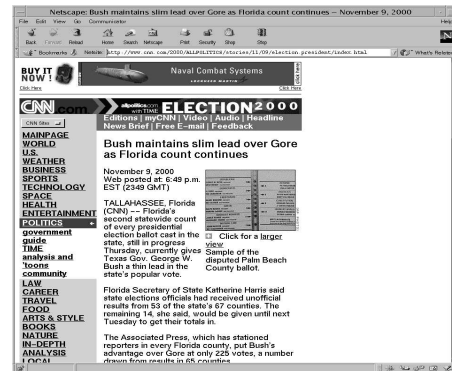


Web Protocols and Web Search Engines

Torsten Suel
CIS Department
Polytechnic University
suel@poly.edu

CS308, Fall 2005
Lecture on 12/06/2005

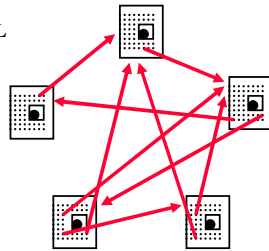
I - Introduction: What is the Web?



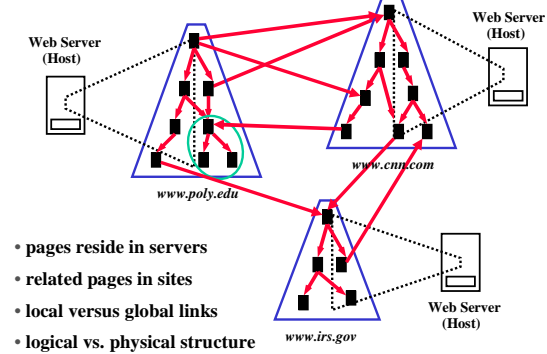
What is the web? (another view)

- pages containing (fairly unstructured) text
- images, audio, etc. embedded in pages
- structure defined using HTML (Hypertext Markup Language)
- hyperlinks between pages!
- over 2 billion pages
- over 10 billion hyperlinks

➔ a giant graph!



How is the web organized?

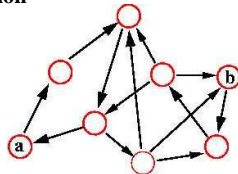


- pages reside in servers
- related pages in sites
- local versus global links
- logical vs. physical structure

How do we find pages on the web?

- more than 4 billion pages
- more than 50 billion hyperlinks
- plus images, movies, .., database content

➔ we need specialized tools for finding pages and information



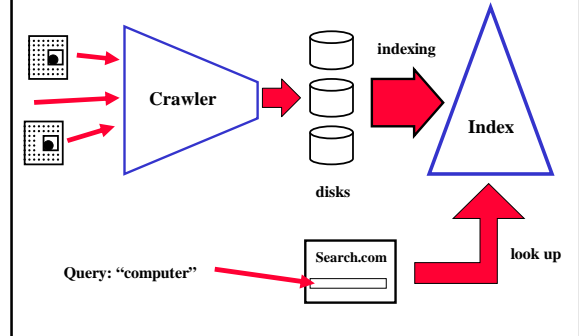
Overview of web search tools

- Major search engines (*google, fast, altavista, inktomi, teoma, wisenut, openfind*)
- Web directories (*yahoo, open directory project*)
- Specialized search engines (*cora, citeseer, aahoo, findlaw*)
- Local search engines (*for one site*)
- Meta search engines (*dogpile, mamma, search.com*)
- Personal search assistants (*alexa, google toolbar*)
- Comparison shopping agents (*mysimon, pricewatch*)
- Image search (*ditto, visoo*)
- Natural language questions (*askjeeves?*)
- Database search (*completeplanet, brightplanet*)

Major search engines:

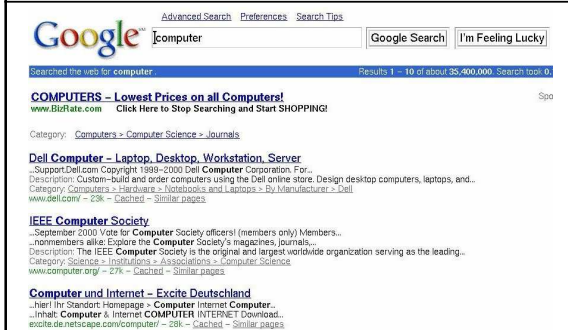


Basic structure of a search engine:



Ranking:

- return best pages first
- term- vs. link-based approaches



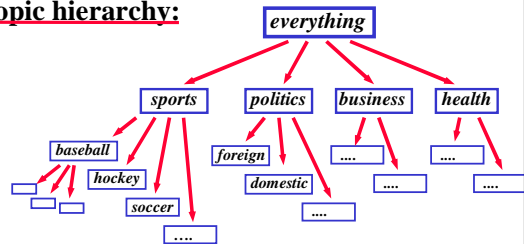
Challenges for search engines:

- coverage (need to cover large part of the web)
 - ➔ need to crawl and store massive data sets
- good ranking (in the case of broad queries)
 - ➔ smart information retrieval techniques
- freshness (need to update content)
 - ➔ frequent recrawling of content
- user load (up to 10000 queries/sec - Google)
 - ➔ many queries on massive data
- manipulation (sites want to be listed first)
 - ➔ naïve techniques will be exploited quickly

Web directories: (Yahoo, Open Directory Project)



Topic hierarchy:



Challenges:

- designing topic hierarchy
- automatic classification: "what is this page about?"
- Yahoo and Open Directory mostly human-based

Specialized search engines: (*achoo, findlaw*)

- be the best on one particular topic
- use domain-specific knowledge
- limited resources → do not crawl the entire web!
- focused crawling techniques (or Meta search)

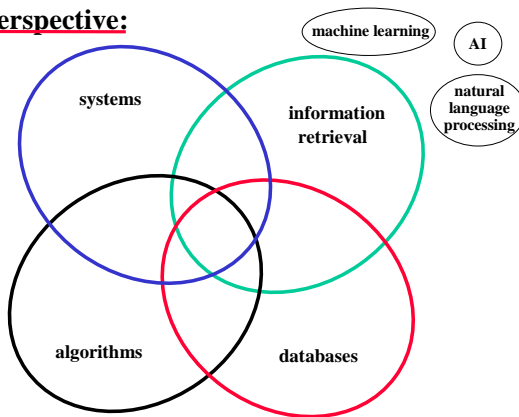
Meta search engines: (*dogpile, search.com, mamma*)

- uses other search engines to answer questions
- ask the right specialized search engine, or
- combine results from several large engines
- may need to be “familiar” with thousands of engines

Personal Search Assistants: (*Alexa, Google Toolbar*)

- embedded into browser
- can suggest “related pages”
- search by “highlighting text” → can use context
- may exploit individual browsing behavior
- may collect and aggregate browsing information
→ privacy issues
- architectures:
 - on top of crawler-based search engine (alexa, google), or
 - based on meta search (*MIT Powerscout*)
 - based on limited crawls by client or proxy (*MIT Letizia, Stanford Powerbrowser*)

Perspective:

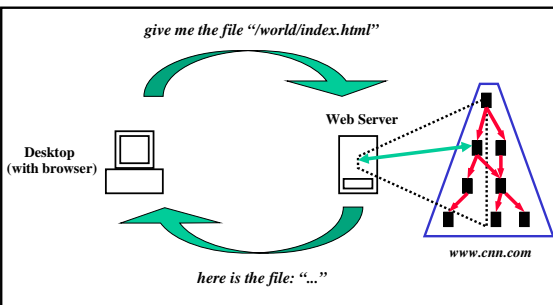


II - Basic Techniques:

- How the web works: (*HTML, HTTP, DNS, web servers, ..*)
- Basic search engine architecture (*google, inktomi*)
- Crawling: (*following links, robot exclusion, black holes, ..*)
- Storage
- Indexing: (*inverted files, index compression, ..*)
- Boolean querying and term-based ranking
- Text classification

3 - How the web works (*more details*)

Fetching “*www.cnn.com/world/index.html*”



Three Main Ingredients:

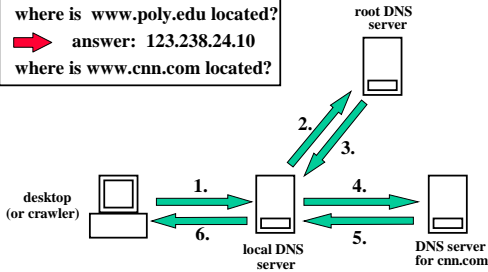
- Naming: URL (uniform resource locators) (used to identify and locate objects)
- Communication: HTTP (hypertext transfer protocol) (used to request and transfer objects)
- Rendering: HTML (hypertext markup language) (used to defined how object should be presented to user)

Client Server Paradigm:

- Client (browser) used HTTP to ask server (web server) for object identified by URI, and renders this object according to rules defined by HTML

Domain Name Service:

where is www.poly.edu located?
 → answer: 123.238.24.10
 where is www.cnn.com located?



Names, addresses, hosts, and sites

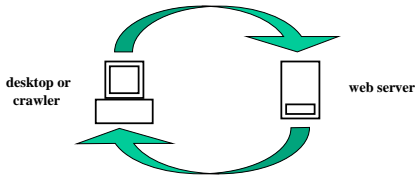
- one machine can have several host names and IP addresses
- one host name may correspond to several machines
- one host can have several “sites” (what is a site?)
- one “site” on several hosts
- issues: detecting duplicates, crawling, local vs. global links

```
wease1% nslookup www.cnn.com
Server: photon.poly.edu
Address: 128.238.32.22

Non-authoritative answer:
Name:      cnn.com
Addresses: 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28
           207.25.71.29, 207.25.71.30, 207.25.71.5, 207.25.71.6, 207.25.71.20
           207.25.71.22, 207.25.71.23, 207.25.71.24
Aliases:   www.cnn.com
```

HTTP:

```
GET /world/index.html HTTP/1.0
User-Agent: Mozilla/3.0 (Windows 95/NT)
Host: www.cnn.com
From: ...
Referer: ...
If-Modified-Since: ...
```



```
HTTP/1.0 200 OK
Server: Netscape-Communications/1.1
Date: Tuesday, 8-Feb-99 01:22:04 GMT
Last-modified: Thursday, 3-Feb-99 10:44:11 GMT
Content-length: 5462
Content-type: text/html

<the html file>
```

HTML:

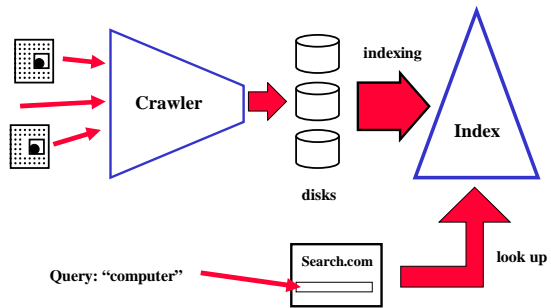


```
<HTML>
<HEAD>
<TITLE> Some interesting links </TITLE>
</HEAD>
<body BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="#000099">
<H1>
Some interesting links
</H1>
<p>
<strong>Search Engines:</strong><p>
<a href="http://www.google.com">Google Search Engine</a><br>
<a href="http://www.altavista.com">AltaVista Search</a>
</BODY>
</HTML>
```

HTTP & HTML issues:

- “dynamic” URLs:
 - <http://www.google.com/search?q=brooklyn>
 - <http://www.amazon.com/exec/obidos/ASIN/1558605703/qid%3D9...>
 - <http://cis.poly.edu/search/search.cgi>
- result file can be computed by server in arbitrary manner!
- persistent connections in HTTP/1.1
- mime types and extensions
- frames
- redirects
- javascript/java/JEB/flash/activeX ????????

Search Engine Architecture:



Crawler

- fetches pages from the web
- starts at set of “seed pages”
- parses fetched pages for hyperlinks
- then follows those links (e.g., BFS)
- variations:
 - recrawling
 - focused crawling
 - random walks

Indexing

aardvark	3452, 11437,
...	...
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
...	...
zebra	602, 1189, 3209, ...

“inverted index”

- parse & build lexicon & build index
- index very large

➔ I/O-efficient techniques needed

Querying

Boolean queries:
(zebra AND armadillo) OR armani

➔ unions/intersections of lists

aardvark	3452, 11437,
...	...
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
...	...
zebra	602, 1189, 3209, ...

look up

Google:

[Source: *Brin/Page, WWW Conf., 1998*]

Inktomi:

- network of workstations/servers (*Sun or Linux, Myrinet SAN*)
- BASE vs. ACID (*Basically Available, Soft-state, Eventual consistency*)
- data and index partitioned over machines
- each node responsible for part of the web (*horizontal partitioning*)

Crawling the Web:

- Basic idea:
 - start at a set of known URLs
 - explore the web in “concentric circles” around these URLs

- start pages
- distance-one pages
- distance-two pages

Simple Breadth-First Search Crawler:

```
insert set of initial URLs into a queue Q
while Q is not empty
  currentURL = dequeue(Q)
  download page from currentURL
  for any hyperlink found in the page
    if hyperlink is to a new page
      enqueue hyperlink URL into Q
```

*this will eventually download all pages reachable from the start set
(also, need to remember pages that have already been downloaded)*

Traversal strategies: (why BFS?)

- crawl will quickly spread all over the web
- load-balancing between servers
- in reality, more refined strategies (*but still BFSish*)
- many other strategies (*focused crawls, recrawls, site crawls*)

Tools/languages for implementation:

- Scripting languages (*Python, Perl*)
- Java (*performance tuning tricky*)
- C/C++ with sockets (*low-level*)
- available crawling tools (*usually not completely scalable*)

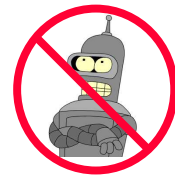
Details: (lots of 'em) (*see this paper for details*)

- handling filetypes
(*exclude some extensions, and use mime types*)
- URL extensions and CGI scripts
(*to strip or not to strip? Ignore?*)
- frames, imagemaps, base tags
- black holes (robot traps)
(*limit maximum depth of a site*)
- different names for same site?
(*could check IP address, but no perfect solution*)

Performance considerations: later!

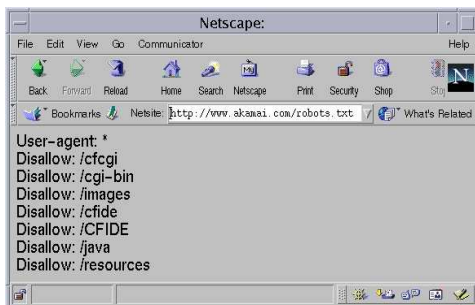
Robot Exclusion Protocol

(*see Web Robots Pages*)



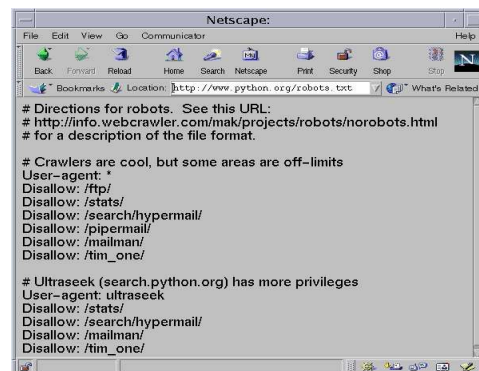
- file robots.txt in root directory
- allows webmaster to “exclude” crawlers (*crawlers do not have to obey*)
- may exclude only certain robots or certain parts of the site
 - to “protect proprietary data” (e.g., eBay case)
 - to prevent crawlers from getting lost
 - to avoid load due to crawling
 - to avoid crashes (protect CGI bin)
- if at all possible, follow robot exclusion protocol!

Robot exclusion - example:



```
User-agent: *
Disallow: /cfcgi
Disallow: /cgi-bin
Disallow: /images
Disallow: /cfide
Disallow: /CFIDE
Disallow: /java
Disallow: /resources
```

Robot exclusion - example:



```
# Directions for robots. See this URL:
# http://info.webcrawler.com/mak/projects/robots/norobots.html
# for a description of the file format.

# Crawlers are cool, but some areas are off-limits
User-agent: *
Disallow: /ftp/
Disallow: /stats/
Disallow: /search/hypermail/
Disallow: /pipermail/
Disallow: /mailman/
Disallow: /tim_one/

# Ultraseek (search.python.org) has more privileges
User-agent: ultraseek
Disallow: /stats/
Disallow: /search/hypermail/
Disallow: /mailman/
Disallow: /tim_one/
```

Robot META Tags

(see [Web Robots Pages](#))

- allow page owners to restrict access to pages
- does not require access to root directory
- excludes all robots
- not yet supported by all crawlers
- “noindex” and “nofollow”

Crawling courtesy

- minimize load on crawled server
- no more than one outstanding request per site
- better: wait 30 seconds between accesses to site
(this number is not fixed)
- problems:
 - one server may have many sites (use domain-based load-balancing)
 - one site may have many pages (3 years to crawl 3-million page site)
 - intervals between requests should depend on site
- give contact info for large crawls (email or URL)
- expect to be contacted ...

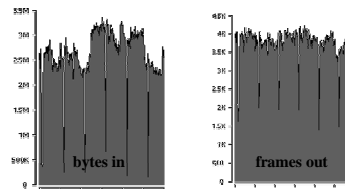
Crawling challenges

- crawler may have to run for several weeks or months
- will interact with millions of web server
- some of them will be odd:
 - noncompliant server responses
 - unfamiliarity with robot exclusion protocol
 - robot traps
 - CGI and unintended consequences
 - network security tools
 - weird webmasters
- unclear legal situation

High Performance Crawling:

Example: PolyBot (V. Shkapenyuk)

- crawler for massive data acquisition
hundreds of pages per second, billions of pages total
- scalable implementation on a cluster
- optimized networking and data structures



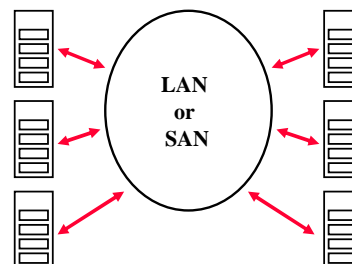
Poly T3 connection over 24 hours of 5/28/01 (courtesy of AppliedTheory)

Storage:

- average HTML page size: ~ 14KB (plus ~ 40KB images)
- 2 billion pages = 28 TB of HTML
- compression with gzip/zlib: 7-8 TB (3-4 KB per page)
- or about 3 KB text per page after stripping tags
(according to Stanford WebBase group)
- 1 KB per page if stripping and compressing
- 1-4 KB compressed index size per page
(depends on whether we store position in document)
- 2-8 TB index size for 2 billion pages
- page and index compression important

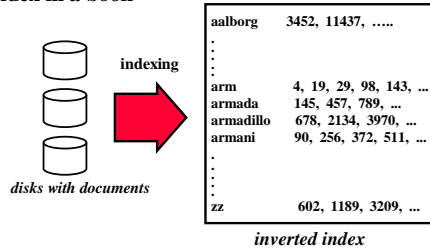
Low cost storage:

- Linux PCs connected by Ethernet or [Mvrinet](#) SAN (system area network)
- several disks per node (160GB IDE for \$230)
- [Stanford WebBase](#), [Internet Archive](#) (and here at Poly)
- parallel processing, active/intelligent disks paradigm
- separate data and index, or not?

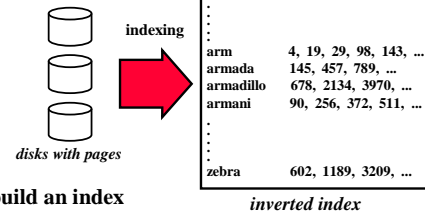


Text Index:

- a data structure that for supporting IR queries
- most popular form: *inverted index structure*
- like index in a book



Indexing



- how to build an index
 - in I/O-efficient manner
 - in-place (no extra space)
 - in parallel (later)
- closely related to I/O-efficient sorting
- how to compress an index (*while building it in-place*)
- goal: intermediate size not much larger than final size

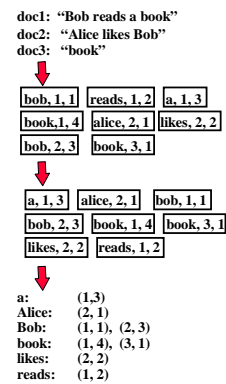
Basic concepts and choices:

- **lexicon:** set of all "words" encountered
millions in the case of the web, mostly non-words
- for each word occurrence:
store index of document where it occurs
- also store position in document? (*probably yes*)
 - increases space for index significantly!
 - allows efficient search for phrases
 - relative positions of words may be important for ranking
- also store additional context? (*in title, bold, in anchor text*)
- stop words: common words such as "is", "a", "the"
- ignore stop words? (*maybe better not*)
 - saves space in index
 - cannot search for "to be or not to be"
- stemming: "runs = run = running" (*depends on language*)

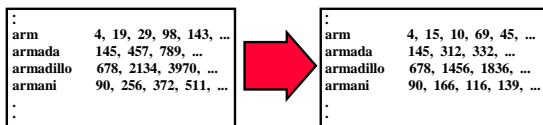
Indexing: (simplified approach)

(see Witten/Moffat/Bell for details)

- (1) scan through all documents
- (2) for every word encountered generate entry (word, doc#, pos)
- (3) sort entries by (word, doc#, pos)
- (4) now transform into final form



Improvements



- encode sorted runs by their gaps
➔ significant compression for frequent words!
- less effective if we store position
(adds incompressible lower order bits)
- many highly optimized schemes have been studied
(see book)

Additional issues:

- keep data compressed during index construction
- try to keep index in main memory? (*altaVista*)
- keep important parts in memory? (*fancy hits in google*)
- use database to store lists? (*e.g., Berkeley DB*)
use BLOBs for compressed lists; rely on DB for caching
- or use text indexes provided by databases?

Alternative to inverted index:

- signature files (Bloom filters): false positives
- bitmaps
- better to stick with inverted files!

Boolean querying and term-based ranking:

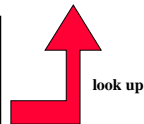
(see *Managing Gigabytes* book)

Recall Boolean queries:

(zebra AND armadillo) OR armani

→ unions/intersections of lists

aardvark	3452, 11437,
...	...
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
...	...
zebra	602, 1189, 3209, ...



Boolean queries vs. ranking

- most web queries involve one or two common words
 - Boolean querying returns thousands of hits
- would like to rank results by ...
 - importance?
 - relevance?
 - accuracy?
- in general, arbitrary score function:
 - “return pages with highest score relative to query”
- use inverted index as access path for pages
 - start with (possibly expanded) Boolean query
 - only rank Boolean results
 - in fact, try to avoid computing complete Boolean results

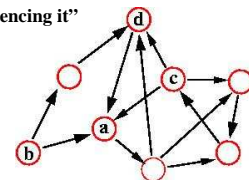
Ranking continued:

- vast amount of vector space work in IR
 - (see *Witten/Moffat/Bell* and *Baeza-Yates/Ribeiro-Neto* for intro & pointers)
- not all results directly applicable to search engines
- additional factors in ranking:
 - distance between terms in text
 - titles and headings and font size
 - use of meta tags?
 - user feedback or browsing behavior?
 - link structure!
- efficiency extremely important! (Google: 10000 queries/sec)

(3) Fundamentals of IR Systems

Link-based ranking techniques

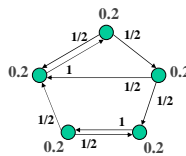
- Ragerank (Brin&Page/Google)
 - “significance of a page depends on significance of those referencing it”
- HITS (Kleinberg/IBM)
 - “Hubs and Authorities”



$$s(a) \sim s(b) + s(c) + s(d) ?$$

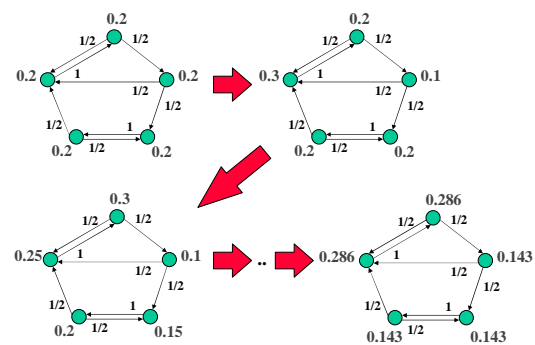
(4) Mathematical Techniques

Pagerank



- initialize the rank value of each node to $1/n$ (0.2 for 5 nodes)
- a node with k outgoing links transmits a $1/k$ fraction of its current rank value over that edge to its neighbor
- iterate this process many times until it converges
- NOTE: this is a random walk on the link graph
- Pagerank: stationary distribution of this random walk

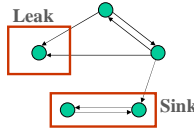
(4) Mathematical Techniques



(4) Mathematical Techniques

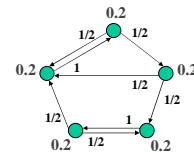
Convergence

- dealing with leaks:
 - pruning, or
 - adding back links
- dealing with sinks
 - add a random jump to escape sink
 - with prob. $b = 0.15$ jump to random node
- assume: if in a leak, always take random jump
- in this case, always converges to unique solution!



(4) Mathematical Techniques

Matrix notation



A

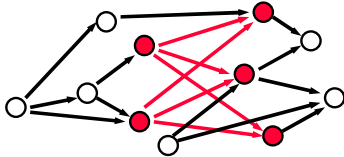
0	1/2	0	0	1/2
0	0	1/2	0	1/2
0	0	0	1	0
0	0	1/2	0	1/2
1	0	0	0	0

- stationary distribution: vector x with $xA = x$
- A is primitive, and x Eigenvector of A
- computed using Jacobi or Gauss Seidel iteration

(4) Mathematical Techniques

Other iterative techniques for link analysis

- “HITS” (Jon Kleinberg)



- query-dependent: first get 100 docs using term-based techniques
- build a subgraph on these nodes and their neighbors
- run iterative process on this subgraph
- each node has *hub score* and *authority score*

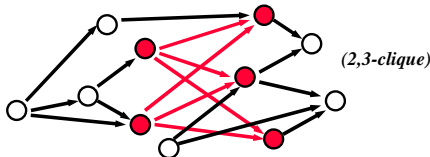
(4) Mathematical Techniques

Other iterative techniques for link analysis

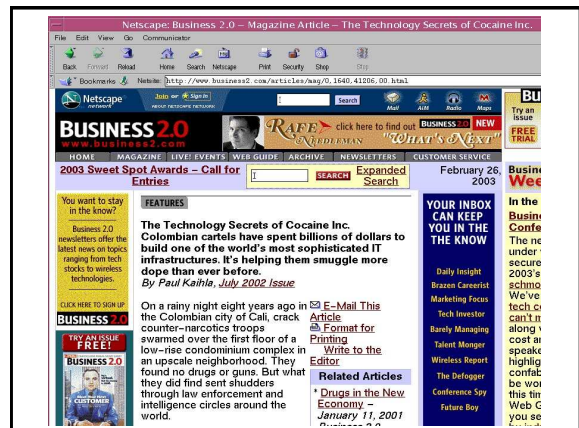
- many other techniques based on techniques such as markov chains, SVD, principal component analysis
- web graph result of billions of decisions by millions of independent web page creators (or not!)
- data mining of web graphs and web usage patterns
- techniques not really new
 - analysis of social network (Katz 1953)
 - citation analysis (Garfield)
- security-related applications: call pattern analysis

Example: Analysis of the web graph

- What does the web look like? (*diameter, connectivity, in-degree*)
- Why are there so many bipartite cliques? (IBM) (*and why do we care?*)



- How do you compute with a 500 million node graph?



Useful Software Tools: (and how about databases ...)

- major search engines based on proprietary software
 - must scale to very large data and large clusters
 - must be very efficient
 - low cost hardware, no expensive Oracle licenses
- site search based on standard software
 - appliance; set up via browser (e.g., google)
 - or software with limited APIs (e.g., altaVista, fast (gone), ...)
 - or part of web server or application server
 - or offered as remote services (fast (gone), atomz, ...)
- enterprise search: different ballgame
 - security/confidentiality issues
 - data can be extremely large
 - established vendors (e.g., Verity)
- other cases: what to do?

IR/Search & Databases

- database text extensions (*Oracle, IBM, Informix, Texis*)
 - e.g., Oracle9i text extensions (*formerly interMedia text*)
 - e.g., IBM DB2: text extender, text information extender, net search extender
- offer inverted indexes, querying, crawling
- allow mixing of IR and DB operators
- optimizing mixed queries may be a problem (DB2 a bit better integrated IMO)
- support for IR operations such as categorization, clustering
- support for languages (stemming) and various file types
- integrated with database, ACID properties

- simple search almost out of the box
- efficient enough for most cases, but not cost-effective for largest systems (scalability, cost)

IR/Search & Databases

- when to use DBMS?
 - transaction properties needed?
 - complex queries that mix text and relational data?
 - which features are needed? which ones are really provided?
 - efficiency (DBMS overhead, index updates?)
 - how far do you need to scale? (Oracle, IBM: scaling == \$\$\$)
- DBMS / IR gap:
 - getting smaller (DBMSs are getting there)
 - but be aware of differences: not everything is a relation, and a standard DB index is not a good text index
 - also fundamental tension between goals of DB and IR
 - DB: precise semantics, not good with black-box IR

Useful software tools (ctd.)

- lucene (*part of Apache Jakarta*)
 - free search software in Java: crawling, indexing, querying
 - inverted index with efficient updates
 - documents are stored outside
 - good free foundation
 - currently being integrated into Nutch open search engine
 - also, mg and zettair systems
- IBM intelligent miner for text (*not sure still on market*)
 - similar features as lucene, plus extra IR operations
 - categorization, clustering, languages, feature extraction
 - uses DB2 to store documents, but not fully integrated (*different from DB2 text extensions*)
- MS indexserver/siteserver
 - provides indexing and crawling on NT
- many other tools ... (*see here for list*)

Conclusions: software tools

- evolving market: vendors from several directions moving in
- massive-data engines: proprietary code, made from scratch
- site search: out of the box
- many other applications in between should try to utilize existing tools, but cannot expect complete solutions
- most freely or widely available tools are not completely scalable or stable

Questions:

- how much do you need to scale, and how much can you pay?
- transaction properties needed?
- mixture of text and relational data?
- support for different languages and data types needed?
- advanced IR and data mining, or simple queries?

Scenario 1: Major Search Engine

- 8 billion pages, up to 10000+ queries per second (google)
- very large, scalable clusters of rackmounted servers
- Google: linux with proprietary extensions
- mostly Linux on Intel (earlier: Inktomi Solaris/Sun, AltaVista DEC)
- large-scale parallel processing
- sort of simplified database? (GFS, bigTable, mapReduce)
- parallel crawler for data acquisition: 1000+ pages per second
- pages and index are partitioned over cluster in redundant way
- all major engines: horizontal partitioning
 - each node contains subset of pages, and an inverted index for this subset only

Course Offering at Poly

- **CS912: Web Search Engines**
- **Course Objectives**
 - web and search engine architecture *how does it all work?*
 - working with massive data sets *storing and analyzing terabytes*
 - introduction to Information Retrieval *unstructured data, text*
 - system building skills *building distributed systems*
 - the Web as a social network *adversarial behavior, spam, communities*
- **Next offering probably in Fall 2006**