WEB SEARCHING AND INFORMATION RETRIEVAL

Why the web?

In 1994, there were an estimated 100,000 documents on the Internet. By the year 2000, that number rose to over a billion. Keeping up with the processing and storing power needed encourages people to view the increase as following "Moore's law" [the often *misquoted* concept that "information doubles every 18 months" but Moore was discussing the processing power of computers, predicting that the power will increase every 18 months by a factor of at least $2^4 = 16$.] But can this rapid technological development continue? In November 1997, AltaVista processed 20 million searches/day. In 2004, Google reported 250 million web searches/day and estimated that all search engines collectively process about 500,000,000 searches/day.

To IR, the web poses many challenges because potentially one can store and retrieve tremendously large amounts of data from all sorts of formats. For instance:

- The volume of material: billions of items, more daily
- Items generated dynamically (stateless) or stored in databases
- Great variety: length, formats, quality control, purposes, etc.
- Users' range of experience and skill
- Economic modes to pay for access
- Staffing
- Storage

For example, in 2000 Google had a staff of 28 people, 14 of whom were the technical staff, all with Ph.D.s in computer science. Google's equipment consisted of 2,500 Linux machines, 80 terabytes of spinning disks and was installing 30 new machines daily! By 2002, Google grew to over 400 people; by 2004 about 1,400. [Larry Page, Google, March 2000]. It is still hiring!

Processing data on this scale requires considerable management bravado: programmers must be exceptionally well-trained and the code must be highly modular and single image. Preventive maintenance isn't worth it and broken computers aren't worth repairing: throw 'em away and get another computer. Run data mining programs on the access logs for analysis. Automate every related to customer service. [End-users don't care for it, but…]

To process 5.5 million searches a day, one would need 2,500 computers; it is estimated that Google has 100,000 computers.

In response, as by now you suspect, there are many approaches: subject hierarchies, such as Yahoo!'s human indexing; web crawling and automatic indexing, the most common approach used by InfoSeek, Lycos, AltaVista, Google, and most any other search engine; and mixed models, human directed web crawling and automatic indexing, e.g., iVia/NSDL. Whatever the model, there are common components of web search services.

Each consists of these components: a web crawler, an indexing system, and a search system. Each of these aspects is moderated by economic, scalability, and legal considerations.

Economic models

All web retrieval models, it can be argued, have some cost involved. The cost is not always economic: loss of bandwidth and storage space as well as end-user frustration are costs associated with cookies, ads, spam, and weighting of commercial sites. Lycos introduced the idea of free access with display advertisements. This results in users being mislead, the results distorted to suit the advertisers. Some costs are in money: InfoSeek offers unlimited access for a monthly fee. The cost of running a search engine company comes from fees, licensing of software, and specialized services.

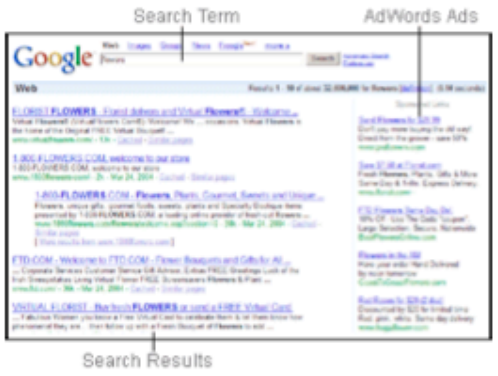
Google, at least, is forthcoming with these data, which can be useful for end-users interpretation (http://www.google.com/)

Google Advertising Programs

Home | About Google

For Advertisers: Google AdWords

Reach people when they are actively looking for information about your products and services online, and send targeted visitors directly to what you are offering. With AdWords cost-per-click pricing, it's easy to control costs—and you only pay when people click on your ad.



Apply Online: Create ads and start managing your account—takes just minutes. Learn more.

Contact Sales: Find out how our sales team can help you reach your online advertising goals. Learn more.

For Web Publishers: Google AdSense

Earn more revenue from your website, while providing visitors with a more rewarding online experience. Google AdSense™ automatically delivers text and image ads that are precisely targeted to your site and your site content—ads so well-matched, in fact, that your readers will actually find them useful. And when you add Google WebSearch to your site, AdSense delivers targeted ads to your search results pages too. With AdSense you earn more ad revenue with minimal effort—and no additional cost.

Apply Online: Maximize your revenue with relevant ads—takes just minutes. Learn more.

Contact Sales: If your site receives more than 20 million page views a month, our sales team can help customize the AdSense program for you. Learn more.

A web crawler is merely a computer application for downloading web pages. Given an initial set of seed URLs, the application downloads recursively every page that is linked from pages in the set. [Coincidentally, this is how email viruses work, too; when they find email addresses in web pages or download Microsoft Outlook's address book.] This technique is indiscriminate. There are also "focused" web crawlers, aka *web spiders*, which download only those pages that satisfy some criterion. How?

The basic algorithm for web crawling is simple: Let S be a set of URLs to pages waiting to be indexed. Initially, S is the singleton, s, known as the seed. [A singleton is a set with only one member, e.g., {s}.] Take an element, u, of S and retrieve the page, p, that it references. Then parse the page p and extract a new set of URLs, L, that are links in that page. Update $S = S + L - u$. Repeat until all links have been extracted and checked. Sounds simple, no?

Now the issue of scalability becomes evident. How would one crawl 1,000,000,000 pages efficiently? After a while the server you're downloading from could become overloaded. If that happens, you could be guilty of a Denial of Service attack or the server could protect itself by blocking any requests from your computer.

Given the volume, how would you pursue all the pages? How deep should you go? Depth first or breadth first? How would you handle broken links, time outs (inability to contact the server in a set amount of time) or "spider traps"?

How might you store and update S and other data structures as needed?

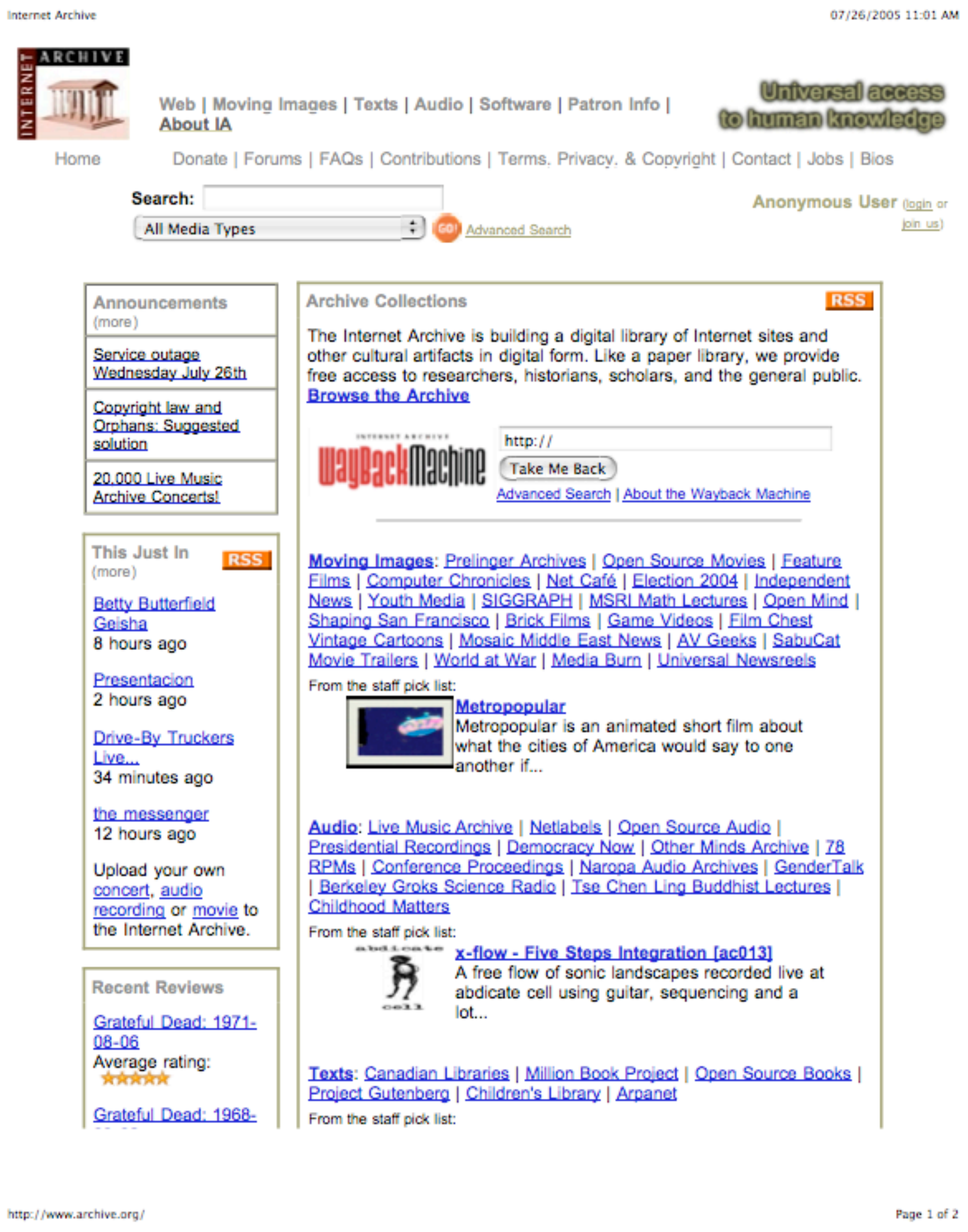
What to retrieve?

Most crawlers retrieve only HTML (leaves and nodes in the tree) and ASCII text. Some retrieve .pdf and .ps (Post Script) files. Once retrieved, what would you index? Some crawlers index only the terms extracted between the <head> and </head> tags; others collect data from the entire .html file. Should the downloaded files be stored, as Google does? The advantage is faster retrieval; the disadvantage is the need for very large storage devices.

To crawl a billion pages/month, the crawler must download about 400 pages per second; this is called a high-performance crawler.

Archives

As more files are "born digital", and more pages' content are changed, a phenomenon akin to a monograph's edition has grabbed the attention of Open Source enthusiasts in the form of the *web archive*. people turn to new ways of creating archives. One archive for the Internet is Brewster Kahle's Internet Archive (http://www.archive.org) to store early versions of pages, accessible via the "WaybackMachine."



Internet Archive 07/26/2005 11:01 AM

Web | Moving Images | Texts | Audio | Software | Patron Info | About IA

Universal access to human knowledge

Home Donate | Forums | FAQs | Contributions | Terms, Privacy, & Copyright | Contact | Jobs | Bios

Search: All Media Types GO! Advanced Search

Anonymous User (login or join us)

Announcements (more)

Service outage Wednesday July 26th

Copyright law and Orphans: Suggested solution

20,000 Live Music Archive Concerts!

This Just In RSS (more)

Betty Butterfield Geisha
8 hours ago

Presentacion
2 hours ago

Drive-By Truckers Live...
34 minutes ago

the messenger
12 hours ago

Upload your own concert, audio recording or movie to the Internet Archive.

Recent Reviews

Grateful Dead: 1971-08-06
Average rating: ★★★★★

Grateful Dead: 1968-

Archive Collections RSS

The Internet Archive is building a digital library of Internet sites and other cultural artifacts in digital form. Like a paper library, we provide free access to researchers, historians, scholars, and the general public.
Browse the Archive

WaybackMachine http:// Take Me Back
Advanced Search | About the Wayback Machine

Moving Images: Prelinger Archives | Open Source Movies | Feature Films | Computer Chronicles | Net Café | Election 2004 | Independent News | Youth Media | SIGGRAPH | MSRI Math Lectures | Open Mind | Shaping San Francisco | Brick Films | Game Videos | Film Chest Vintage Cartoons | Mosaic Middle East News | AV Geeks | SabuCat Movie Trailers | World at War | Media Burn | Universal Newsreels

From the staff pick list:
Metropopular
Metropopular is an animated short film about what the cities of America would say to one another if...

Audio: Live Music Archive | Netlabels | Open Source Audio | Presidential Recordings | Democracy Now | Other Minds Archive | 78 RPMs | Conference Proceedings | Naropa Audio Archives | GenderTalk | Berkeley Groks Science Radio | Tse Chen Ling Buddhist Lectures | Childhood Matters

From the staff pick list:
x-flow - Five Steps Integration [ac013]
A free flow of sonic landscapes recorded live at abdicate cell using guitar, sequencing and a lot...

Texts: Canadian Libraries | Million Book Project | Open Source Books | Project Gutenberg | Children's Library | Arpanet

From the staff pick list:

http://www.archive.org/ Page 1 of 2

Another archive site is http://september11.archive.org/

Keeping crawlers off your site

There are ways, of course, to exclude robots: the Robots Exclusion Protocol. A web site administrator can indicate which parts of the site should not be visited by a robot, by providing a specially formatted file on the site, in http://…/robots.txt. See a live one at http://www.microsoft.com/robots.txt. Here's an example from part of a /robots.txt file

```
# Disallow allow all robots
User-agent: *
Disallow: /cyberworld/map/
Disallow: /tmp/           # these will soon disappear
Disallow: /foo.html

# To allow Cybermapper
User-agent: cybermapper
Disallow:
```

Here is part of *The New York Times*'s robots.txt file:

```
# robots.txt, nytimes.com 4/10/2002
 User-agent: *
 Disallow: /2000
 Disallow: /2001
 Disallow: /2002
 Disallow: /learning
 Disallow: /library
 Disallow: /reuters
 Disallow: /cnet
 Disallow: /archives
 Disallow: /indexes
 Disallow: /weather
 Disallow: /RealMedia
```

The robots meta tag can be included by the web author whether or not the page may be indexed or analyzed for links. No web server administrator action is required. Note, though, that very few robots implement this. In this example the meta tag is in a .html page.

```
<meta name="robots" content="noindex, nofollow">
```

The robot should neither index this document, nor analyze it for links (see http://www.robotstxt.org/wc/exclusion.html#meta)

For the Apache web server, web masters can control access by domain (e.g., exclude all .edu, or include only simmons.edu) in the server's server.xml file.

Example Crawlers – Mercator and Heritrix

Mercator:

One well known crawler is Mercator, which was a continuation of Digital Equipment Corp.'s AltaVista group. Here are some of its features.
A repository with two pluggable methods: add a URL, get a URL.
Most web crawlers use variations of breadth-first traversal, but ...

- Most URLs on a web page are relative (about 80%).
- A single FIFO queue, serving many threads, would send many simultaneous requests to a single server.

Weak politeness guarantee: Only one thread allowed to contact a particular web server.
Stronger politeness guarantee: Maintain *n* FIFO queues, each for a single host, which feed the queues for the crawling threads by rules based on priority and politeness factors.
Duplicate URLs are not added to the URL Frontier
Requires efficient data structure to store all URLs that have been seen and to check a new URL.
In memory:
Represent URL by 8-byte checksum. Maintain in-memory hash table of URLs.
Requires 5 Gigabytes for 1 billion URLs.
Disk based:
Combination of disk file and in-memory cache with batch updating to minimize disk head movement.

DNS:
Resolving domain names to IP addresses is a major bottleneck of web crawlers.
Approach:

- Separate DNS resolver and cache on each crawling computer.
- Create multi-threaded version of DNS code (BIND).

These changes reduced DNS loop-up from 70% to 14% of each thread's elapsed time.

Heritrix

Heritrix is an open source, high performance web crawler project (http://crawler.archive.org/) developed by the Internet Archive and others. Heritrix tries to search as much as possible without timing out or overloading the server. This means Heritrix uses focused crawling of small- to medium-sized crawls (meaning less than 10,000,000 unique documents) in which the quality criterion is complete coverage of selected sites or topics. This application also revisits pages to check for changes. Like most open source projects, Heritrix experiments with techniques, such as number and order of what's crawled, and analyzes the results to improve.

Design parameters

- Extensible. Many components are plugins that can be rewritten for different tasks.
- Distributed. A crawl can be distributed in a symmetric fashion across many machines.
- Scalable. Size of within memory data structures is bounded.
- High performance. Performance is limited by speed of Internet connection (e.g., with 160 Mbit/sec connection, downloads 50 million documents per day).
- Polite. Options of weak or strong politeness.
- Continuous. Will support continuous crawling.

Main Components:

Scope: Determines what URIs are ruled into or out of a certain crawl. Includes the **seed URIs** used to start a crawl, plus the rules to determine which discovered URIs are also to be scheduled for download.

Frontier: Tracks which URIs are scheduled to be collected, and those that have already been collected. It is responsible for selecting the next URI to be tried, and prevents the redundant rescheduling of already-scheduled URIs.

Processor Chains: Modular Processors that perform specific, ordered actions on each URI in turn. These include fetching the URI, analyzing the returned results, and passing discovered URIs back to the Frontier.

- Crawling is carried out by multiple **worker threads**, e.g., 500 threads for a big crawl.
- The **URL frontier** stores the list of absolute URLs to download.
- The **DNS resolver** resolves domain names into IP addresses.
- **Protocol modules** download documents using appropriate protocol (e.g., HTML).
- **Link extractor** extracts URLs from pages and converts to absolute URLs.
- **URL filter** and **duplicate URL eliminator** determine which URLs to add to frontier.

Want to build your own crawler? (or at least evaluate one?)

While it is easy to open a file using a computer program, and equally easy to search for known patterns, extracting links from web pages has proven to be surprisingly difficult. Links in web pages may be absolute, that is the URL is completely expressed, or relative, meaning the part of the link in the page is relative to the current page. The below example shows both relative and absolute links. Programmers may recall that Java includes the method getCodeBase() to determine the front half of the URL (e.g., "http://server.xxx/mainpage.html/")

Example:

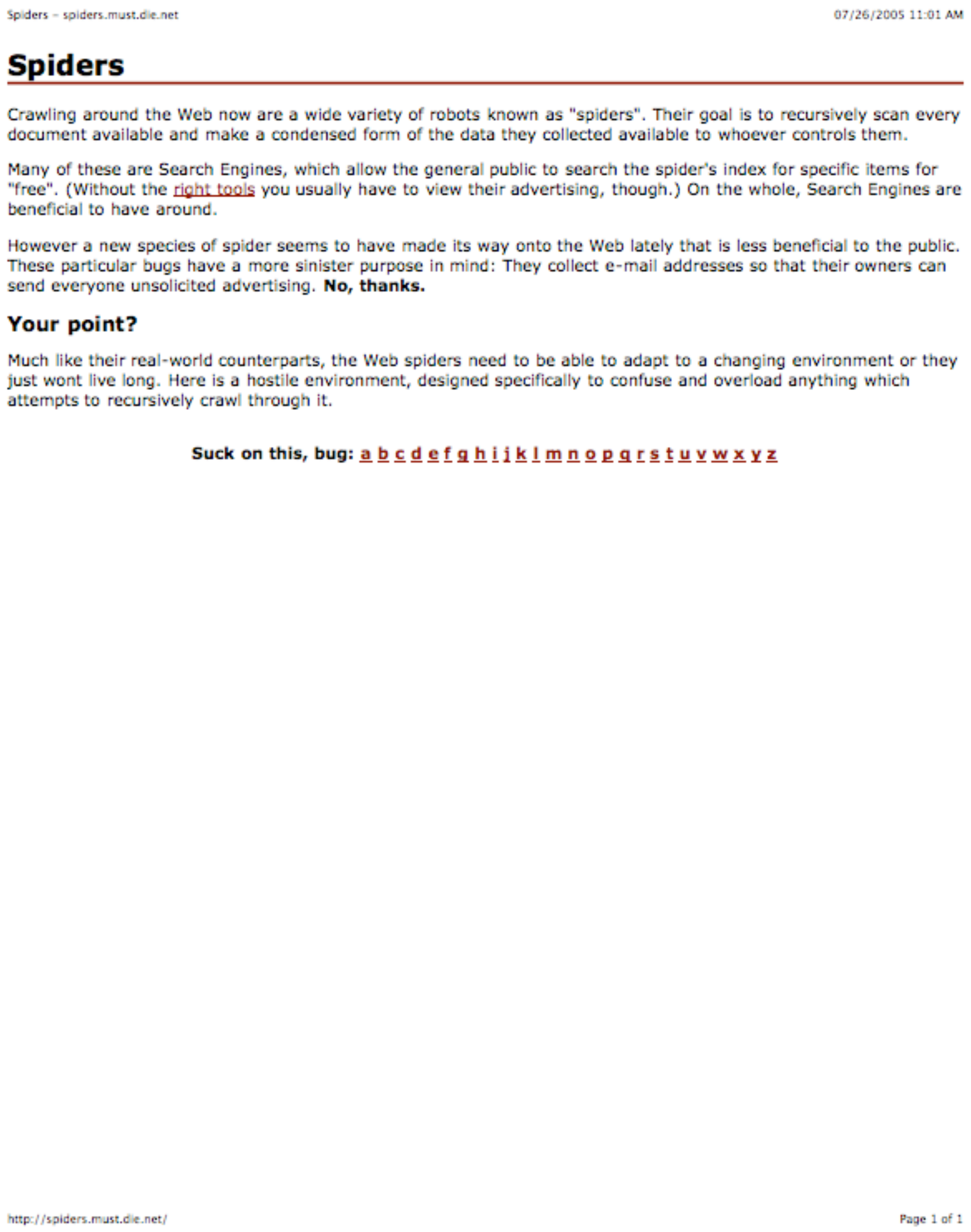
Welcome to the <a href="syllabus.html">relative link syllabus</a> and here is the <a href="http://www.simmons.edu">absolute link</a> to Simmons' home page.

Another difficulty comes from CGI – common gateway interface. Web pages can be generated dynamically, that is, they are stateless, in other words they don't exist on the server until the server's program, such as a Java servlet or PHP script, builds the page, as a response to a request.

Some links are not obvious, such as one in server-side scripts, server-side image maps, and links buried in scripting code.

Arachnaphobia!

No everyone likes spiders! (I don't!) Visit http://spiders.must.die.net/ for an un-friendly perspective on spiders.



Spiders – spiders.must.die.net 07/26/2005 11:01 AM

Spiders

Crawling around the Web now are a wide variety of robots known as "spiders". Their goal is to recursively scan every document available and make a condensed form of the data they collected available to whoever controls them.

Many of these are Search Engines, which allow the general public to search the spider's index for specific items for "free". (Without the right tools you usually have to view their advertising, though.) On the whole, Search Engines are beneficial to have around.

However a new species of spider seems to have made its way onto the Web lately that is less beneficial to the public. These particular bugs have a more sinister purpose in mind: They collect e-mail addresses so that their owners can send everyone unsolicited advertising. **No, thanks.**

Your point?

Much like their real-world counterparts, the Web spiders need to be able to adapt to a changing environment or they just wont live long. Here is a hostile environment, designed specifically to confuse and overload anything which attempts to recursively crawl through it.

Suck on this, bug: a b c d e f g h i j k l m n o p q r s t u v w x y z

http://spiders.must.die.net/ Page 1 of 1

Spider Research:

There is considerable research in improving web crawling. Some revolves around the question of how frequently to crawl and what strategy to use. The goal is to identify anomalies and crawling traps. As an IR issue, web crawling *could* help us to determine automatically the topic of a web page and then do focused/selective crawling to retrieve intellectually-related pages.

Future reading:

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

Heydon, A., & Najork, M. (1999, June 26). Mercator: a scalable, extensible web crawler. Compaq Systems Research Center. Available: http://www.research.Compaq.com/SRC/mercator/papers/www/paper.html

Part 2: The Web as an IR question

The Internet can be conceived of as an information retrieval system. If so, then the usual concepts, such as precision and recall, query/retrieval set, etc., can be applied.

Some IR research shows that information seekers use small queries when using web search engines. The result is that short queries applied to very large number of items (the billions of web pages) leads to a large retrieval set (obviously!) The goal is to try to get the first 10-100 documents that satisfy the user's information need to rank the highest in the retrieval set. This means recall is not important; precision is. It means, too, that ranking that matches the *user's* needs and not the advertisers. As a corollary completeness of index is not an important factor, which means comprehensive crawling is unnecessary.

The concept of relevance is usually binary: the document matches the query or it doesn't. Matching, however, is usually probabilistically done: it is usually estimated by the similarity between terms in the query and each document. In web passed IR, some researchers add the idea of *importance*. Importance measures documents by their likelihood of being useful to a variety of users. Importance is usually estimated by some measure of popularity! Consider this from a librarian's point of view. Before *Harry Potter* was published there was a great deal of media encouragement. This suggests, then, that Harry Potter (as a concept) was *important*.

Web search engines usually rank documents by a *combination of relevance and importance*. The goal is to present the user with the most important of the relevant documents.

Recall that traditional IR uses weighting schemes to improve performance. On the net, however, there are several ranking options:

1. advertisers can pay to have their site rank higher
2. manually created classification
3. vector spacing ranking with corrections for document length
4. extra weighting for specific fields, e.g., title, anchors, etc.
5. popularity, e.g., PageRank.

The balance between numbers 3, 4, and 5 is not made public.

Web-IR Research Trends

There is a standard research technique in LIS to learn about the development of a field of study, using citations, called *bibliometrics*. Bibliometrics uses techniques similar to citation analysis to measure the similarity of journal articles or their importance. ISI, the Institute for Scientific Information, publishes one result of bibliometrics, called *impact factor*. The impact factor is the frequency with which the average article in a journal has been cited in a particular year or period. This knowledge is used to track research trends, interdisciplinary research, and the development of a faculty member's standing in the community. [Something Deans use in their tenure decisions.]

Researchers and librarians are usually interested in two other outcomes: co-citation analysis and bibliographic coupling. Co-citation refers to two papers that *were cited* by many of the same (other) papers. The flip site, bibliographic coupling, refers to papers that *cite* many of the same papers. These data can be represented graphically to help people understand a host of factors, such as social networks, interdisciplinary research, new terminology, new concepts, and so on.

[ADD CITATION GRAPH]

Hyperlinks

Hyperlink analysis is closely related to bibliometrics but uses links from webpages that cite other pages (Thelwell, 2004).

PageRank Algorithm

The same way that co-citation analysis is used to see *who* is important, PageRank algorithm (Brin & Page) is used to estimate the importance of documents. The concept is the rank of a web page is higher if many pages link to it. Links from highly related pages are given greater weight than links from less highly ranked pages.

How? The basic concept is that a user starts with a random web page. Then select a random hyperlink from the current page and jump to the corresponding page. Keep repeating this a number of times. Pages are ranked according to the relative frequency with which they are visited. The result is a matrix:

		Citing Page (from)						
Cited Page								Number
		P1	P2	P3	P4	P5	P6	
	P1					1		1
	P2	1		1				2
	P3	1	1		1			3
	P4	1	1			1	1	4
	P5	1						1
	P6					1		1
Number		4	2	1	1	3	1	

As in traditional IR, the numbers are normalized by the number of links from the page:

		Citing Page (from)						
Cited Page								= normalized link matrix **B**
		P1	P2	P3	P4	P5	P6	
	P1					0.33		
	P2	0.25		1				
	P3	0.25	0.5		1			
	P4	0.25	0.5			0.33	1	
	P5	0.25						
	P6					0.33		
Number		4	2	1	1	3	1	

Weights

Initially, all pages have a weight of 1. The weights are recalculated based on the normalized link matrix. If the user starts at a random page, the *j*th element of w_1 is the probability of reaching page *j* after one step. [This is the beginning of a Markov model.]

$$w_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{After weights are recalculated:} \quad w_1 = Bw_0 = \begin{bmatrix} 0.33 \\ 1.25 \\ 1.75 \\ 2.08 \\ 0.25 \\ 0.33 \end{bmatrix}$$

The idea is to calculate the links iteratively each state *k* from w_k where *k=0*, the start state, until it converges at w (each step is numbered: w_0 to start, then w_1, w_2, w_3, … until the last step, convergence, w. Convergence is the final state.]

Iterate: $w_k = Bw_{k-1}$

$$w_0 = \begin{bmatrix}1\\1\\1\\1\\1\\1\end{bmatrix} w_1 = \begin{bmatrix}0.33\\1.25\\1.75\\2.08\\0.25\\0.33\end{bmatrix} w_2 = \begin{bmatrix}0.08\\1.83\\2.79\\1.12\\0.08\\0.08\end{bmatrix} w_3 = \begin{bmatrix}0.03\\2.80\\2.06\\1.05\\0.02\\0.03\end{bmatrix} \ldots w = \begin{bmatrix}0.00\\2.39\\2.39\\1.19\\0.00\\0.00\end{bmatrix}$$

[Graphic Analysis of Hyperlinks on the Web – GRAPH TO BE ADDED]

Example: Google's PageRank with Damping

A user starts at a random page on the web and the (a) with probability *d* selects any random page and jumps to it; and (b) with probability 1- *d*, selects a random hyperlink from the current page and jumps to the corresponding page. Repeat (a) and (b) a very large number of times. Pages are ranked according to the relative frequency with which they are visited.

The basic method iterates using normalized link matrix, **B**, as described above: $\mathbf{w}_k = \mathbf{B}\mathbf{w}_{k-1}$. This **w** is the higher order eigenvector of **B**. PageRank iterates using a damping factor. The method iterates: $\mathbf{w}_k = d\mathbf{w}_0 + (1-d)\mathbf{B}\mathbf{w}_{k-1}$, where $\mathbf{w}_0$ is a vector with every element equal to 1, *d* is a constant found by experimentation.

This chart shows the iteration with damping of $\mathbf{w}_k = \mathbf{B}\mathbf{w}_{k-1}$ (d=0.3)

$$w_0 = \begin{bmatrix}1\\1\\1\\1\\1\\1\end{bmatrix} w_1 = \begin{bmatrix}0.53\\1.18\\1.53\\1.76\\0.48\\0.53\end{bmatrix} w_2 = \begin{bmatrix}0.41\\1.46\\2.03\\1.29\\0.39\\0.41\end{bmatrix} w_3 = \begin{bmatrix}0.39\\1.80\\1.78\\1.26\\0.37\\0.39\end{bmatrix} \ldots w = \begin{bmatrix}0.38\\1.68\\1.87\\1.31\\0.37\\0.38\end{bmatrix}$$

The Google PageRank estimation (probability, P) is usually written in the below notation. If page *A* has pages T_i point to it, *d* is a damping factor, and *C*(*A*) is the number of links leading out of *A*. Iterate until:

$$P(A) = (1-d) + d\left(\sum_{i=1}^{n} \frac{P(T_i)}{C(T_i)}\right).$$

Information Retrieval using PageRank:

One technique is to consider all hits (that is, all documents vectors that share at least one term with the query vector) as equal. Then display the hits ranked by PageRank. The disadvantage of this method is that it goes no attention to how closely a document matches a query. Naturally, we would like to return documents that *are* similar to the user's query. This is achieved by combining term weighting with reference pattern ranking.

Find all documents that share a term with the query vector. Let s_j be the similarity, using convention term weighting, between the query and document *j*. Thank of document *j* using PageRank of other reference pattern ranking is p_j. Now calculate a *combined rank* c_j=ls_j + (1-l)p_j, where l is a constant. Display the hits ranked by c_j.

This method is actually employed in commercial systems – but the details are not released to the public!

Teoma Dynamic Ranking Algorithm (used in AskJeeves)

As above this search engine starts with conventional term weighting. The hits are ranked using a similarity measure between query and documents. The highest ranking hits are selected for further processing (e.g., the top 5,000 hits). Now the PageRank or a similar algorithm is applied to these documents. This creates a subset of ranked documents, specific to the query. The results are ranked in order of the reference patterns calculated.

The technology developed above is applied also to non-search services. The similarity between documents has been applied by Amazon for its "Other people also bought …" and ePinion's recommender and reputation system.

Google API

No doubt you've noticed how many companies' and universities' [http://www.google.com/options/universities.html] search engines are implementations of Google. Google encourages this by making available its *Application Program Interface* or *API* [http://www.google.com/apis/]. Any API is a rule book, guiding the developer in how to integrate some technology or software into the project. Having invested all this energy into locating, ranking, and storing documents, it makes sense that Google would expand with Google News [news.google.com/] and image searching. Except perhaps for copyright, it'd be relatively easy to write a program to download clustered files from Google and then write an .rss feed!

Part 3 Evaluating Web-based Information Retrieval

Unlike closed, full-text document collections, web-based retrieval is less effective because of the heterogeneity of the materials. Comprehensive metadata with Boolean retrieval works excellently for well-understood categories of material, especially when metadata is included. Creation of metadata-tagged records, however, can be expensive. Full-text indexing for ranked retrieval is very useful, but requires the full text, obviously, and works best when materials are homogeneous. [However, please note that some researchers do not concur.]

Including additional information can be done only with richly-structured documents (e.g., Teoma). Contextual information also improves non-text materials' retrieval. Very few systems, however, make contextual information available to the end-user to help them help themselves understand the rationale for the IR system's retrieval and ranking logic.

Recall from Part 2 that web-based retrieval's goal is to emphasize precision of the most highly-ranked group of hits. Relevance has changed to include "importance" as a factor in ranking. Unlike traditional IR, web-based IR removes the boundary between *searching and browsing*.

Browsing According to Amil Singhal (of Google, 2004), users tend to submit short queries, only 2-4 words long, and they tend to check only the first few results, rarely going beyond the first page! He sees web users as searching to find the site, but then browsing to find the information.

The hit list (retrieval set) displays summary records (aka "snippets") and removes duplicates and groups results from a single site. This is why, for example, searching Google for "Microsoft" retrieves and ranks the Microsoft Corp. homepage and not the millions of pages about Microsoft.

Users also browse the web page itself, looking for direct links to the source pages.

Contextual Information

The context in which an item exists may give useful information for searching. What info? The terms from the document, formatting, and ? form the content. The metadata themselves are data about the document. XML tags, for example, provide great contextual data because of the semantic tagging. In addition, the same extra facets of a document that humans use in determining a document's utility – citations, links, reviews, annotations – can also be searched or used in helping users.

HTML tags, too, are contextually useful. Consider a source page that has the *anchor text* tag <a href="http://www.simmons.edu">Simmons College</a>. That or another page may have that link in the middle of the text {words words words words Simmons College words words words}, which, of course, in the end link to the page. This is the same technique used for searching for images: <img src="images/Bix.jpg" alt="Photo of my beloved cat Bix">

Evaluating Web Searching

Evaluating web searching depends on the purpose of the evaluation. For IR developers, there are some basics considerations. The test corpus must be dynamic because the Web is dynamic. 10%-20% of URLs change monthly and spam prevention methods are always being updated. User queries, too, change all the time: some topics are hot and then not. The text collection, then, should have real queries. As the Internet is truly international, the search engine should be tested for at least 90 different languages and take into consideration both cultural and technical differences around the world. These are *not* simple tasks!