Ch. 8: Web Crawling

By Filippo Menczer
Indiana University School of Informatics

in Web Data Mining by Bing Liu
Springer, 2007
Outline

• Motivation and taxonomy of crawlers
• Basic crawlers and implementation issues
• Universal crawlers
• Preferential (focused and topical) crawlers
• Evaluation of preferential crawlers
• Crawler ethics and conflicts
• New developments: social, collaborative, federated crawlers
Q: How does a search engine know that all these pages contain the query terms?
A: Because all of those pages have been crawled.
Crawler: basic idea
Many names

- Crawler
- Spider
- Robot (or bot)
- Web agent
- Wanderer, worm, ...
- And famous instances: googlebot, scooter, slurp, msnbot, ...
Motivation for crawlers

- Support universal search engines (Google, Yahoo, MSN/Windows Live, 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...
- ... Can you think of some others?...
A crawler within a search engine
One taxonomy of crawlers

- Universal crawlers
- Preferential crawlers
- Focused crawlers
- Topical crawlers
  - Adaptive topical crawlers
  - Evolutionary crawlers
  - Reinforcement learning crawlers
- Static crawlers
  - Best-first
  - PageRank
  - etc...

- Many other criteria could be used:
  - Incremental, Interactive, Concurrent, Etc.
Outline

• Motivation and taxonomy of crawlers
• Basic crawlers and implementation issues
• Universal crawlers
• Preferential (focused and topical) crawlers
• Evaluation of preferential crawlers
• Crawler ethics and conflicts
• New developments: social, collaborative, federated crawlers
Basic crawlers

- 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
Graph traversal (BFS or DFS?)

- **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 stuff…

- **Depth First Search**
  - Implemented with STACK (LIFO)
  - Wander away (“lost in cyberspace”)
A basic crawler in Perl

- Queue: a FIFO list (shift and push)

```perl
my @frontier = read_seeds($file);
while (@frontier && $tot < $max) {
    my $next_link = shift @frontier;
    my $page = fetch($next_link);
    add_to_index($page);
    my @links = extract_links($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
  – 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 overhead of extra Internet requests
More 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, and other errors
More implementation issues: Parsing

- 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
  - E.g. tidy.sourceforge.net
- Must pay attention to HTML entities and unicode in text
- What to do with a growing number of other formats?
  - Flash, SVG, RSS, AJAX...
More implementation issues

- **Stop words**
  - Noise words that do not carry meaning should be eliminated ("stopped") before they are indexed
  - E.g. in English: AND, THE, A, AT, OR, ON, FOR, etc...
  - Typically syntactic markers
  - Typically the most common terms
  - Typically kept in a negative dictionary
    - 10-1,000 elements
    - E.g. [http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words](http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words)
  - Parser can detect these right away and disregard them
More implementation issues

Conflation and thesauri

• Idea: improve recall by merging words with same meaning

1. We want to ignore superficial morphological features, thus merge semantically similar tokens
   - \{student, study, studying, studious\} => studi

2. We can also conflate synonyms into a single form using a thesaurus
   - 30-50% smaller index
   - Doing this in both pages and queries allows to retrieve pages about ‘automobile’ when user asks for ‘car’
   - Thesaurus can be implemented as a hash table
More implementation issues

- **Stemming**
  - Morphological conflation based on rewrite rules
  - Language dependent!
  - Porter stemmer very popular for English
    - http://www.tartarus.org/~martin/PorterStemmer/
    - Context-sensitive grammar rules, eg:
      - “IES” except (“EIES” or “AIES”) --> “Y”
    - Versions in Perl, C, Java, Python, C#, Ruby, PHP, etc.
  - Porter has also developed Snowball, a language to create stemming algorithms in any language
    - http://snowball.tartarus.org/
    - Ex. Perl modules: Lingua::Stem and Lingua::Stem::Snowball
More implementation issues

- **Static vs. dynamic pages**
  - Is it worth trying to eliminate dynamic pages and only index static pages?
  - Examples:
    - [http://www.census.gov/cgi-bin/gazetteer](http://www.census.gov/cgi-bin/gazetteer)
    - [http://informatics.indiana.edu/research/colloquia.asp](http://informatics.indiana.edu/research/colloquia.asp)
    - [http://www.imdb.com/name/nm0578801/](http://www.imdb.com/name/nm0578801/)
  - Why or why not? How can we tell if a page is dynamic? What about 'spider traps'?
  - What do Google and other search engines do?
More 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**
  - **Relative URL**: int1.html
  - **Relative URL**: /US/
More implementation issues

- **URL canonicalization**
  - All of these:
    - `http://www.cnn.com/TECH`
    - `http://WWW.CNN.COM/TECH/`
  - Are really equivalent to this canonical form:
  - 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 :80
More on Canonical URLs

- Some transformation are trivial, for example:

  × http://informatics.indiana.edu
  ✓ http://informatics.indiana.edu/

  × http://informatics.indiana.edu/index.html#fragment
  ✓ http://informatics.indiana.edu/index.html

  × http://informatics.indiana.edu/dir1/./../dir2/
  ✓ http://informatics.indiana.edu/dir2/

  × http://informatics.indiana.edu/%7Efil/
  ✓ http://informatics.indiana.edu/~fil/

  × http://INFORMATICS.INDIANA.EDU/fil/
  ✓ http://informatics.indiana.edu/fil/
More on Canonical URLs

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

1. Removing default file name
   - ✓ http://informatics.indiana.edu/fil/index.html
   - × http://informatics.indiana.edu/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’

2. Trailing directory
   - × http://informatics.indiana.edu/fil
   - ✓ http://informatics.indiana.edu/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?
More 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
  - Only heuristic defensive measures:
    - 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 can detect
More implementation issues

- **Page repository**
  - Naïve: 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 databases such as Berkeley DB
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

• Motivation and taxonomy of crawlers
• Basic crawlers and implementation issues
• Universal crawlers
• Preferential (focused and topical) crawlers
• Evaluation of preferential crawlers
• Crawler ethics and conflicts
• New developments: social, collaborative, federated crawlers
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
Large-scale universal crawlers

- Two major issues:
  1. **Performance**
     - Need to scale up to billions of pages
  2. **Policy**
     - Need to trade-off coverage, freshness, and bias (e.g. toward “important” pages)
Large-scale crawlers: scalability

- Need to minimize overhead of DNS lookups
- Need to optimize utilization of network bandwidth and disk throughput (I/O is bottleneck)
- Use asynchronous sockets
  - Multi-processing or multi-threading do not scale up to billions of pages
  - Non-blocking: hundreds of network connections open simultaneously
  - Polling socket to monitor completion of network transfers
Several parallel queues to spread load across servers (keep connections alive)

DNS server using UDP (less overhead than TCP), large persistent in-memory cache, and prefetcing

High-level architecture of a scalable universal crawler

Optimize use of network bandwidth

Huge farm of crawl machines

Optimize disk I/O throughput
Universal crawlers: Policy

• 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
This assumes we know the size of the entire the Web. Do we? Can you define “the size of the Web”?
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
  - Brewington & Cybenko; Cho, Garcia-Molina & Page
- 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
- What are appropriate priority measures in these cases? Approximations?
Breadth-first crawlers

- BF crawler tends to crawl high-PageRank pages very early
- Therefore, BF crawler is a good baseline to gauge other crawlers
- But why is this so?

Najork and Weiner 2001
Bias of breadth-first crawlers

- The structure of the Web graph is very different from a random network
- Power-law distribution of in-degree
- Therefore there are hub pages with very high PR and many incoming links
- These are attractors: you cannot avoid them!
Outline

• Motivation and taxonomy of crawlers
• Basic crawlers and implementation issues
• Universal crawlers
• Preferential (focused and topical) crawlers
• Evaluation of preferential crawlers
• Crawler ethics and conflicts
• New developments: social, collaborative, federated crawlers
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 \( I(p) \)
- Possible figures of merit:
  - Precision ~ \[
  \frac{\left| p: \text{crawled}(p) \& I(p) > \text{threshold} \right|}{\left| p: \text{crawled}(p) \right|}
  \]
  - Recall ~ \[
  \frac{\left| p: \text{crawled}(p) \& I(p) > \text{threshold} \right|}{\left| p: I(p) > \text{threshold} \right|}
  \]
Preferential crawlers

- Selective bias toward some pages, e.g., 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

- **Topical crawlers**
  - Best-first search based on similarity(topic, parent)
  - Adaptive crawlers
    - Reinforcement learning
    - Evolutionary algorithms/artificial life
Preferential crawling algorithms: Examples

- **Breadth-First**
  - Exhaustively visit all links in order encountered

- **Best-N-First**
  - Priority queue sorted by similarity, explore top N at a time
  - Variants: DOM context, hub scores

- **PageRank**
  - Priority queue sorted by keywords, PageRank

- **SharkSearch**
  - Priority queue sorted by combination of similarity, anchor text, similarity of parent, etc. (powerful cousin of FishSearch)

- **InfoSpiders**
  - Adaptive distributed algorithm using an evolving population of learning agents
Preferential crawlers: Examples

- For $I(p) = \text{PageRank}$ (estimated based on pages crawled so far), we can find high-PR pages faster than a breadth-first crawler (Cho, Garcia-Molina & Page 1998)
Focused crawlers: Basic idea

- Naïve-Bayes classifier based on example pages in desired topic, \( c^* \)
- \( \text{Score}(p) = \Pr(c^* | p) \)
  - Soft focus: frontier is priority queue using page score
  - Hard focus:
    - Find best leaf \( \hat{c} \) for \( p \)
    - If an ancestor \( c' \) of \( \hat{c} \) is in \( c^* \) then add links from \( p \) to frontier, else discard
  - Soft and hard focus work equally well empirically

Example: Open Directory
Focused crawlers

- Can have multiple topics with as many classifiers, with scores appropriately combined (Chakrabarti et al. 1999)
- Can use a distiller to find topical hubs periodically, and add these to the frontier
- Can accelerate with the use of a critic (Chakrabarti et al. 2002)
- Can use alternative classifier algorithms to naïve-Bayes, e.g. SVM and neural nets have reportedly performed better (Pant & Srinivasan 2005)
Context-focused crawlers

- Same idea, but multiple classes (and classifiers) based on link distance from relevant targets
  - $\ell=0$ is topic of interest
  - $\ell=1$ 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
Topical crawlers

• All we have is a topic (query, description, keywords) and a set of seed pages (not necessarily relevant)
• No labeled examples
• Must predict relevance of unvisited links to prioritize
• Original idea: Menczer 1997, Menczer & Belew 1998
Example: myspiders.informatics.indiana.edu
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)
Quantifying topical locality

• Different ways to pose the question:
  - How quickly does semantic locality decay?
  - How fast is topic drift?
  - How quickly does content change as we surf away from a starting page?

• To answer these questions, let us consider exhaustive breadth-first crawls from 100 topic pages
The "link-cluster" conjecture

- Connection between semantic topology (relevance) and link topology (hypertext)
  - $G = \text{Pr}[\text{rel}(p)] \sim$ fraction of relevant/topical pages (topic generality)
  - $R = \text{Pr}[\text{rel}(p) \mid \text{rel}(q) \ \text{AND} \ \text{link}(q,p)] \sim$ cond. prob. Given neighbor on topic

- Related nodes are clustered if $R > G$
  - Necessary and sufficient condition for a random crawler to find pages related to start points
  - Example: 2 topical clusters with stronger modularity within each cluster than outside
Link-cluster conjecture

- Stationary hit rate for a random crawler:

\[ \eta(t + 1) = \eta(t) \cdot R + (1 - \eta(t)) \cdot G \geq \eta(t) \]

\[ \eta \xrightarrow{t \to \infty} \eta^* = \frac{G}{1 - (R - G)} \]

\[ \eta^* > G \iff R > G \]

\[ \frac{\eta^*}{G} - 1 = \frac{R - G}{1 - (R - G)} \]
Link-cluster conjecture

- Preservation of semantics (meaning) across links
- 1000 times more likely to be on topic if near an on-topic page!

\[
R(q,\delta) \equiv \frac{\Pr[\text{rel}(p) \mid \text{rel}(q) \land \|\text{path}(q,p)\| \leq \delta]}{G(q) \Pr[\text{rel}(p)]} \\
L(q,\delta) \equiv \frac{\sum_{\|\text{path}(q,p)\| \leq \delta} 1}{\sum_{\|\text{path}(q,p)\| \leq \delta} 1}
\]
The “link-content” conjecture

- Correlation of lexical (content) and linkage topology
- $L(\delta)$: average link distance
- $S(\delta)$: average content similarity to start (topic) page from pages up to distance $\delta$
- Correlation $\rho(L, S) = -0.76$

$$S(q, \delta) \equiv \frac{\sum_{\{p: \|path(q, p)\| \leq \delta\}} sim(q, p)}{\{ p : \|path(q, p)\| \leq \delta\}}$$

Yahoo crawl data

exponential decay fit
noise level
Heterogeneity of link-content correlation

\[ S = c + (1 - c)e^{aL^b} \]

.com has more drift

Signif. diff. \( a \) only (\( \alpha=0.05 \))

Signif. diff. \( a \) & \( b \) (\( \alpha=0.05 \))
Topical locality-inspired tricks for topical crawlers

- **Co-citation (a.k.a. sibling locality):** A and C are good hubs, thus A and D should be given high priority.
- **Co-reference (a.k.a. bibliographic coupling):** E and G are good authorities, thus E and H should be given high priority.
Correlations between different similarity measures

- **Semantic similarity** measured from ODP, correlated with:
  - **Content similarity**: TF or TF-IDF vector cosine
  - **Link similarity**: Jaccard coefficient of (in+out) link neighborhoods

- Correlation overall is significant but weak

- Much stronger topical locality in some topics, e.g.:
  - Links very informative in news sources
  - Text very informative in recipes
Naïve Best-First

Simplest topical crawler: Frontier is priority queue based on text similarity between topic and parent page

```plaintext
BestFirst(topic, seed_urls) {
    foreach link (seed_urls) {
        enqueue(frontier, link);
    }
    while (#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 (#frontier >= MAX_BUFFER) {
                dequeue_link_with_min_score(frontier);
            }
            enqueue(frontier, outlink, score);
        }
    }
}
```
Best-first variations

- Many in literature, mostly stemming from different ways to score unvisited URLs. E.g.:
  - Giving more importance to certain HTML markup in parent page
  - Extending text representation of parent page with anchor text from “grandparent” pages (SharkSearch)
  - Limiting link context to less than entire page
  - Exploiting topical locality (co-citation)
  - Exploration vs exploitation: relax priorities

- Any of these can be (and many have been) combined
Link context based on text neighborhood

- Often consider a fixed-size window, e.g., 50 words around anchor
- Can weigh links based on their distance from topic keywords within the document (InfoSpiders, Clever)
- Anchor text deserves extra importance
Link context based on DOM tree

- Consider DOM subtree rooted at parent node of link’s `<a>` tag
- Or can go further up in the tree (Naïve Best-First is special case of entire document body)
- Trade-off between noise due to too small or too large context tree (Pant 2003)
DOM context

Link score = linear combination between page-based and context-based similarity score
Co-citation: hub scores

Link score_{hub} = linear combination between link and hub score
Combining DOM context and hub scores

Experiment based on 159 ODP topics (Pant & Menczer 2003)

Split ODP URLs between seeds and targets

Add 10 best hubs to seeds for 94 topics
Exploration vs Exploitation

- Best-$N$-First (or BFS$N$)
- Rather than re-sorting the frontier every time you add links, be lazy and sort only every $N$ pages visited
- Empirically, being less greedy helps crawler performance significantly: escape “local topical traps” by exploring more

Pant et al. 2002
InfoSpiders

- A series of intelligent multi-agent topical crawling algorithms employing various adaptive techniques:
  - Evolutionary bias of exploration/exploitation
  - Selective query expansion
  - (Connectionist) reinforcement learning

Menczer & Belew 1998, 2000; Menczer et al. 2004
Outline

• Motivation and taxonomy of crawlers
• Basic crawlers and implementation issues
• Universal crawlers
• Preferential (focused and topical) crawlers
• Evaluation of preferential crawlers
• Crawler ethics and conflicts
• New developments: social, collaborative, federated crawlers
Evaluation of topical crawlers

• Goal: build “better” crawlers to support applications (Srinivasan & al. 2005)

• Build an unbiased evaluation framework
  - Define common tasks of measurable difficulty
  - Identify topics, relevant targets
  - Identify appropriate performance measures
    • Effectiveness: quality of crawler pages, order, etc.
    • Efficiency: separate CPU & memory of crawler algorithms from bandwidth & common utilities
Evaluation corpus = ODP + Web

- Automate evaluation using edited directories
- Different sources of relevance assessments

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---

**Evaluation corpus = ODP + Web**

**• Automate evaluation using edited directories**

**• Different sources of relevance assessments**

---
Topics and Targets

- topic level ~ specificity
- depth ~ generality

Ch. 8 Web Crawling by Filippo Menczer
Tasks

Start from seeds, find targets and/or pages similar to target descriptions

Back-crawl from targets to get seeds
Target based performance measures

Q: What assumption are we making?  A: Independence!...
### Performance Matrix

<table>
<thead>
<tr>
<th>Target Pages</th>
<th>Target Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sum \sigma_c(p, D_d) ) for ( p \in S_c^t )</td>
<td>( \sum \sigma_c(p, D_d) ) for ( p \in S_c^t )</td>
</tr>
</tbody>
</table>

- **Precision**
  - \( \frac{|S_c^t \cap T_d|}{|T_d|} \)
  - \( \frac{|S_c^t \cap T_d|}{|S_c^t|} \)

- **Recall**
  - \( \frac{|S_c \cap T_d|}{|T_d|} \)
  - \( \frac{|S_c \cap T_d|}{|S_c|} \)

The matrix represents a performance matrix for web crawling, showing the relationship between target pages and target descriptions at different depths.
Crawling evaluation framework

- Keywords
- Seed URLs

Main

Crawler 1
Logic

Crawler N
Logic

Private Data Structures (limited resource)

Common Data Structures

Concurrent Fetch/Parse/Stem Modules

HTTP

HTTP

HTTP

Web
Using framework to compare crawler performance

Pages crawled vs Average target page recall for InfoSpiders, BFS256, and BreadthFirst crawlers.
Efficiency & scalability

![Graph showing link frontier size vs. performance/cost for different crawling algorithms: InfoSpiders, BreadthFirst, BFS, and Shark. The graph indicates varying performance and cost implications across different link frontier sizes.](image-url)
Topical crawler performance depends on topic characteristics

\[ \begin{align*}
C &= \text{target link cohesiveness} \\
A &= \text{target authoritativeness} \\
P &= \text{popularity (topic kw generality)} \\
L &= \text{seed-target similarity}
\end{align*} \]

<table>
<thead>
<tr>
<th>Crawler</th>
<th>Target pages</th>
<th>Target descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>BreadthFirst</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>BFS-1</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>BFS-256</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>InfoSpiders</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Outline

• Motivation and taxonomy of crawlers
• Basic crawlers and implementation issues
• Universal crawlers
• Preferential (focused and topical) crawlers
• Evaluation of preferential crawlers
• New developments
New developments: social, collaborative, federated crawlers

- **Idea**: go beyond the “one-fits-all” model of centralized search engines
- Extend the search task to anyone, and distribute the crawling task
- Each search engine is a **peer agent**
- Agents collaborate by **routing queries and results**
6S: Collaborative Peer Search

- WWW
- bookmarks
- Crawler
- Index
- Peer
- local storage

Emerging communities

Data mining & referral opportunities

A

B

C

query

hit

hit

hit

query

query

query

query
Basic idea: Learn based on prior query/response interactions
Learning about other peers

\[ w_{(10, Lama)}^{f}(t+1) = (1 - \gamma) \times w_{(10, Lama)}^{f}(t) + \gamma \times \left( \frac{S_{10} + 1}{S_{1} + 1} - 1 \right) \]

Neighbor Hits:
- score = 0.96, www.alaa.org/lama/
- score = 0.7, www.nlc.state.ne.us/
- ...

TF analysis for top N hits

Average score of local hits

Average score of neighbor's hits

Learning rate

Focused Profile \( w^{f} \)

Expanded Profile \( w^{e} \)
Need crawling code?

- Reference C implementation of HTTP, HTML parsing, etc
  - w3c-libwww package from World-Wide Web Consortium: www.w3c.org/Library/
- LWP (Perl)
  - http://search.cpan.org/~gaas/libwww-perl-5.804/
- Open source crawlers/search engines
  - Nutch: http://www.nutch.org/ (Jakarta Lucene: jakarta.apache.org/lucene/)
  - Heretrix: http://crawler.archive.org/
  - WIRE: http://www.cwr.cl/projects/WIRE/
  - Terrier: http://ir.dcs.gla.ac.uk/terrier/
- Open source topical crawlers, Best-First-N (Java)
  - http://informatics.indiana.edu/fil/IS/JavaCrawlers/
- Evaluation framework for topical crawlers (Perl)
  - http://informatics.indiana.edu/fil/IS/Framework/