Crawling the Web

Web Crawling

- Retrieve (for indexing, storage, ...) Web pages by using the links found on a page to locate more pages.
- Must have some starting point

Type of crawl

- **Web crawl** versus
crawl of more limited network — web
  - cs.princeton.edu
  - internal co. network
- **complete crawl** versus
  focused crawl by some criteria
  - pages on one topic
- Type of crawl will affect necessity/usability of various techniques

Main Issues I

- starting set of pages?
  - a.k.a “seed” URLs
- can visit whole of Web (or web)?
- how determine order to visit links?
  - graph model:
    - breadth first vs depth first
  - “black holes”
  - other aspects /considerations
    - how deep want to go?
    - associate priority with links

“Black holes” and other “baddies”

- “Black hole”: Infinite chain of pages
  - dynamically generated
  - not always malicious
    - link to “next month”, which uses perpetual calendar generator
- Other bad pages
  - other behavior damaging to crawler?
    - servers
    - spam content
    - use URLs from?

Robust crawlers must deal with black holes and other damaging behavior
Main Issues II

• Web is dynamic
  – time to crawl “once”
  – how mix crawl and re-crawl
    • priority of pages
• Social behavior
  – crawl only pages allowed by owner
    • robot exclusion protocol: robots.txt
  – not flood servers
    • expect many pages to visit on one server

Technical issues

• maintain one or more queues of URLs to be visited: URL frontier
  – order of URLs in queues?
    • FIFO = breadth first
    • LIFO = depth first
    • priority queues
• resolve hostname in URLs to get actual IP addresses – Domain Name Service servers (DNS lookup)
  – bottleneck:
    • servers distributed
    • can have high lookup latency

Technical issues continues

• To do large crawls must have multiple crawlers with multiple network connections (sockets) open and probably multiple queues

• large crawls generate large amount data
  – need fast access => main memory
  – cache: hold items most likely to use in main memory instead of
    • on disk
    • request from server

DNS lookup

• cache DNS map
  – large, local, in memory
  – hold most recently used mappings
• don’t want temporal locality of reference
  – be nice to servers (or else)
• prefetch DNS resolution for URLs on page when it parsed?
  – batch requests
  – put in cache
  – use when URL gets to head of queue
  – resolution stale?
• How “large” cache?
  – Problems?

(Near?) Duplicate pages

Has page been indexed already?

• mirror sites – different URLs, same page
  – bad: duplicate page in search results
  – worse?: add links from duplicate pages to queues
    • also mirrors?
  – mirrored pages may have slight differences
    • e.g. indicate which mirror they on
• other sources duplicates & near duplicates
  – eg …/spr12/cos435/ps1.html
    …/spr11/cos435/ps1.html
(Near?) Duplicate page removal

- table of fingerprints or sketches of pages
  - fit in main memory?
  - if not, costs disk access per page crawler retrieves
- cache?
  - less likely to hit sketch in cache than, say, URL?

When apply duplicate removal?

- while crawling versus for search results
  - crawling larger problem
  - search results demand faster results
- duplicates versus near duplicates
  - same policy?

Duplicate URL removal

IS URL in URL frontier?
Has URL already been visited? if not recrawling
⇒ Has URL ever been in URL frontier?

- Use:
  - canonical, fully specified URLs
  - canonical hostname provided by DNS
- Visited? hash table
  - hash canonical URL to entry
- Visited? table may be too large for MM

Caching Visited? table

- not temporal but “popularity” locality:
  - most popular URLs
  - most popular sites
    - some temporal locality within
- to exploit site-level locality need hash that brings pages on same site together:
  - two-level hash:
    - hash hostname and port
    - hash path
  - can use B+ tree, sorted on i then ii
    - if no entry for URL in tree, not visited

Re-crawling

- When re-crawl what pages?
  - finish crawl and start over
    - finish = have enough?
  - re-crawl high priority pages in middle of crawl
    - how determine priority?
- How integrate re-crawl of high priority pages?
  - One choice – separate cycle for crawl of high priority pages

Another choice: Mercator scheme

from slides for Intro to IR, Sec. 20.2.3
Mercator prioritizing

- Assigning priority
  - properties of page from previous visits
  - e.g. how often page change
  - class of pages
  - news, blogs, … high priority for recrawl
- Front queue for each priority: FIFO
- “Biased front queue selector” implements policy by choosing which queue next

Mercator politeness enforcement: Back queues

- at any point each queue contains only URLs from one host
- additional information
  - table mapping host to queue
  - heap containing entry for each queue/host: earliest time can next request from host
- heap min gives next queue to use for URL to fetch
  - wait until earliest allowed time to fetch

Maintaining back queues

- When a back queue emptied, remove URLs from front queues - putting in appropriate back queues until remove URL from new host
- put URL from new host in empty back queue
  - update host- back queue table
  - determine “earliest request time”
  - insert in heap

Crawling large number pages

- indexing is not* dynamic and continuous
  - Google in fall 2010 announced now has dynamic index
  - Index all pages collected at certain time (end of crawl?)
  - Provide search half of engine with new index
- crawling is continuous
  - some choices:
    - reinsert seed URLs in queue when fetch
    - also reinsert high-priority URLs when fetch
    - reinsert all URLs with varying priority when fetch

Focused Web Crawling

How change crawling strategy if only want pages that
- on a particular topic
- match particular query
- satisfy a particular predicate

- example: crawling for 3D models

Question
**Issues**

- Are issues:
  - Depth v.s. Breadth
    - desired pages may be "deep" in Web
  - 100% coverage of relevant pages

- Are not issues:
  - recrawl (?)
  - 100% coverage of web

---

**How Prune Search?**

One method (Chakrabarti et. al.):
- have desired topic + classifier
- each time acquire page, use classifier to ask if it on topic
- harvest links of page only if on topic

---

**Alternative:**

*Intelligent Crawling on the World Wide Web with Arbitrary Predicates*

- Do not assume, build statistical evidence:
  - parent interesting => page interesting
  - siblings interesting => page interesting
- crawler learns importance of different features of pages as indicators of relevance of other pages yet to visit
- learns how prioritize pages for visiting
- Start as random crawler and adjust as learn

---

**Calculating priority of pages in queue for visiting**

- Features considered
  - content of parent web pages
  - % of parents satisfying predicate
  - % of siblings satisfying predicate
  - "tokens" in URL of page
    - e.g. "edu", "princeton"
- Use a numerical *interest ratio* to prioritize

---

**Summary**

- focused crawling for specialized applications
- have been many proposed methods
- need
  - more analysis per page
  - less throughput