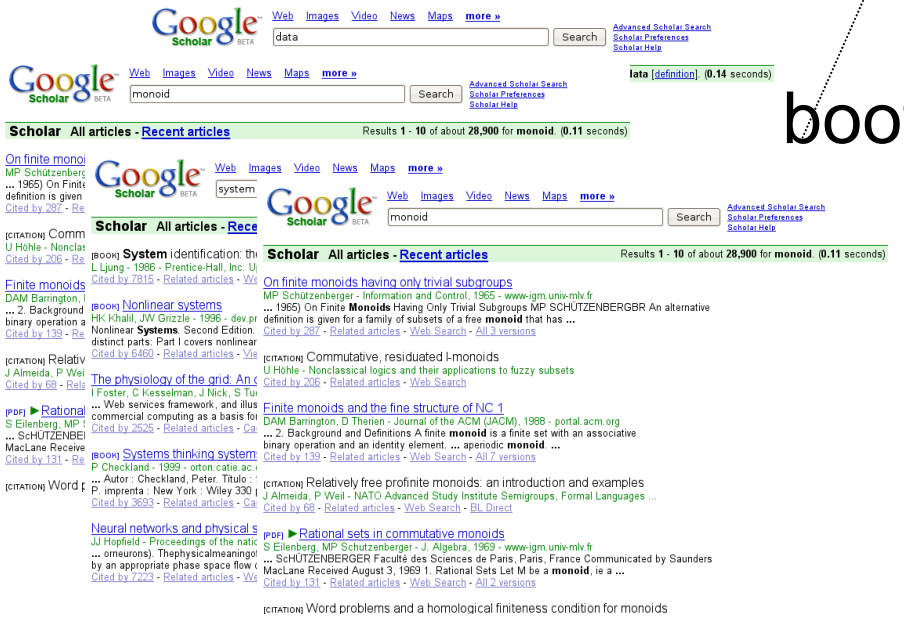
siphoning





indexing



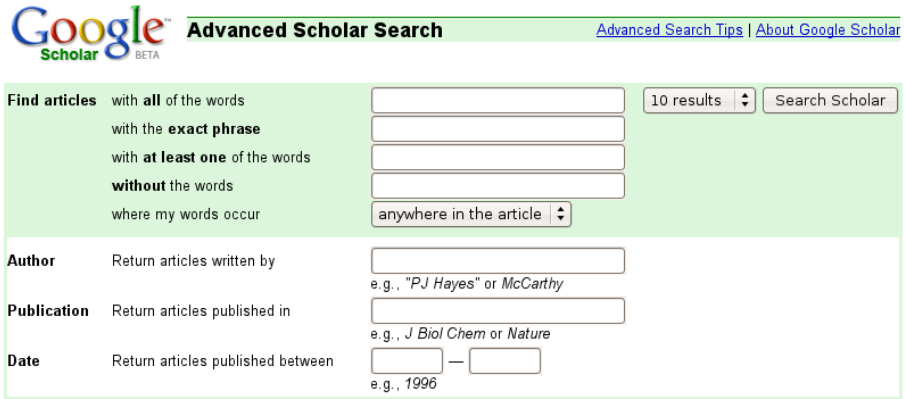


bootstrap

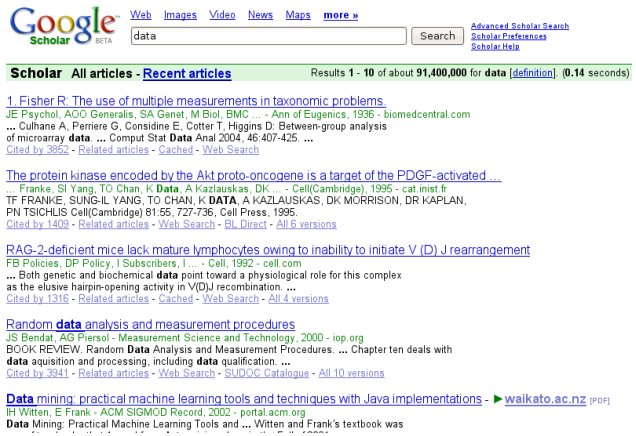
- Main issues:
  - Discovering services
  - Choosing appropriate data to submit forms
  - Use of data found in result pages to bootstrap the siphoning process
  - Ensure good coverage of the database
- Approach favored by Google, used in production (**?**)
- Not always feasible (huge load on Web servers)

Intensional Approach

discovery



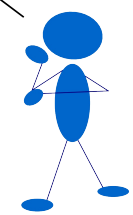
probing



analyzing

Form wrapped as a Web service

query



- More ambitious (**??**)
- Main issues:
  - Discovering services
  - Understanding the structure and semantics of a form
  - Understanding the structure and semantics of result pages
  - Semantic analysis of the service as a whole
  - Query rewriting using the services
- No significant load imposed on Web servers

- Numerous works on form understanding and information extraction from the deep Web (**???**)
- Formal models for answering queries under access pattern restrictions (**????**)
- Siphoning of hidden Web databases (**???**)

- Those works ignore lots of quirky dimensions of deep Web interfaces
- Here: towards a more comprehensive framework for deep Web modeling and querying

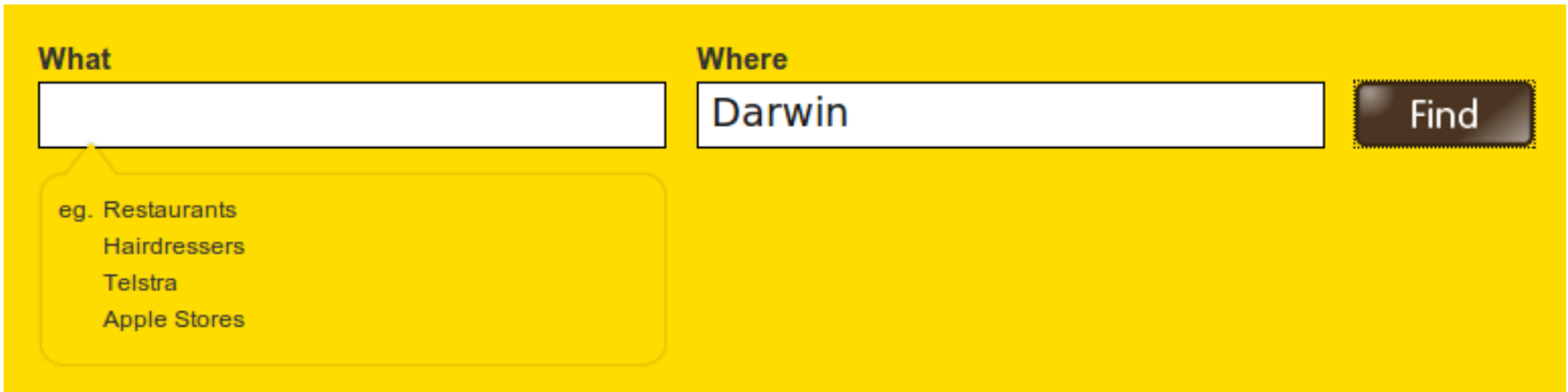
Deep Web sources offer views over (most often relational) data, through, at the very least:

- selection (depending on user's query, or implicit in the service), in particular inequalities
- projection (not available attributes are exported by a given service)

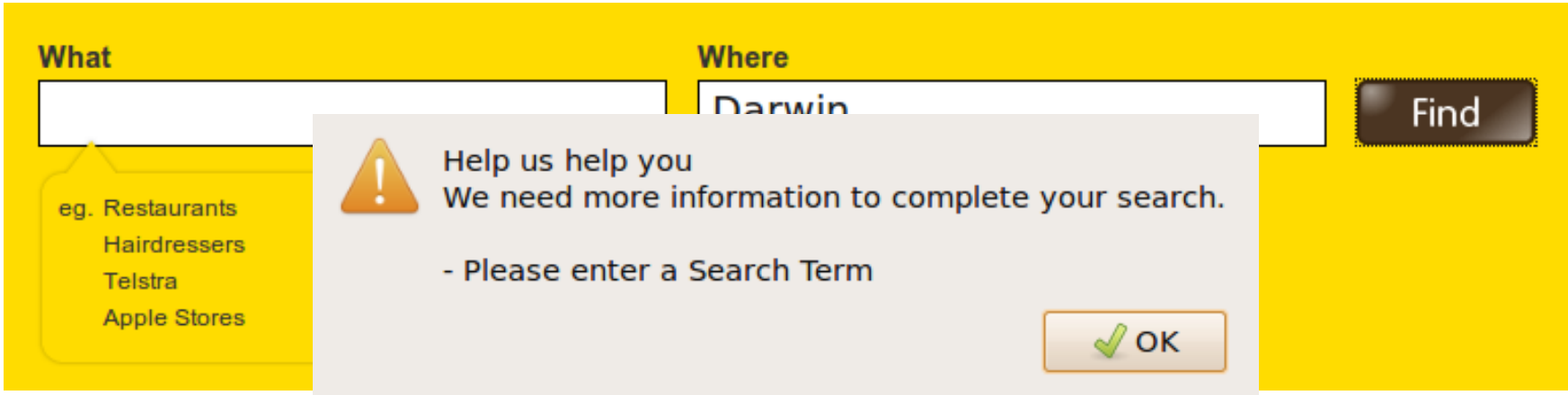
And also (but less critically):

- joins (quite common in a Web application – but from an outsider's perspective, often enough to see the result of a join as the relation of interest)
- union, intersubsection, difference, etc. (relatively rare)
- aggregation (usually not the most important part of the service)
- more complex processing (rare in practice)

Australian Yellow Pages search form:



Australian Yellow Pages search form:



Required attributes, dependencies between attributes of the form, etc.

IMDb advanced search sort criteria:

Sort by: **MOVIEmeter▲** | A-Z | User Rating | Num Votes | US Box Office | Runtime | Year | US Release Date

1. 

Friends (1994 TV Series)
Episode: **The One with the Routine** (1999)

Add to Watchlist

★★★★★★★★½☆ **8.4**/10

Janine is going to be a party person in a New Year's Eve TV broadcast and asks Joey, Monica and Ross to come along for the taping...

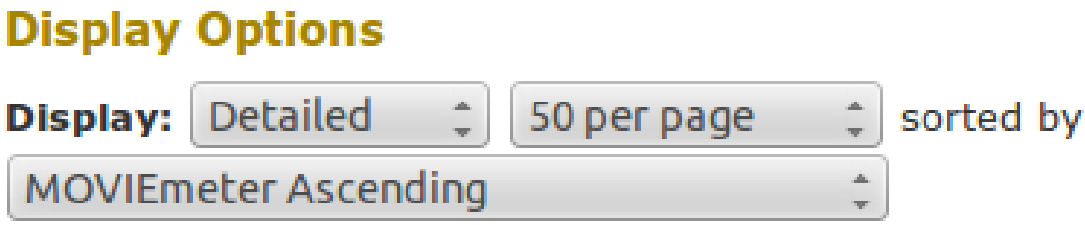
Dir: Kevin S. Bright With: Jennifer Aniston, Courteney Cox, Lisa Kudrow

Comedy | Romance

22 mins. TV14

Different possible sort criteria, some according to non-exported attributes

Paging in IMDb:



10,001-10,050 of 100,289 titles. « Prev Next »

Each page of results requires a separate network access, and therefore has a cost

What you get when you try to access the 100,001-th result to an IMDb advanced query:

Error

Sorry, IMDb does not serve more than 100000 results for any query. (You asked for results starting from 100001)

Only a (top-ranked) subset of the results is available for each access

Twitter API rate limitation:

REST API Rate Limiting

The default rate limit for calls to the REST API varies depending on the authorization method being used and whether the method itself requires authentication.

- Unauthenticated calls are permitted 150 requests per hour. Unauthenticated calls are measured against the public facing IP of the server or device making the request.
- OAuth calls are permitted 350 requests per hour and are measured against the `oauth_token` used in the request.

Limited rate of queries per minute, hour, query... Several services of the same source may share the same limits.

Several views of the same information on IMDB:

It's a Wonderful Life (1946) Top 5000

UR 130 min - Drama | Fantasy - 7 January 1947 (USA)

8.7

Your rating: ★★★★★★★★★★ -/10

Ratings: **8.7**/10 from 146,420 users

Reviews: 556 user | 162 critic

An angel helps a compassionate but despairingly frustrated businessman by showing what life would have been like if he never existed.

Director: Frank Capra

Writers: Frances Goodrich (screenplay), Albert Hackett (screenplay), and 4 more credits »

Stars: James Stewart, Donna Reed and Lionel Barrymore | See full cast and crew

+ Watchlist Share...

Several views of the same information on IMDB:

1. It's a Wonderful Life (1946)
   - aka "Frank Capra's It's a Wonderful Life" - USA *(complete title)*
   - aka "La vie est belle" - Belgium *(French title)*, Canada *(French title)*, France
   - aka "¡Qué bello es vivir!" - Peru *(imdb display title)*, Spain
   - aka "Ist das Leben nicht schön?" - Austria *(TV title)*, West Germany *(TV title)*
   - aka "¡Que bello es vivir!" - Uruguay
   - aka "A Felicidade Não Se Compra" - Brazil
   - aka "Az élet csodaszép" - Hungary
   - aka "Det er herligt at leve" - Denmark
   - aka "Divan život" - Serbia
   - aka "Divan zivot" - Yugoslavia *(Croatian title)* *(imdb display title)*
   - aka "Do Céu Caiu Uma Estrela" - Portugal
   - aka "Ihmeellinen on elämä" - Finland
   - aka "La vita è meravigliosa" - Italy
   - aka "Livet är underbart" - Sweden
   - aka "Livet er vidunderlig" - Norway *(imdb display title)*
   - aka "Mens, durf te leven" - Netherlands *(informal literal title)*
   - aka "Mia yperohi zoi" - Greece *(transliterated ISO-LATIN-1 title)*
   - aka "O viata minunata" - Romania *(imdb display title)*
   - aka "Qué bello es vivir" - Argentina
   - aka "Que bonic és viure!" - Spain *(Catalan title)*
   - aka "Que la vie est belle" - Belgium *(French title)*
   - aka "Sahane hayat" - Turkey *(Turkish title)* *(DVD title)*
   - aka "Subarashiki kana, jinsei!" - Japan
   - aka "To wspaniale zycie" - Poland
   - aka "Wat een mooi leven" - Belgium *(Flemish title)*
   - aka "Zycie jest cudowne" - Poland

Several views of the same information on IMDB:

1.  **It's a Wonderful Life** (1946) — Add to Watchlist
8.7/10
An angel helps a compassionate but despairingly frustrated businessman by showing what life would have been like if he never existed.
Dir: Frank Capra With: James Stewart, Donna Reed, Lionel Barrymore
Drama | Fantasy
130 mins. UR

2.  **It Happened One Night** (1934) — Add to Watchlist
8.3/10
A spoiled heiress, running away from her family, is helped by a man who's actually a reporter looking for a story.
Dir: Frank Capra With: Clark Gable, Claudette Colbert, Walter Connolly
Comedy | Romance
105 mins. UR

3.  **Mr. Smith Goes to Washington** (1939) — Add to Watchlist
8.4/10
A naive man is appointed to fill a vacancy in the US Senate. His plans promptly collide with political corruption, but he doesn't back down.
Dir: Frank Capra With: James Stewart, Jean Arthur, Claude Rains
Comedy | Drama
129 mins. Approved

Same relation(s), different attributes projected out

Release date API on IMDb:

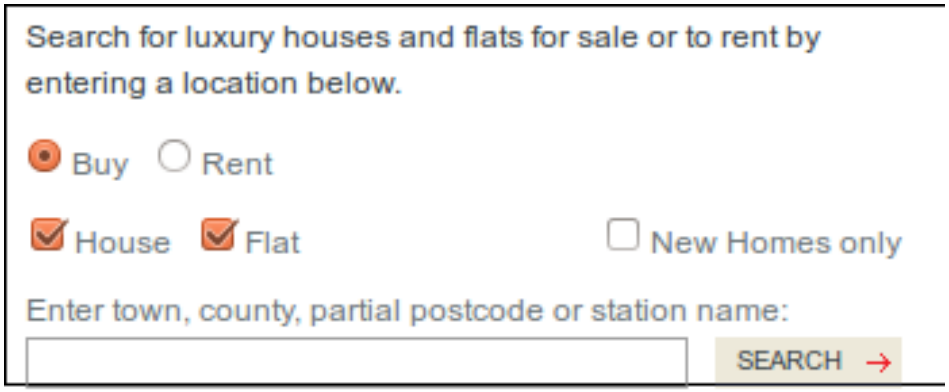
Release dates for

It's a Wonderful Life (1946) More at IMDbPro »

Country	Date
USA	20 December 1946 (New York City, New York)

The granularity of the presented information may not be the most precise one

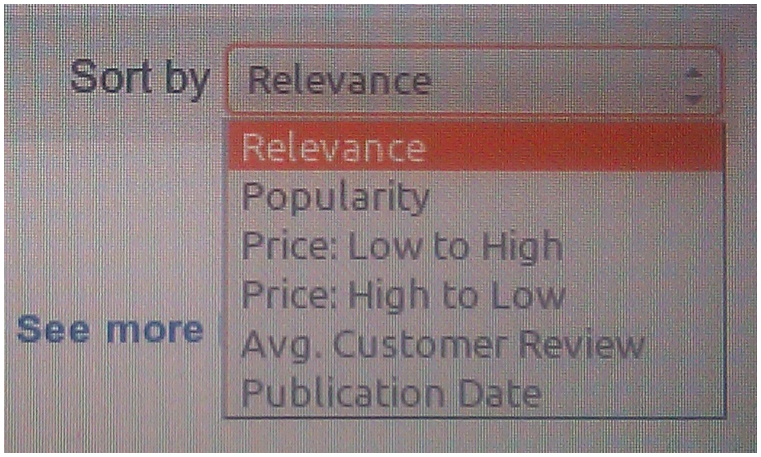
Savills property search:



Publication time is a special attribute of interest:

- may or may not be exported
- may or may not be queriable (sometimes in a very weird way!)
- often used as a ranking criterion
- granularity plays an important role
- publication date $<$ query date

Amazon: Books sorting options:



- Proprietary ranking functions
- Weighted combination of attributes with unknown weights (**?**)
- Ranking according to an unexported attribute

Some of IMDb advanced search options:

Advanced Title Search

Want to get a list of comedies from the 1970s that have at least 1000 votes and an average rating of 7.5 or higher? Use Advanced Title Search.

Advanced Name Search

Want a list of males in the database who are Virgos and over 6 feet tall? Use Advanced Name Search.

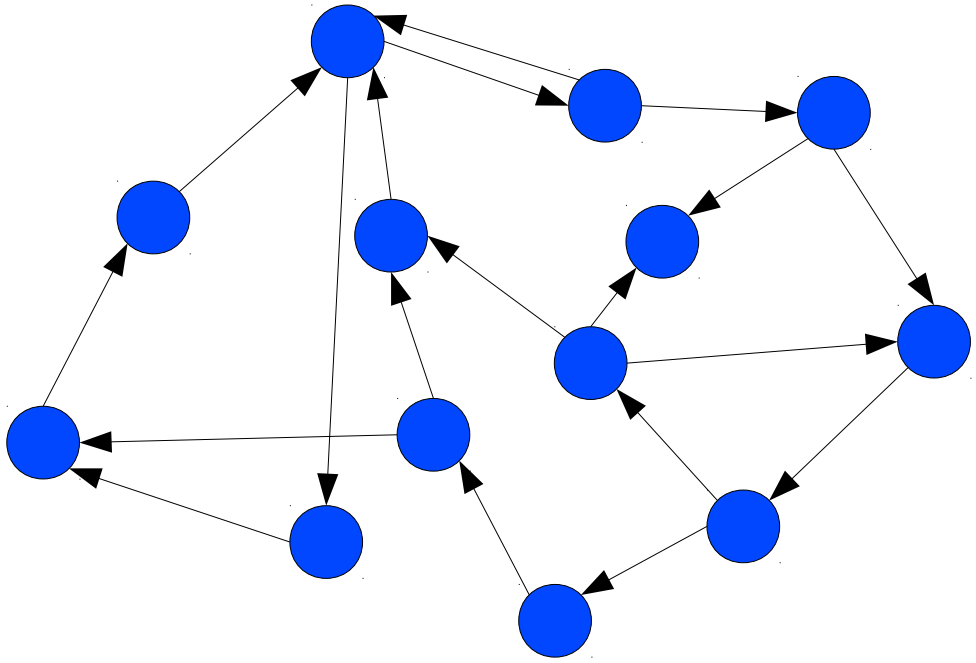
Collaborations and Overlaps

Want a list of titles in which both Brad Pitt and George Clooney appeared? Or a list of people who worked on both Forrest Gump and Apollo 13? Try searching Collaborations and Overlaps.

- services of the same source provide different correlated views of the same data
- dependencies (inclusion) across services are common too
- a given service often satisfies some key dependencies

- non-conjunctive forms (common in digital library applications)
- unknown characteristics of information retrieval systems (keyword querying vs exact querying, indexing of stop words, stemming used, etc.)
- intricate interactions (AJAX autocompletion, submitting a form as a first step before submitting another form, etc.)
- potential side effects of a service

A directed graph



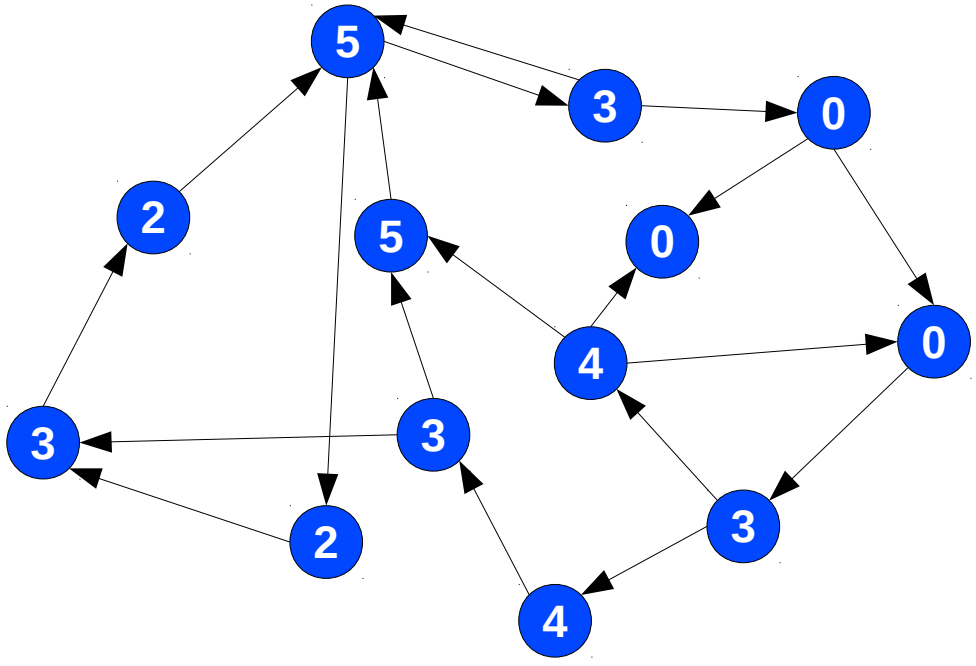
Web

Social network

P2P

etc.

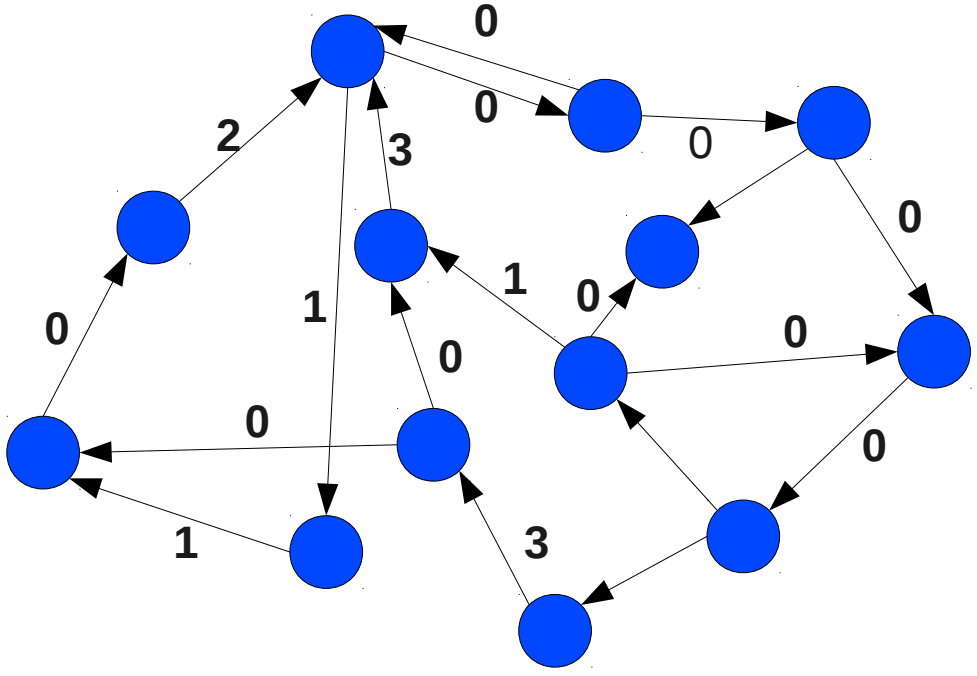
Weighted



Let *u* be a node,

β(u) = count of the word *Bhutan* in all the tweets of u

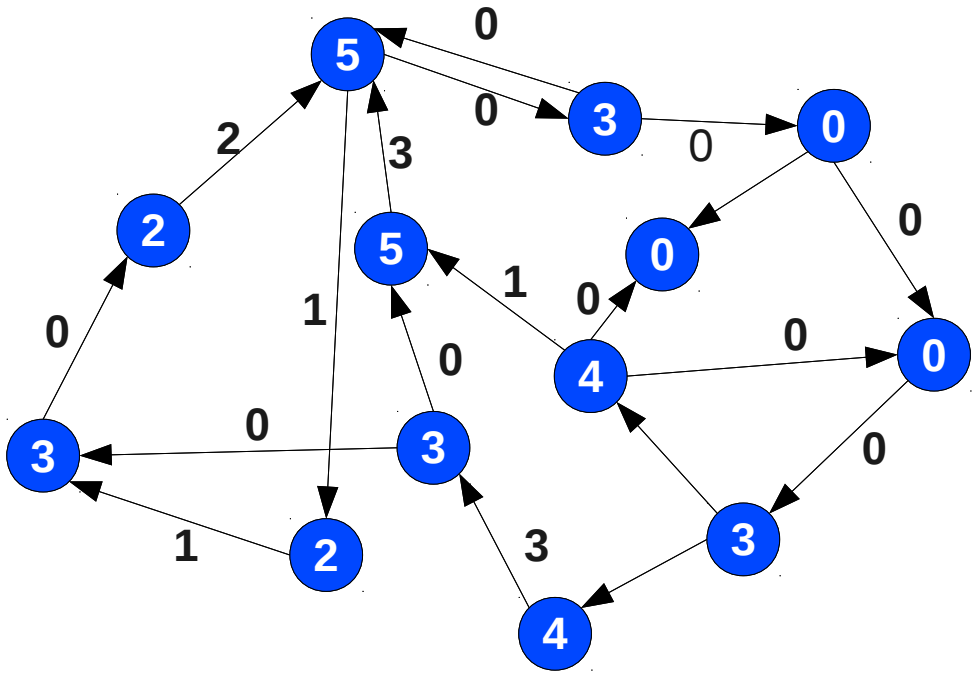
Even more weighted



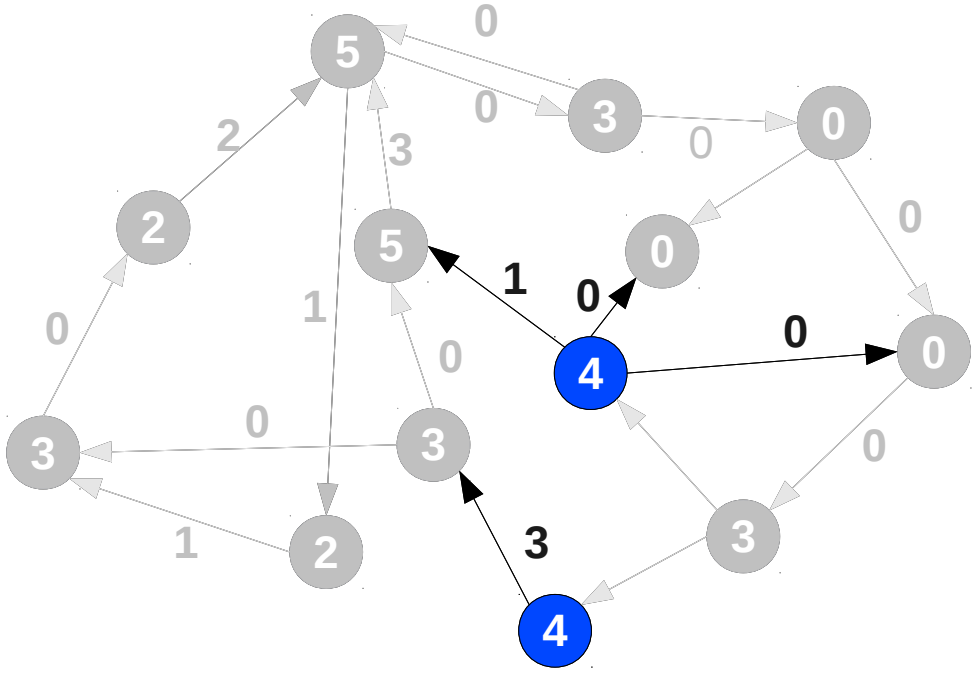
Let (*u*, *v*) be an edge,

α(u) = count of the word *Bhutan* in all the tweets of u mentioning v

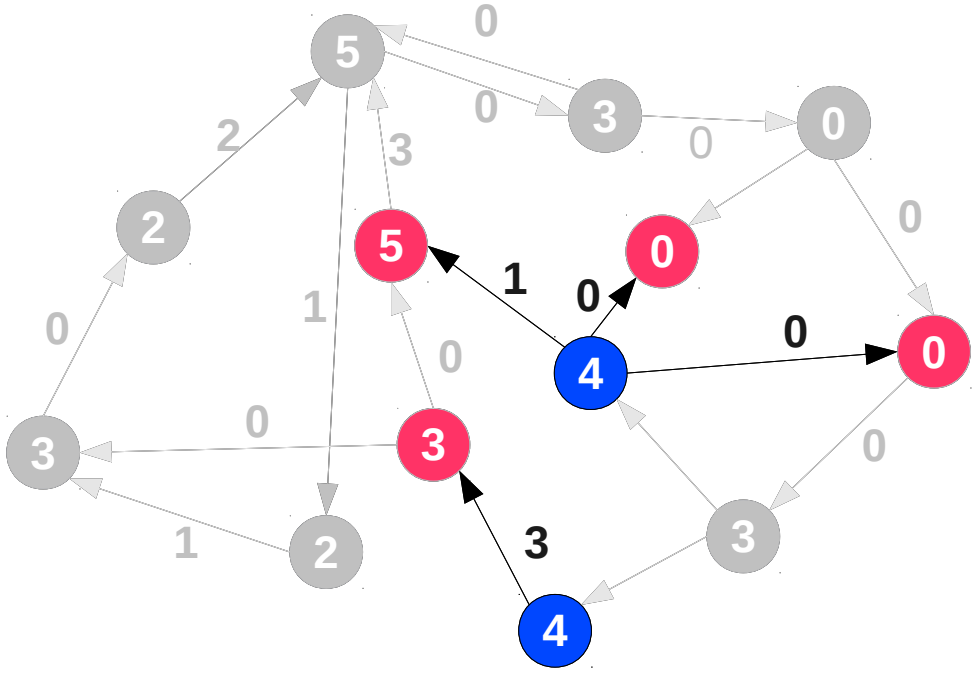
The total graph



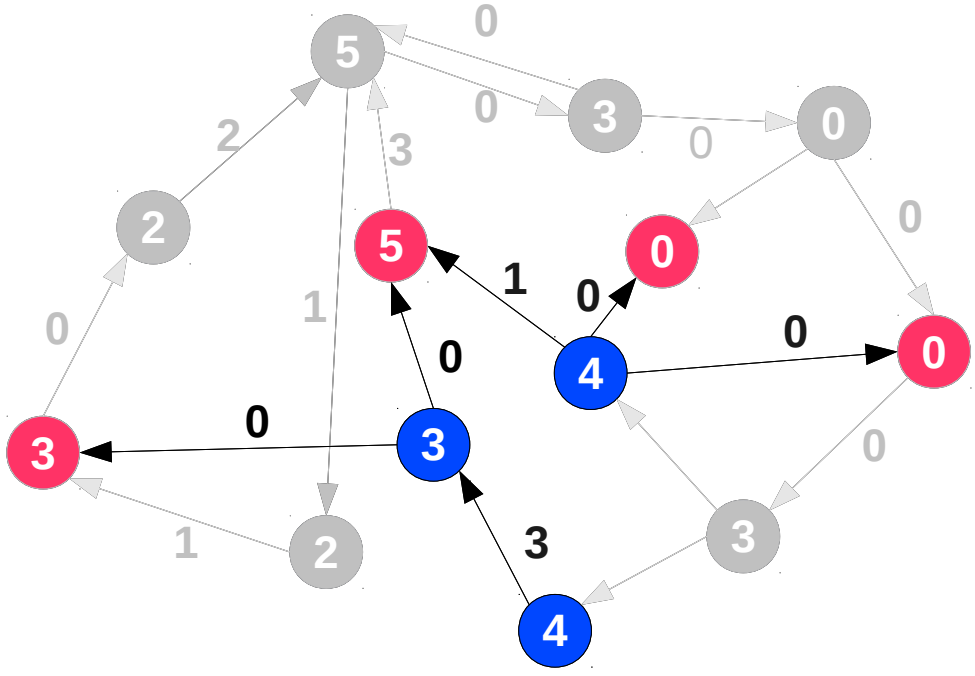
A seed list



The frontier



Crawling one node



A crawl sequence

Let V_0 be the seed list, a set of nodes,
a *crawl sequence,* starting from V_0, is

$$\{ v_i, v_i \text{ in frontier}(V_0 \cup \{v_0, v_1, .. , v_{i-1}\}) \}$$

Goal of a focused crawler

Produce crawl sequences with

global scores (sum) as high as possible

A high-level algorithm

Estimate scores at the frontier

Pick a node from the frontier

Crawl the node

- Different estimators can be used:
  - Distance with respect to a seed node
  - Average score of pages pointing to a node
  - Average score of pages pointed to by pages pointing to a node
  - etc.
- Possible to automatically find the estimators best adapted to a given focus crawl using reinforcement learning (**?**)

What you should remember

- Crawling as a graph-browsing problem.
- Shingling for identifying duplicates.
- Numerous engineering issues in building a Web-scale crawler.
- Crawling modern Web content is not as easy as launching a traditional Web crawler
- Often critical to focus the crawl towards content of interest
- Ideally: a traditional large-scale crawler that knows when to delegate to more specialized crawling mechanisms (tools querying social networking APIs, deep Web crawlers, JS-aware crawlers, etc.)
- Huge variety of tools, techniques, suitable for different needs

- Wget, a simple yet effective Web spider (free software)
- Heritrix, a Web-scale highly configurable Web crawler, used by the Internet Archive (free software)
- HTML Parser, TagSoup: Java libraries for parsing real-world Web pages

To go further

- A good textbook (**?**)
- Main references:
  - HTML 4.01 recommendation (**?**)
  - HTTP/1.1 RFC (**?**)