Processamento e Recuperação de Informação

Web Crawling

Departamento de Engenharia Informática
Instituto Superior Técnico

1º Semestre
2018/2019

Bibliography

- Bing Liu, Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data, 2nd edition. Chapter 8
- most slides based on presentation by Filippo Menczer, Indiana University School of Informatics

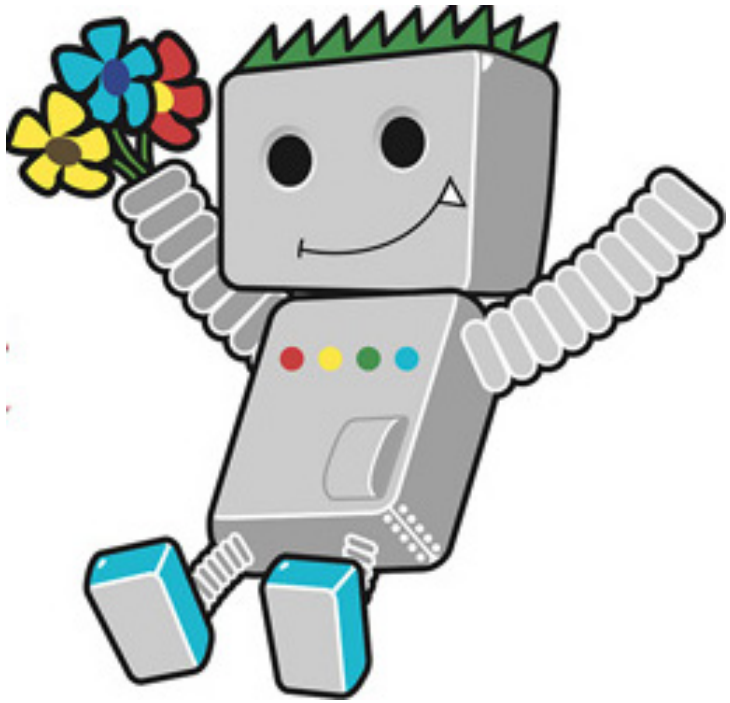
Outline

- Motivation and taxonomy of crawlers
- Basic crawlers and implementation issues
- Universal crawlers
- Preferential (focused and topical) crawlers
- Crawler ethics and conflicts

Outline

Names

- Crawler
- Spider
- Robot (or bot)
- Web agent
- Wanderer, worm, ...



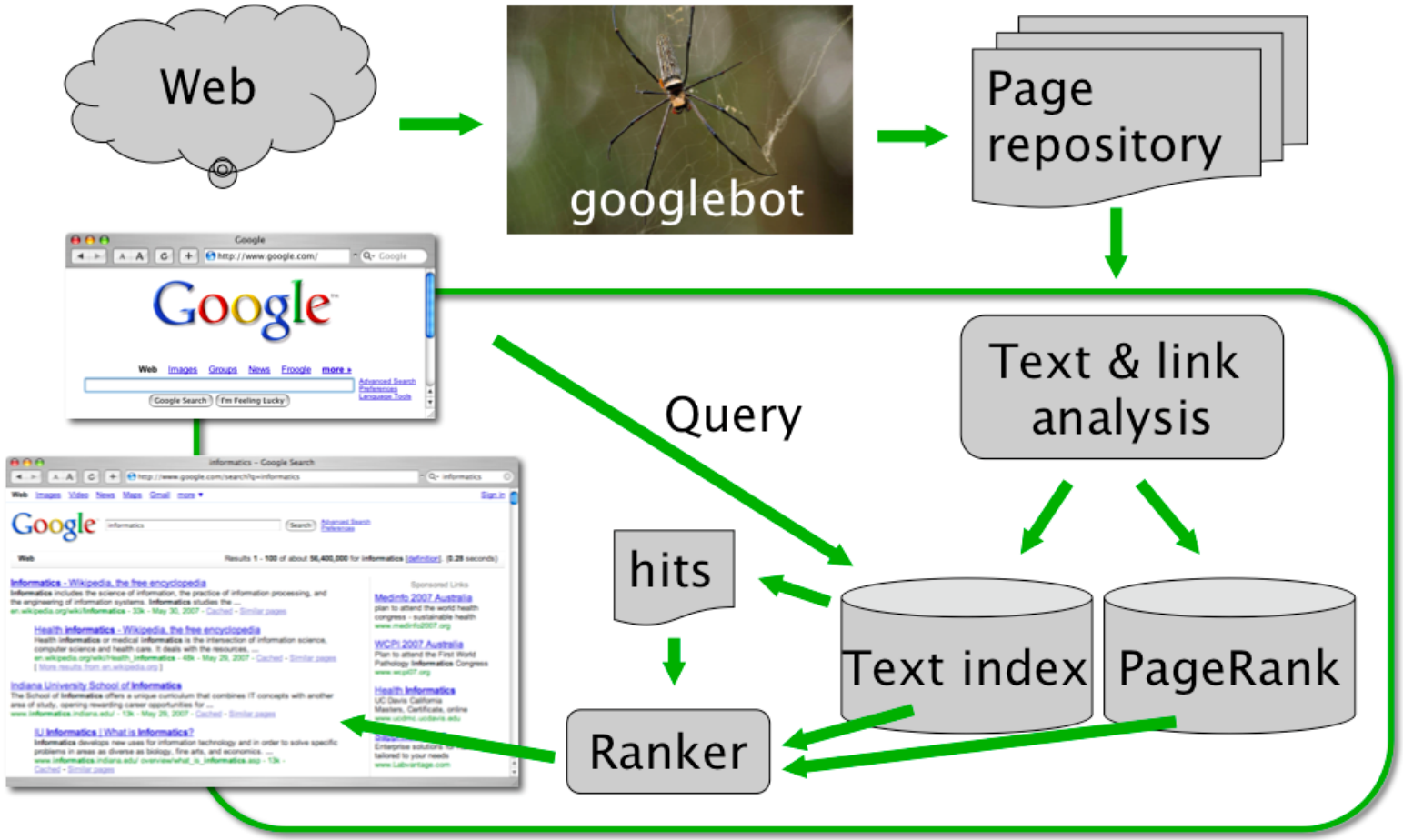
instances:

googlebot, scooter (altavista), slurp (yahoo), msnbot, ...

Why Develop a Crawler?

- Support universal search engines (Google, Yahoo, Bing, Ask, etc.)
- Vertical (specialized) search engines, e.g. news, shopping, papers, recipes, reviews, etc.
- Business intelligence: keep track of potential competitors, partners
- Monitor Web sites of interest
- Evil: harvest emails for spamming, phishing ...

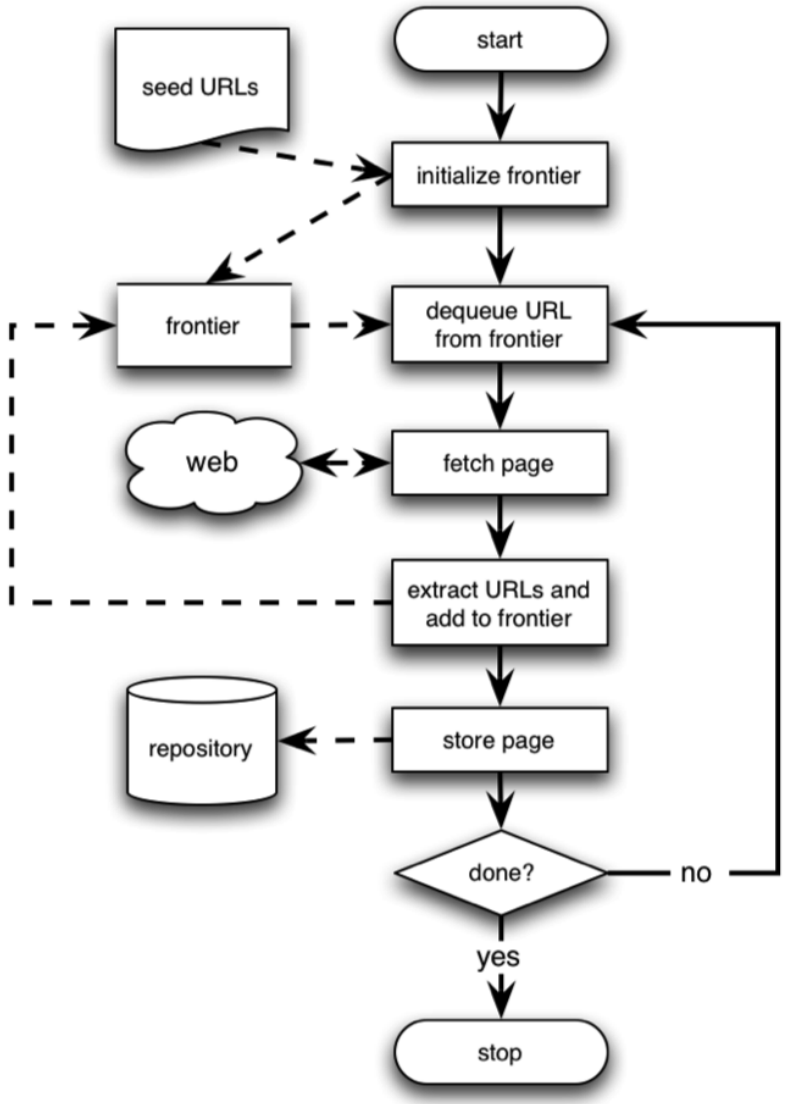
The Crawler and the Search Engine



Outline

Basic Crawler

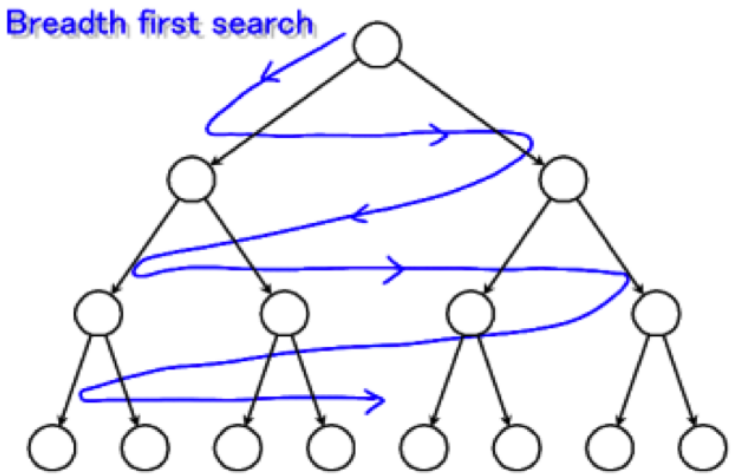
- This is a **sequential crawler**
- Seeds can be any list of starting URLs
- Order of page visits is determined by **frontier** data structure
- Stop criterion can be anything



BFS vs DFS Graph Traversal

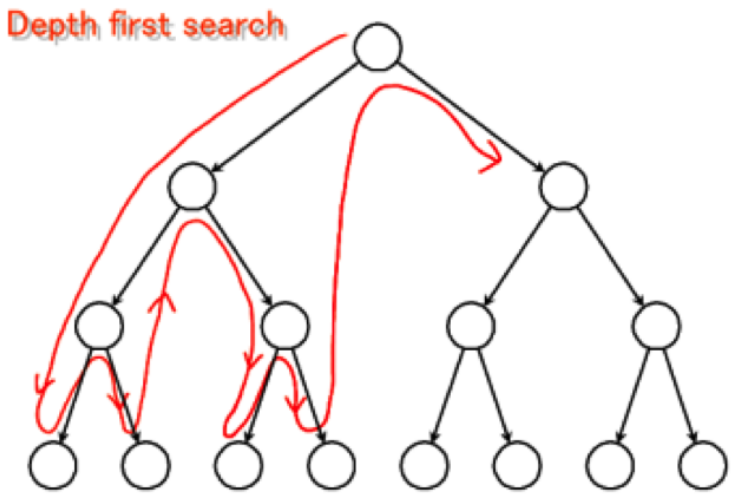
Breadth First Search

- Implemented with QUEUE (FIFO)
- Finds pages along shortest paths
- If we start with "good" pages, this keeps us close; maybe other good stuf...



Depth First Search

- Implemented with STACK (LIFO)
- Wander away ("lost in cyberspace")



Main Loop of a Basic Crawler


```
frontier is a queue
frontier = read_seeds(file)

while length(frontier) > 0 & total_collected < max

    next_link = shift frontier

    page = fetch(next_link)

    add_to_index(page)

    links = extract_links(page, next_link)

    push frontier, process(links)
```

Implementation Issues

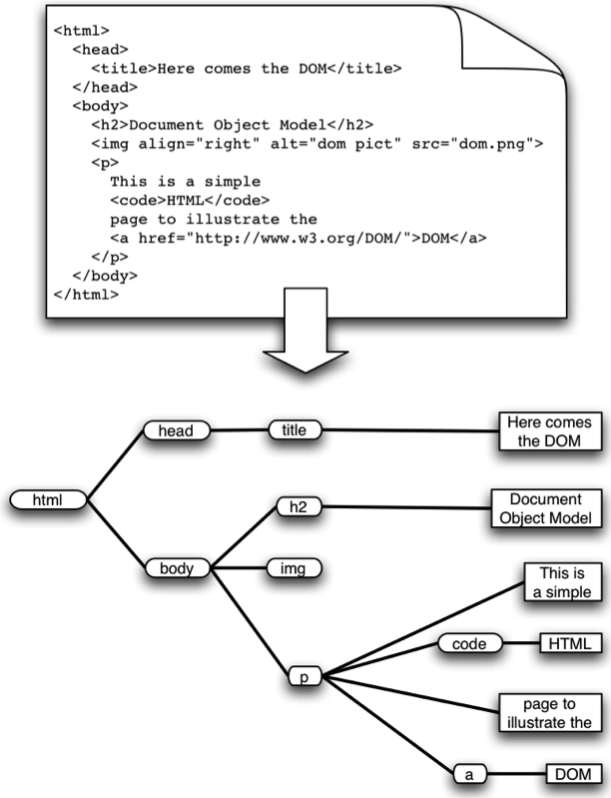
- Don't want to fetch same page twice!
  - Keep lookup table (hash) of visited pages
  - What if not visited but in frontier already?
- The frontier grows very fast!
  - May need to prioritize for large crawls
- Fetcher must be robust!
  - Don't crash if download fails
  - Have a timeout mechanism
- Determine file type to skip unwanted files
  - Can try using extensions, but not reliable
  - Can issue HEAD HTTP commands to get Content-Type (MIME) headers, but there is overhead of extra Internet requests

Implementation Issues: Fetching

- Get only the first 10-100 KB per page
- Take care to detect and break **redirection loops**
- Soft fail for
  - timeout,
  - server not responding,
  - file not found
  - ...other errors

Implementation Issues: Parsing HTML

- HTML has the structure of a DOM (Document Object Model) tree
- Unfortunately actual HTML is often incorrect in a strict syntactic sense
  - Crawlers, like browsers, must be robust/forgiving
  - Fortunately there are tools that can help
- Must pay attention to HTML entities and unicode in text



Implementation issues: Parsing Other Formats

What to do with other formats?

- JavaScript,
- SVG,
- RSS,
- Flash,
- ...

Implementation issues: Relative vs. Absolute URLs

Crawler must translate relative URLs into absolute URLs
Need to obtain Base URL from HTTP header, or HTML Meta tag, or else current page path by default

Examples:

Base: `http://www.cnn.com/linkto/`

Relative URL: `intl.html`

Absolute URL: `http://www.cnn.com/linkto/intl.html`

Relative URL: `/US/`

Absolute URL: `http://www.cnn.com/US/`

Implementation issues: URL canonicalization

All of these:

`http://www.cnn.com/TECH`
`http://WWW.CNN.COM/TECH/`
`http://www.cnn.com:80/TECH/`
`http://www.cnn.com/bogus/../TECH/`

Are really equivalent to this canonical form:

http://www.cnn.com/TECH/

In order to avoid duplication, the crawler must transform all URLs into canonical form
Definition of "canonical" is arbitrary, e.g.: Could always include port, or only include port when not default

More on Canonical URLs

Some transformations are trivial, for example:

`http://tecnico.ulisboa.pt/`
`http://tecnico.ulisboa.pt`

`http://tecnico.ulisboa.pt/index.html#fragment`
`http://tecnico.ulisboa.pt/index.html`

`http://tecnico.ulisboa.pt/dir1/./../dir2/`
`http://tecnico.ulisboa.pt/dir2/`

`http://tecnico.ulisboa.pt/%7Efil/`
`http://tecnico.ulisboa.pt/~fil/`

`http://TECNICO.ULISBOA.PT/fil/`
`http://tecnico.ulisboa.pt/fil/`

More on Canonical URLs

Other transformations require heuristic assumption about the intentions of the author or configuration of the Web server:

Removing default file name

`http://tecnico.ulisboa.pt/fil/index.html`
`http://tecnico.ulisboa.pt/fil/`
This is reasonable in general but would be wrong in this case because the default happens to be `default.asp` instead of `index.html`

Trailing directory

`http://tecnico.ulisboa.pt/fil`
`http://tecnico.ulisboa.pt/fil/`
This is correct in this case but how can we be sure in general that there isn't a file named "fil" in the root dir?

Implementation issues: Spider Traps

Misleading sites: indefinite number of pages dynamically generated by CGI scripts
Paths of arbitrary depth created using soft directory links and path rewriting features in HTTP server

Heuristic defensive measures (the single option):

- Check URL length; assume spider trap above some threshold, for example 128 characters
- Watch for sites with very large number of URLs
- Eliminate URLs with non-textual data types
- May disable crawling of dynamic pages, if detected

Implementation issues: Page Repository

Naive: store each page as a separate file. Can map URL to unique filename using a hashing function, e.g. MD5 This generates a huge number of files, which is inefficient from the storage perspective

Better: combine many pages into a single large file, using some XML markup to separate and identify them. Must map URL to *filename*, *page_id*

Database options

Any RDBMS – large overhead

Light-weight, embedded key-value stores, such as Berkeley DB

Implementation Issues: Concurrency

A crawler incurs several delays:

Resolving the host name in the URL to an IP address using DNS

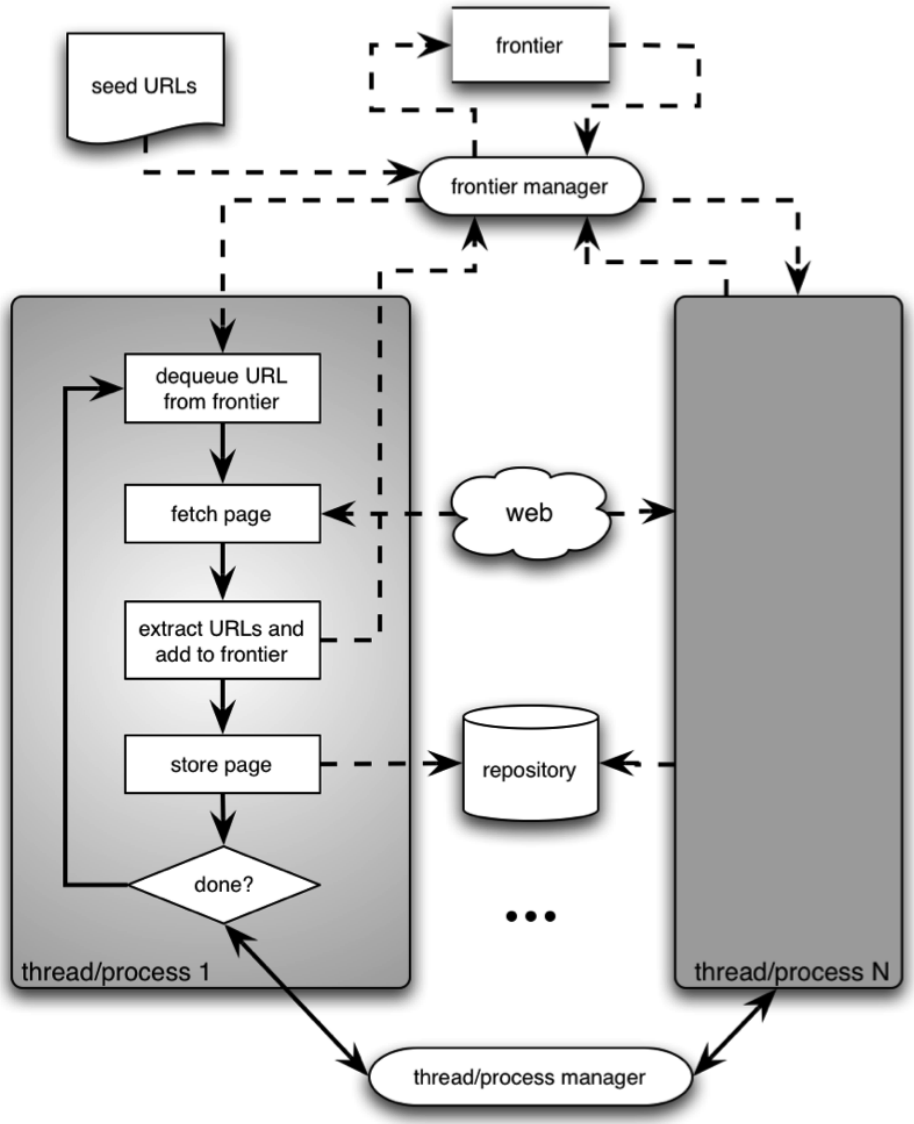
Connecting a socket to the server and sending the request

Receiving the requested page in response

Solution:

Overlap the above delays by fetching many pages concurrently

Architecture of a Concurrent Crawler



Concurrent Crawlers

- Can use multi-processing or multi-threading
- Each process or thread works like a sequential crawler, except they share data structures: frontier and repository
- Shared data structures must be synchronized (locked for concurrent writes)
- Speedup of factor of 5-10 are easy this way

Outline

Universal Crawlers

- Support universal search engines
- Large-scale
- Huge cost (network bandwidth) of crawl is amortized over many queries from users
- Incremental updates to existing index and other data repositories

Universal Crawlers

Two major issues:

Performance: Need to scale up to billions of pages

Policy: Need to trade-off coverage, freshness, and bias (e.g. toward "important" pages)

Large-Scale Crawlers: Performance and Scalability

Issues:

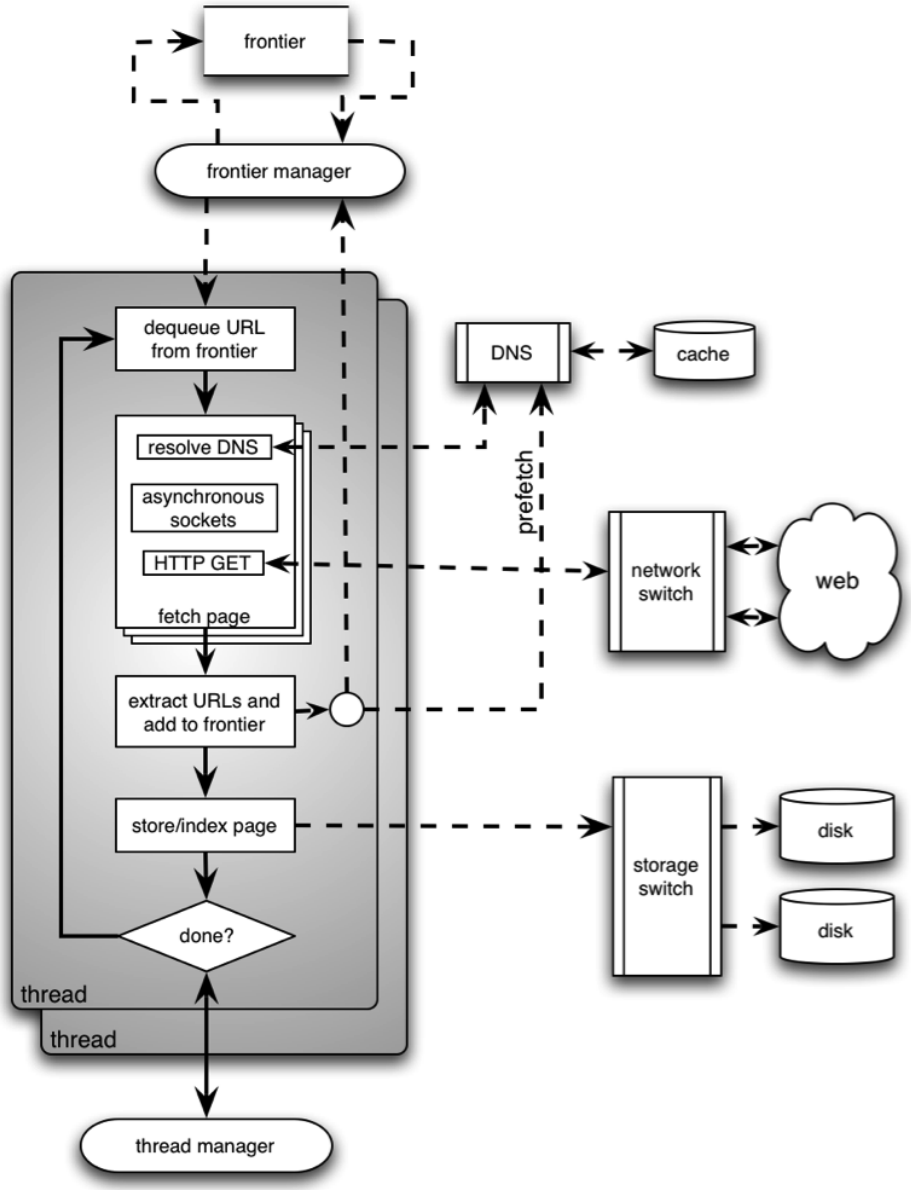
- Need to minimize overhead of DNS lookups
- Need to optimize utilization of network bandwidth and disk throughput (I/O is bottleneck)
  - Multi-processing or multi-threading do not scale up to billions of pages

Use asynchronous sockets

Non-blocking: hundreds of network connections open simultaneously

Polling socket to monitor completion of network transfers

High-level Architecture of a Scalable Universal Crawler



Universal Crawlers: Policy

Two main requirements must be met:

Coverage

- New pages get added all the time
- Can the crawler find every page?

Freshness

- Pages change over time, get removed, etc.
- How frequently can a crawler revisit ?

Trade-off!

Focus on most "important" pages (crawler bias)
"Importance" is subjective

Maintaining a "fresh" collection

Universal crawlers are never "done"
High variance in rate and amount of page changes
HTTP headers are notoriously unreliable

- Last-modified
- Expires

Solution

- Estimate the probability that a previously visited page has changed in the meanwhile
- Prioritize by this probability estimate

Estimating Page Change Rates

Algorithms for maintaining a crawl in which most pages are fresher than a specified epoch

Assumption: recent past predicts the future

(Ntoulas, Cho & Olston 2004)

- Frequency of change not a good predictor
- Degree of change is a better predictor

Do We Need to Crawl the Entire Web?

If we cover too much, it will get stale

There is an abundance of pages in the Web

For PageRank, pages with very low prestige are largely useless

What is the goal?

- General search engines: pages with high prestige
- News portals: pages that change often
- Vertical portals: pages on some topic

Outline

Breadth-first Crawlers

BF crawler tends to crawl high-PageRank pages very early
But why is this so?

Bias of Breadth-first Crawlers

The structure of the Web graph is very different from a random network

- Scale-free network: Power-law distribution of in-degree
- There are hub pages with very high PR and many incoming links
  - These are attractors: you cannot avoid them!

Preferential Crawlers

Assume we can estimate for each page an *importance measure*, $I(p)$
Want to visit pages in order of decreasing $I(p)$
Maintain the frontier as a priority queue sorted by $I(p)$

Preferential Crawlers

Selective bias toward some pages, eg. most "relevant"/topical, closest to seeds, most popular/largest PageRank, unknown servers, highest rate/amount of change, etc...

Focused crawlers

Supervised learning: classifier based on labeled examples drives the crawler frontier

Topical crawlers

Best-first search based on $similarity(topic, parent)$

Adaptative crawlers

- Reinforcement learning
- Evolutionary algorithms/artificial life

Preferential Crawling Algorithms: Examples

Best-N-First: Priority queue sorted by preference, explore top N at a time

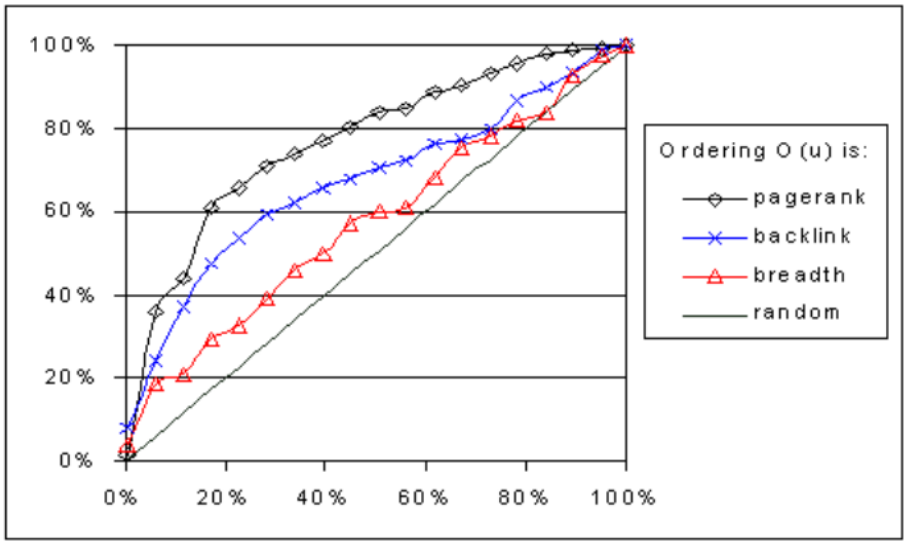
PageRank: Priority queue sorted PageRank

SharkSearch: Priority queue sorted by combination of similarity, anchor text, similarity of parent, etc.

InfoSpiders: Adaptive distributed algorithm using an evolving population of learning agents

Preferential Crawlers: PageRank to Prioritize Crawl

For I(p) = PageRank (estimated based on pages crawled so far), we can **find high backlink pages faster** than a breadth-first crawler.



% pages crawled with backlinks above 100 as function of % of pages crawled.

Cho, Garcia-Molina & Page, *Efficient Crawling Through URL Ordering*, WWW 1998.
http://ilpubs.stanford.edu:8090/347/1/1998-51.pdf

Preferential Crawlers: Figures of Merit

$$Precision \approx \frac{|p : crawled(p) \wedge I(p) > threshold|}{|p : crawled(p)|}$$

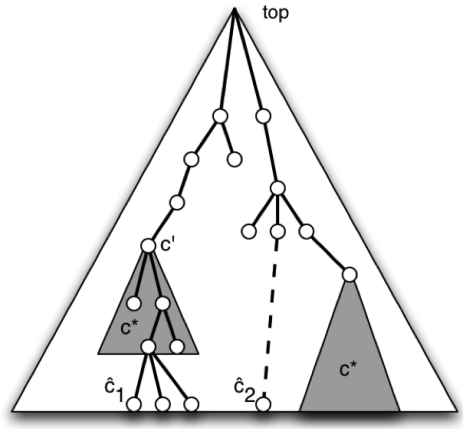
$$Recall \approx \frac{|p : crawled(p) \wedge I(p) > threshold|}{|p : I(p) > threshold|}$$

Focused Crawlers: Basic Idea

Train Naïve-Bayes classifier based on example pages in desired set of topics, $c*$. For each class $c \in c*$ we compute a Score for p

$$Pr(c*|p) = \sum_{c \in c*} Pr(c|p)$$

Decision is based on defined threshold.



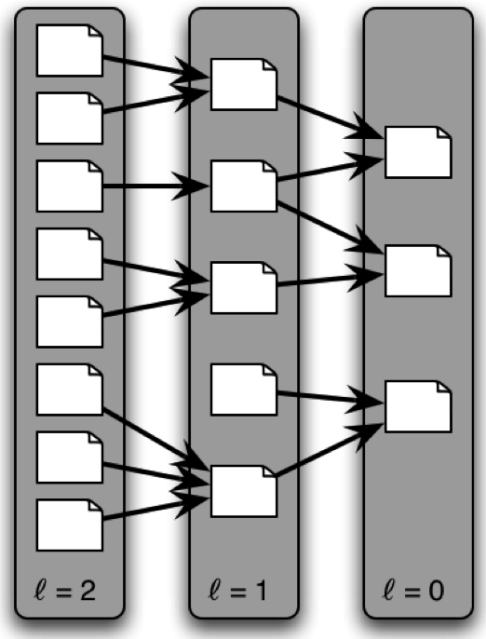
The ODP topic hierarchy

Focused Crawlers

- Can have multiple topics with as many classifiers, with scores appropriately combined (Chakrabarti et al. 1999)
- Can use alternative classifier algorithms to naïve-Bayes.
  - SVM and neural nets have reportedly performed better (Pant & Srinivasan 2005)

Context-focused Crawlers

- Classifiers trained based on link distance from relevant targets
  - l=0 is topic of interest
  - l=1 link to topic of interest
  - Etc.
- Initially needs a back-crawl from seeds (or known targets) to train classifiers to estimate distance
- Links in frontier prioritized based on estimated distance from targets
- Outperforms standard focused crawler empirically



Context graph with 3 layers

Topical Crawlers

No labeled examples

All we have is a topic (query, description, keywords) and a set of seed pages (not necessarily relevant).

Must predict relevance of unvisited links to prioritize
Original idea: Menczer 1997, Menczer & Belew 1998

Topical Locality

Topical locality is a necessary condition for a topical crawler to work, and for surfing to be a worthwhile activity for humans
Links must encode semantic information, i.e. say something about neighbor pages, not be random
It is also a sufficient condition if we start from "good" seed pages
Indeed we know that Web topical locality is strong :

- Indirectly (crawlers work and people surf the Web)
- From direct measurements (Davison 2000; Menczer 2004, 2005)

Simplest topical crawler: Naïve Best-First


```
BestFirst(topic, seed_urls) {
   foreach link (seed_urls) {
      enqueue(frontier, link);
   }
   while (len(frontier) > 0 and visited < MAX_PAGES) {
      link := dequeue_link_with_max_score(frontier);
      doc := fetch_new_document(link);
      score :=  sim(topic, doc);
      foreach outlink (extract_links(doc)) {
         if (len(frontier) >= MAX_BUFFER) {
            dequeue_link_with_min_score(frontier);
         }
         enqueue(frontier, outlink, score);
} } }
```

Frontier is priority queue based on text similarity between topic and parent page.

Best-first Variations

Correspond to different ways to score unvisited URLs:

- Giving more importance to certain HTML markup in parent page
- Extending text representation of parent page with anchor text from "grandparent" pages
- Limiting link context to less than entire page
- Exploiting topical locality (co-citation)

Any of these can be (and many have been) combined

Outline

Crawler ethics and Conflicts

Crawlers can cause trouble, even unwillingly, if not properly designed to be "polite" and "ethical"
For example, sending too many requests in rapid succession to a single server can amount to a Denial of Service (DoS) attack!

- Server administrator and users will be upset
- Crawler developer/admin IP address may be blacklisted

Crawler Etiquette (important!)

Identify yourself

- Use User-Agent HTTP header to identify crawler, website with description of crawler and contact information for crawler developer
- Use From HTTP header to specify crawler developer email
- Do not disguise crawler as a browser by using their User-Agent string

Always check that HTTP requests are successful, and in case of error, use HTTP error code to determine and immediately address problem

Pay attention to anything that may lead to too many requests to any one server, even unwillingly, e.g.:

- redirection loops
- spider traps

Crawler Etiquette (important!)

Spread the load, do not overwhelm a server

- Make sure that no more than some max. number of requests to any single server per unit time, say less than 1/second

Honor the Robot Exclusion Protocol

- A server can specify which parts of its document tree any crawler is or is not allowed to crawl by a file named `robots.txt` placed in the HTTP root directory, e.g. `http://www.ist.utl.pt/robots.txt`, `http://www.google.com/robots.txt`
- Crawler should always check, parse, and obey this file before sending any requests to a server
- More info at: `http://www.robotstxt.org/wc/exclusion.html`

More on Robot Exclusion

- Make sure URLs are canonical before checking against `robots.txt`
- Avoid fetching `robots.txt` for each request to a server by caching its policy as relevant to this crawler
- `Sitemap` and `sitemap.xml` may help (e.g. `https://support.google.com/webmasters/answer/183668?hl=en`)
- Let's look at some examples to understand the protocol

Apple.com

http://www.apple.com/robots.txt

```
# robots.txt for http://www.apple.com/
User-agent: *
Disallow:
```

everyone welcome to crawl anything they want!

IST.UTL.PT

http://www.ist.utl.pt/robots.txt

```
User-agent: *
Disallow: /img/
Disallow: /css/
Disallow: /inc/
Disallow: /lib/
Disallow: /newscache/
Disallow: /researchersCache/
```

everyone welcome to crawl anything except specified directories.

Robots.txt Compliance

Is compliance with robot exclusion a matter of law?

No! Compliance is voluntary, but if you do not comply, you may be blocked

Someone (unsuccessfully) sued Internet Archive over a `robots.txt` related issue

Disguised Crawlers

Some crawlers Disguise Themselves

Using false User-Agent

Randomizing access frequency to look like a human/browser

Why? Example: click fraud for ads

Disguised Servers

Cloaking:

present different content based on User-Agent
E.g. stuff keywords on version of page shown to search engine crawler

Search engines do not look kindly on this type of "spamdexing" and remove from their index sites that perform such abuse

Case of bmw.de made the news:
http://en.wikipedia.org/wiki/Spamdexing

Gray Areas for Crawler Ethics

If you write a crawler that unwillingly follows links to ads, are you just being careless, or are you violating terms of service, or are you violating the law by defrauding advertisers?

- Is non-compliance with Google's `robots.txt` in this case equivalent to click fraud?

If you write a browser extension that performs some useful service, should you comply with robot exclusion?

Need Crawling Code?

- Just google "python crawler"
- Large-scale open source crawlers:

  Nutch: `http://nutch.apache.org/`

  Heritrix: `http://crawler.archive.org/`

Questions?