


<isweb> ISWeb – Information Systems & Semantic Web  
University of Koblenz • Landau

## Chapter 5

### Web Crawling

Sergej Sizov  
Web Mining  
Summer term 2007



UNIVERSITÄT  
KOBLENZ • LANDAU

### Basic crawler operation

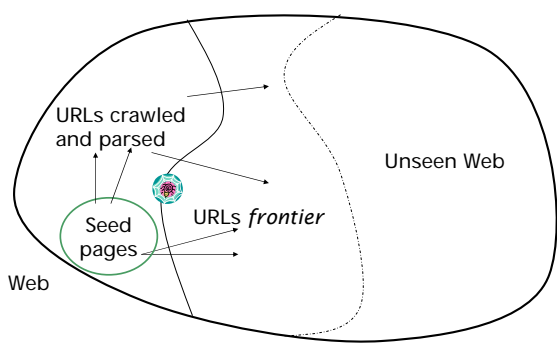
Problem: no catalog of all accessible URLs on the Web

Solution:

- ♦ Begin with known "seed" pages
- ♦ Fetch and parse them
  - Extract URLs they point to
  - Place the extracted URLs on a queue
- ♦ Fetch each URL on the queue and repeat

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 5 – Web Crawling 5-2

### Crawling picture



Web

Seed pages

URLs frontier

URLs crawled and parsed

Unseen Web

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 5 – Web Crawling 5-3

### Simple picture – complications

Web crawling isn't feasible with one machine

- ♦ All of the above steps distributed

Even non-malicious pages pose challenges

- ♦ Latency/bandwidth to remote servers vary
- ♦ Webmasters' stipulations
  - How "deep" should you crawl a site's URL hierarchy?
- ♦ Site mirrors and duplicate pages

Malicious pages

- ♦ Spam pages
- ♦ Spider traps – incl dynamically generated

Politeness – don't hit a server too often

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 5 – Web Crawling 5-4

### What any crawler *must* do

Be Polite: Respect implicit and explicit politeness considerations for a website

- ♦ Only crawl pages you're allowed to, respect "robot exclusion" conventions
- ♦ Be Robust: Be immune to spider traps and other malicious behavior from web servers

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 5 – Web Crawling 5-5

### What any crawler *should* do

Be capable of distributed operation: designed to run on multiple distributed machines

Be scalable: designed to increase the crawl rate by adding more machines

Performance/efficiency: permit full use of available processing and network resources

Eliminate duplicates, recognize redundancy

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 5 – Web Crawling 5-6

### What any crawler *should* do

Fetch pages of "higher quality" first

Continuous operation: Continue fetching fresh copies of a previously fetched page

Extensible: Adapt to new data formats, protocols

### Explicit and implicit politeness

Explicit politeness: specifications from webmasters on what portions of site can be crawled

- ♦ robots.txt
- ♦ meta tags in HTML documents

Implicit politeness: even with no specification, avoid hitting any site too often

### Robots.txt

Protocol for giving spiders ("robots") limited access to a website, originally from 1994

[www.robotstxt.org/wc/norobots.html](http://www.robotstxt.org/wc/norobots.html)

Website announces its request on what can(not) be crawled

- ♦ For a URL, create a file `URL/robots.txt`
- ♦ This file specifies access restrictions

Only use blank lines to separate different User-agent disallowed directories.

One directory per "Disallow" line.

No regex patterns in directories.

### Robot Exclusion Protocol Examples

Exclude specific directories:

```
User-agent: *
Disallow: /tmp/
Disallow: /cgi-bin/
Disallow: /users/paranoid/
```

Exclude a specific robot:

```
User-agent: GoogleBot
Disallow: /
```

Allow a specific robot:

```
User-agent: GoogleBot
Disallow:

User-agent: *
Disallow: /
```

### Robots META Tag

Include META tag in HEAD section of a specific HTML document.

- ♦ `<meta name="robots" content="none">`

Content value is a pair of values for two aspects:

- ♦ `index | noindex`: Allow/disallow indexing of this page.
- ♦ `follow | nofollow`: Allow/disallow following links on this page.

Special values:

- ♦ `all` = `index, follow`
- ♦ `none` = `noindex, nofollow`

Examples:

```
<meta name="robots" content="noindex, follow">
<meta name="robots" content="index, nofollow">
<meta name="robots" content="none">
```

### Processing steps in crawling

Pick a URL from the frontier

Fetch the document at the URL

Parse the URL

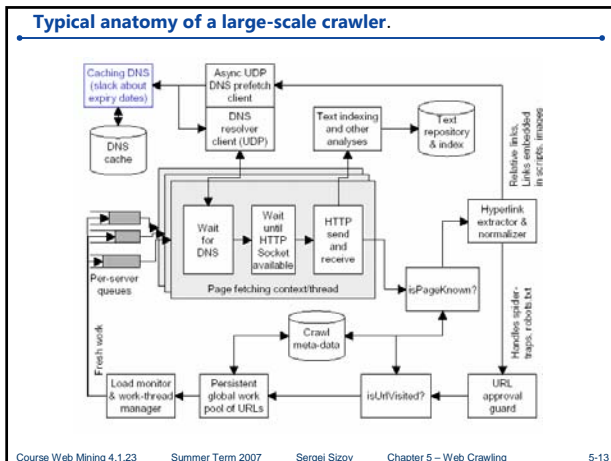
- ♦ Extract links from it to other docs (URLs)

Check if URL has content already seen

- ♦ If not, add to indexes

For each extracted URL

- ♦ Ensure it passes certain URL filter tests
- ♦ Check if it is already in the frontier (duplicate URL elimination)



### URL Syntax

A URL has the following syntax:

- <scheme>://<authority> <path>?<query>#<fragment>

An *authority* has the syntax:

- <host>[:<port-number>]

A *query* passes variable values from an HTML form and has the syntax:

- <variable>=<value>&<variable>=<value>...

A *fragment* is also called a *reference* or a *ref* and is a pointer within the document to a point specified by an anchor tag of the form:

- <A NAME="<fragment>">

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 5 – Web Crawling 5-14

### Parsing: URL normalization

When a fetched document is parsed, some of the extracted links are *relative* URLs

E.g., at [http://en.wikipedia.org/wiki/Main\\_Page](http://en.wikipedia.org/wiki/Main_Page) we have a relative link to [/wiki/Wikipedia:General\\_disclaimer](/wiki/Wikipedia:General_disclaimer) which is the same as the absolute URL [http://en.wikipedia.org/wiki/Wikipedia:General\\_disclaimer](http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer)

During parsing, must normalize (expand) such relative URLs

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 5 – Web Crawling 5-15

### DNS (Domain Name Server)

A lookup service on the internet

- Given a URL, retrieve its IP address
- Service provided by a distributed set of servers – thus, lookup latencies can be high (even seconds)

Common OS implementations of DNS lookup are *blocking*: only one outstanding request at a time

Solutions

- DNS caching
- Batch DNS resolver – collects requests and sends them out together

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 5 – Web Crawling 5-16

### URL frontier: two main considerations

**Politeness:** do not hit a web server too frequently

**Freshness:** crawl some pages more often than others

- E.g., pages (such as News sites) whose content changes often

These goals may conflict each other.

(E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 5 – Web Crawling 5-17

### Spider traps

Protecting from crashing on

- Ill-formed HTML
  - E.g.: page with 68 kB of null characters
- Misleading sites
  - indefinite number of pages dynamically generated by CGI scripts
  - paths of arbitrary depth created using soft directory links and path remapping features in HTTP server

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 5 – Web Crawling 5-18

### Spider Traps: Solutions

No automatic technique can be foolproof

Check for URL length

Guards

- Preparing regular crawl statistics
- Adding dominating sites to guard module
- Disable crawling active content such as CGI form queries
- Eliminate URLs with non-textual data types

### Avoiding repeated expansion of links on duplicate pages

Reduce redundancy in crawls

Duplicate detection

- different URLs – same document
- different hosts (mirrors) – (almost) identical document copies

Detecting exact duplicates

- Checking against MD5 digests of stored URLs
- Representing a relative link  $v$  (relative to aliases  $u1$  and  $u2$ ) as tuples  $(h(u1); v)$  and  $(h(u2); v)$

Detecting near-duplicates

- Even a single altered character will completely change the digest !
  - E.g.: date of update/ name and email of the site administrator
- Solution : Shingling

### Detecting duplicate pages

Duplicates waste crawler resources

We can quickly test for duplicates by computing a fingerprint for every page (e.g., MD5 hash)

- Make fingerprint long enough to make collisions very unlikely

Problem: pages may differ only in ads or other formatting changes

- Also an issue in counting changes for rate estimation

### Detecting approximate duplicates

Can compare the pages using known distance metrics e.g., edit distance

- Takes far too long when we have millions of pages!

One solution: create a sketch for each page that is much smaller than the page

Assume: we have converted page into a sequence of tokens

- Eliminate punctuation, HTML markup, etc

### Shingling

Given document  $D$ , a  $w$ -shingle is a contiguous subsequence of  $w$  tokens

The  $w$ -shingling  $S(D,w)$  of  $D$ , is the set of all  $w$ -shingles in  $D$

e.g.,  $D=(a,rose,is,a,rose,is,a,rose)$

$S(D,W) = \{(a,rose,is,a),(rose,is,a,rose), (is,a,rose,is)\}$

Can also define  $S(D,w)$  to be the bag of all shingles in  $D$

- We'll use sets in the lecture to keep it simple

### Resemblance

Let us define the resemblance of docs  $A$  and  $B$  as:

$$r_w(A, B) = \frac{|S(A,w) \cap S(B,w)|}{|S(A,w) \cup S(B,w)|}$$

In general,  $0 \leq r_w(A,B) \leq 1$

- Note  $r_w(A,A) = 1$
- But  $r_w(A,B)=1$  does not mean  $A$  and  $B$  are identical!

What is a good value for  $w$ ?

### Sketches

Set of all shingles is large

- Bigger than the original document

Can we create a document sketch by sampling only a few shingles?

Requirement

- Sketch resemblance should be a good estimate of document resemblance

Notation: Omit  $w$  from  $r_w$  and  $S(A,w)$

- $r(A,B) = r_w(A,B)$  and  $S(A) = S(A,w)$

### Choosing a sketch

Random sampling does not work!

- E.g., suppose we have identical documents A & B each with  $n$  shingles
- $M(A)$  = set of  $s$  shingles from A, chosen uniformly at random; similarly  $M(B)$

For  $s=1$ :  $E[|M(A) \cap M(B)|] = 1/n$

- But  $r(A,B) = 1$
- So the sketch overlap is an underestimate

Verify that this is true for any value of  $s$

### Choosing a sketch

Better method:  $M(A)$  is "smallest" shingle in A

- Assume set of shingles is totally ordered
- Also define  $M(AB) = \min(M(A) \cup M(B))$  = smallest shingle across A and B
- Define  $r'(A,B) = |M(AB) \cap M(A) \cap M(B)|$

For identical documents A & B

- $r'(A,B) = 1 = r(A,B)$

### Example

$|S(A)| = |S(B)| = 100$ ,  $|S(A) \cap S(B)| = 80$

$|S(A) \cup S(B)| = 100 + 100 - 80 = 120$

$r(A,B) = 80/120 = 2/3$

Suppose  $M(AB) = x$

- $x$  could be any one of 120 shingles

$\Pr[x \in M(A) \cap M(B)] = \Pr[x \in S(A) \cap S(B)] = 80/120 = 2/3$

$E[r'(A,B)] = E[|M(AB) \cap M(A) \cap M(B)|]$   
 $= 1(2/3) + 0(1/3) = 2/3$

In general, for any documents A and B, it's easy to generalize:

$E[r'(A,B)] = r(A,B)$

### Larger sketches

With a sketch of size 1, there is still a high probability of error

- We can improve the estimate by picking a larger sketch of size  $s$

For set  $W$  of shingles, let  $\text{MIN}_s(W)$  = set of  $s$  smallest shingles in  $W$

- Assume documents have at least  $s$  shingles

Define

- $M(A) = \text{MIN}_s(S(A))$
- $M(AB) = \text{MIN}_s(M(A) \cup M(B))$
- $r'(A,B) = |M(AB) \cap M(A) \cap M(B)| / s$

### Larger sketches

Easy to show that  $E[r'(A,B)] = r(A,B)$

- Similar to the case  $s = 1$
- Repeat argument for each element of  $M(AB)$

By increasing sample size ( $s$ ) we can make it very unlikely  $r'(A,B)$  is significantly different from  $r(A,B)$

- 100-200 shingles is sufficient in practice

### Corner case

Some docs have fewer than  $s$  shingles

$$\text{MIN}_s(W) = \begin{cases} \text{set of smallest } s \text{ elements of } W, & \text{if } |W| \geq s \\ W, & \text{otherwise} \end{cases}$$

$$M(A) = \text{MIN}_s(S(A))$$

$$M(AB) = \text{MIN}_s(M(A) \cup M(B))$$

$$r(A,B) = |M(AB) \cap M(A) \cap M(B)| / |M(AB)|$$

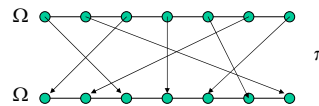
### Random permutations

One problem

- Test is biased towards “smaller” shingles
- May skew results

Solution

- Let  $\Omega$  be the set of all shingles
- Pick a permutation  $\pi: \Omega \rightarrow \Omega$  uniformly at random
- Define  $M(A) = \text{MIN}_s(\pi(S(A)))$



### Implementation

Size of each shingle is still large

- e.g., each shingle = 7 English words = 40-50 bytes
- 100 shingles = 4K-5K

Compute a fingerprint  $f$  for each shingle (e.g., Rabin fingerprint)

- 40 bits is usually enough to keep estimates reasonably accurate
- Fingerprint also eliminates need for random permutation

### Finding all near-duplicates

Naïve implementation makes  $O(N^2)$  sketch comparisons

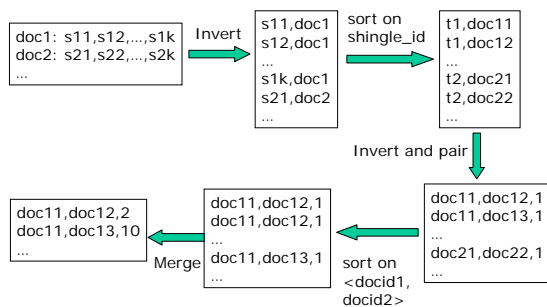
- Suppose  $N=100$  million

Divide-Compute-Merge (DCM)

- Divide data into batches that fit in memory
- Operate on individual batch and write out partial results in sorted order
- Merge partial results

Generalization of external sorting

### DCM Steps



### Finding all near-duplicates

Calculate a sketch for each document

For each document, write out the pairs  $\langle \text{shingle\_id}, \text{docid} \rangle$

Sort by shingle\_id (DCM)

In a sequential scan, generate triplets of the form  $\langle \text{docid1}, \text{docid2}, 1 \rangle$  for pairs of docs that share a shingle (DCM)

Sort on  $\langle \text{docid1}, \text{docid2} \rangle$  (DCM)

Merge the triplets with common docids to generate triplets of the form  $\langle \text{docid1}, \text{docid2}, \text{count} \rangle$  (DCM)

Output document pairs whose resemblance exceeds the threshold

### Some optimizations

"Invert and Pair" is the most expensive step

We can speed it up eliminating very common shingles

- Common headers, footers, etc.
- Do it as a preprocessing step

Also, eliminate exact duplicates up front

### Mirrors

Replication at a higher level

- Entire collections of documents
- Java APIs, perl manuals,...

Use document comparison as a building block

### Maintaining freshness

How often do web pages change?

What do we mean by freshness?

What strategy should we use to refresh pages?

### How often do pages change?

Cho et al (2000) experiment

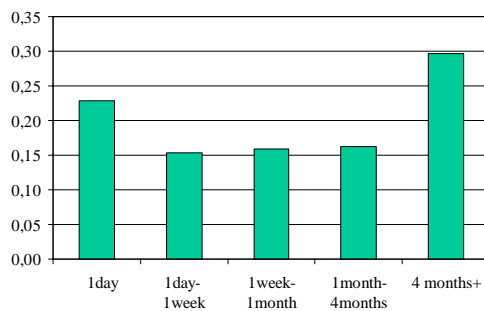
270 sites visited (with permission)

- identified 400 sites with highest "PageRank"
- contacted administrators

720,000 pages collected

- 3,000 pages from each site daily
- start at root, visit breadth first (get new & old pages)
- ran only 9pm - 6am, 10 seconds between site requests

### Average change interval



Source: Cho et al (2000)

### Modeling change

Assume changes to a web page are a sequence of random events that happen independently at a fixed average rate

Poisson process with parameter  $\lambda$

Let  $X(t)$  be a random variable denoting the number of changes in any time interval  $t$

$$\Pr[X(t)=k] = e^{-\lambda t}(\lambda t)^k/k! \quad \text{for } k = 0,1,\dots$$

"Memory-less" distribution

### Poisson processes

Let us compute the expected number of changes in unit time

$$E[X(1)] = \sum_k k e^{-\lambda} \lambda^k / k! = \lambda$$

$\lambda$  is therefore the average number of changes in unit time  
Called the rate parameter

### Time to next event

Let  $T$  be a random variable denoting the time to the next event

Verify that

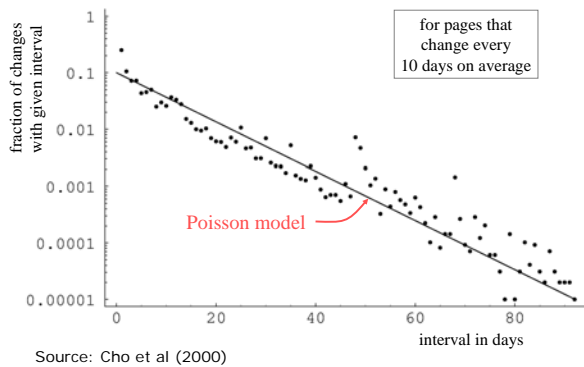
$$\Pr[T > t] = e^{-\lambda t} \quad (t > 0)$$

The corresponding density function is

$$f(t) = \lambda e^{-\lambda t}$$

The distribution of change intervals should follow an exponential distribution

### Change intervals of pages



### Change Metrics (1) - Freshness

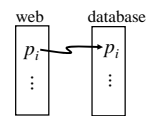
Freshness of element  $e_i$  at time  $t$  is

$$F(p_i; t) = \begin{cases} 1 & \text{if } e_i \text{ is up-to-date at time } t \\ 0 & \text{otherwise} \end{cases}$$

Freshness of the database  $S$  at time  $t$  is

$$F(S; t) = \frac{1}{N} \sum_{i=1}^N F(p_i; t)$$

(Assume "equal importance" of pages)



### Change Metrics (2) - Age

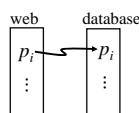
Age of element  $e_i$  at time  $t$  is

$$A(p_i; t) = \begin{cases} 0 & \text{if } e_i \text{ is up-to-date at time } t \\ t - (\text{modification time } p_i) & \text{otherwise} \end{cases}$$

Age of the database  $S$  at time  $t$  is

$$A(S; t) = \frac{1}{N} \sum_{i=1}^N A(p_i; t)$$

(Assume "equal importance" of pages)



### Change Metrics (3) - Delay

Suppose crawler visits page  $p$  at times  $\tau_0, \tau_1, \dots$

Suppose page  $p$  is updated at times  $t_1, t_2, \dots, t_k$  between times  $\tau_0$  and  $\tau_1$

The delay associated with update  $t_i$  is

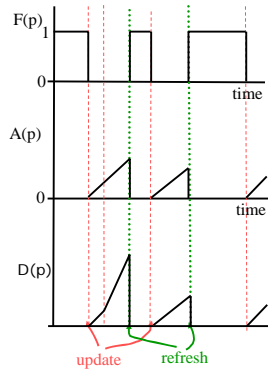
$$D(t_i) = \tau_1 - t_i$$

The total delay associated with the changes to page  $p$  is  $D(p) = \sum_i D(t_i)$

At time  $\tau_1$  the delay drops back to 0

Total crawler delay is sum of individual page delays

### Comparing the change metrics



### Resource optimization problem

Crawler can fetch M pages in time T  
 Suppose there are N pages  $p_1, \dots, p_N$  with change rates  $\lambda_1, \dots, \lambda_N$   
 How often should the crawler visit each page to minimize **delay**?  
 Assume uniform interval between visits to each page

### A useful lemma

**Lemma.**

For page p with update rate  $\lambda$ , if the interval between refreshes is  $\tau$ , the expected delay during this interval is  $\lambda\tau^2/2$ .

**Proof.**

Number of changes generated at between times t and t+dt =  $\lambda dt$

Delay for these changes =  $\tau - t$

$$\text{Total delay} = \int_0^\tau \lambda(\tau - t) dt = \lambda\tau^2/2$$



### Optimum resource allocation

Total number of accesses in time T = M

Suppose allocate  $m_i$  fetches to page  $p_i$

$$\text{Then } \sum_{i=1}^N m_i = M$$

Interval between fetches of page  $p_i = T/m_i$

$$\text{Delay for page } p_i \text{ between fetches} = \frac{\lambda_i(T/m_i)^2}{2}$$

$$\text{Total delay for page } p = \frac{\lambda_i T^2}{2m_i}$$

$$\text{Minimize } f = \sum_{i=1}^N \frac{\lambda_i T^2}{2m_i}$$

$$\text{subject to } g = M - \sum_{i=1}^N m_i = 0$$

### Method of Lagrange Multipliers

To maximize or minimize a function  $f(x_1, \dots, x_n)$  subject to the constraint  $g(x_1, \dots, x_n) = 0$

Introduce a new variable  $\mu$  and define

$$h = f - \mu g$$

Solve the system of equations:

$$\frac{\partial h}{\partial x_i} = 0 \quad \text{for } i = 1, \dots, n$$

$$g(x_1, \dots, x_n) = 0$$

n+1 equations in n+1 variables

### Optimum refresh policy

Applying the Lagrange multiplier method to our problem, we have

$$h = \sum_{i=1}^N \left( \frac{\lambda_i T^2}{2m_i} + \mu m_i \right) - \mu M$$

$$\frac{\partial h}{\partial m_i} = \frac{-\lambda_i T^2}{2m_i^2} + \mu m_i$$

$$m_i = \sqrt{\frac{\lambda_i T^2}{2\mu}} = k\sqrt{\lambda_i}$$

### Optimum refresh policy

To minimize delay, we must allocate to each page a number of visits proportional to the square root of its average rate of change

Very different answer to minimize the freshness and age metrics; see references.

### Estimating the rate parameter $\lambda$

Simple estimator

- Visit page N times in time interval T
- Suppose we see that page has changed X times
- Estimate  $\lambda = X/T$

What is the problem with this estimator?

### A better estimator

The page may have actually changed more than once between visits !

- We therefore tend to underestimate  $\lambda$

A better estimator is  $\lambda = \log\left(\frac{N+0.5}{N-X+0.5}\right)$

- For N=10, X=3, we get:
  - $\lambda = 0.3$  with the simple estimator
  - $\lambda = 0.34$  with the improved estimator.

Details in Cho and Garcia-Molina (2000)