Crawlers - Introduction

- The role of crawlers is to collect Web content
- Starting with some seed URLs, crawlers learn of additional crawl targets via hyperlinks on the crawled pages
- Several types of crawlers:
  - Batch crawlers – crawl a snapshot of their crawl space, until reaching a certain size or time limit
  - Incremental crawlers – continuously crawl their crawl space, revisiting URLs to ensure freshness
  - Focused crawlers – attempt to crawl pages pertaining to some topic/theme, while minimizing number of off-topic pages that are collected
Crawlers - Introduction

- Web scale crawling is carried out by distributed crawlers, complicating what is conceptually a simple operation
- Resources consumed: bandwidth, computation time (beyond communication – e.g. parsing), storage space

Generic Web Crawling Algorithm

Given a root set of distinct URLs:

- Put the root URLs in a (priority) queue
- While queue is not empty:
  - Take the first URL x out of the queue
  - Retrieve the contents of x from the web
  - Do whatever you want with it…
  - Mark x as visited
  - If x is an html page, parse it to extract hyperlinked URLs
  - For each hyperlink URL y within x: if y hasn't been visited (perhaps lately), enqueue y with some priority

Note that this algorithm may never stop on evolving graphs
Batch Crawlers – Stopping Criterion

Batch crawling (perhaps focused) continues until one of the following conditions is met:

- Some elapsed time passes
- A certain number of pages are crawled
- Some storage limit is reached
- Variations:
  - A certain number of sites are crawled
  - A certain number of “hot” pages are crawled
Crawler Design Issues

- Writing a simple, single-threaded crawler is a fairly easy task – one can be written in Java in a few lines of code
- However, many issues come up when writing a scalable and robust crawler, suitable for web search, e.g.:
  - Redirections – chains and loops
  - Encrypted and password-protected pages/sites
  - Politeness - avoiding server load, respecting Robot Exclusion Protocol
  - Script handling, frames, language and encoding issues
  - The challenge of dynamic pages (“deep web”)
  - Crawl prioritization and variable revisit rates to ensure freshness
  - Concurrency and distribution issues, synchronization
  - Avoiding “bottomless pits” (rapidly branching sites, including spam)
  - And many more…

HTTP Redirection - Example

GET /index.html HTTP/1.0

HTTP/1.0 302 Moved Temporarily
Date: Mon, 01 Nov 1999 11:52:06 GMT
Server: NCSA/1.5
Location: http://mapuccino.haifa.ibm.com/mapuccino/index.html
Content-type: text/html

<HEAD><TITLE>Document moved</TITLE></HEAD>
<BODY><H1>Document moved</H1>
This document has moved
</BODY>

[slide courtesy of Michael Herscovici (Google) and Michal Jacobi (IBM)]
Handling Redirections

- Redirection: the response to an HTTP GET request to download URL A might redirect the request to URL B
  - HTTP response code 301/302
- Chains of redirections: A→B→C→…
  - What URL to choose as corresponding to the indexed content, i.e. what URL to report in search results?
    - Most reasonable: the final URL in the redirect chain, so that users will see the URL they will end up on if clicked
    - Need to associate the anchor text and the inlinks of all pages in the redirect chain with the chosen canonical URL prior to indexing and computing any link-based measures

Politeness - Avoiding Server Load

- A multithreaded crawler can overwhelm a web site by sending too many concurrent GET requests (DoS – “Denial of Service”)
- Search engines’ crawlers “throttle” themselves by limiting the rate of requests sent to any single site
  - Implies some synchronization mechanism between the threads to access/update the timestamps of requests to sites
  - In a distributed crawler, simplest solution is to reduce the problem to the single-node multithreaded case by having each site crawled by a single crawl node
    - Imperfect as some servers have multiple names and IPs
    - Might skew the work balance among crawl nodes
  - Need to avoid “Liveloocks” – when most sites in the queue are throttled, the effective throughput of the crawler may become low
Politeness - Robots Exclusion

- By using the Robots Exclusion Protocol, Web site administrators can define parts of their sites to be off-limits to crawlers.
- The Robots Exclusion Protocol cannot prevent a crawler from crawling off-limits pages – avoiding such pages is an act of good will on the part of the crawler.
  - Part of “Web Etiquette”
  - All major search engines respect robot exclusion policies
- Site administrators can define exclusion policies in two ways:
  - A robots.txt file at the root of the server (or directory)
  - Robots HTML meta tags at each page, which are intended at preventing the search engine from indexing that page.

[slide courtesy of Michael Herscovici (Google) and Michal Jacobi (IBM)]

Politeness & Crawl Frontier

- Performing this check efficiently is one of the major performance bottlenecks
- Requires batching of carefully-sized merge-sort operations
Sitemaps (http://www.sitemaps.org/)

- While the robots.txt file defines which parts of a site should not be crawled, the Sitemaps protocol allows Webmasters to inform crawlers which URLs should be crawled.
- A Sitemap is an XML file that contains a list of URLs that are available for crawling, along with some metadata (last update time, update rate, relative importance in the eyes of the Webmaster).
- Allows for more comprehensive and better prioritized crawling of the site by search engines:
  - Can list dynamic pages to which no static links exist.
- The Sitemaps protocol is supported by all major search engines.
  - However, the engines do not guarantee crawling each URL in the sitemap, nor do they guarantee any other conforming behavior.

Crawl Prioritization

- Crawlers prioritize their downloads in an attempt to fetch “high-quality” pages first.
- Pages are prioritized in light of importance metrics:
  - Interest driven metrics (focused crawlers)
    - Fit to some predefined category or similarity to a driving query.
  - Popularity driven metrics
    - Inlinks count, PageRank
    - Can be estimated as the crawl progresses, or taken from previous crawl cycles.
  - URL driven metrics
    - E.g. URLs containing “home”, or according to depth of URL.
Crawl Prioritization - Example

- Corpus: 179K pages from the standord.edu domain
- “Hot Pages” – pages with more than X inlinks in the crawl space
- Measured rate of crawling the hot pages using 4 different URL prioritization policies
- Surprisingly, although hotness was defined according to inlink counts, PageRank ordering collected hot pages fastest
- However, ordering crawls by PageRank is expensive

Fig. 2. The performance of various ordering metrics for IB(P); G = 100.

Image taken from "Searching the Web", Arasu et al., ACM TOIT 1(1), 2001

Crawl Prioritization - Continued

- Corpus: 328M pages, crawled over 58 days
- PageRank normalized to sum up to 328M instead of 1 – average PageRank is 1
- BFS crawling, which is computationally cheap, is shown to grab high-PageRank pages rather quickly
- Results even more extreme when computing PageRank only by inter-site links

Image taken from: "Breadth First Search Crawling Yields High Quality Pages", Najork and Wiener, WWW’2001

- Assume we want to prioritize a continuous crawl by a PageRank-like link-based importance measure of Web pages
- One option is to build a link database as we crawl, and use it to compute PageRank every K crawl operations
  - This is both difficult and expensive
- The paper above details a method to compute a PageRank-like measure in an on-line, asynchronous manner with little overhead in terms of computations, I/O and memory
- The description in the following slides is for a simplified version of a continuous crawl over N pages whose links never change
  - The set of pages to be continuously crawled is fixed, and only their text is subject to change

“Page Importance” Definition

- Let $G=(V,E)$ be the (directed) link graph of the Web
- We build $G’=(V’,E’)$ by adding a virtual node $y$ to $G$, with links to and from all other nodes; thus, $G’$ will be strongly connected
  - Formally, $V’ = V \cup \{y\}$, $E’ = E \cup \{ \forall x \in V: x \to y, y \to x \}$
  - Whenever $|E|>0$, $G’$ is also aperiodic
- Let $M(G’)$ denote the row-normalized adjacency matrix of $G’$; obviously $M(G’)$ is stochastic, and by the above assumption also Ergodic
- Let $n$ denote the principal eigenvector of $M(G’)$ (and stationary distribution of the Markov chain it represents), i.e. $n = n M(G’)$
- We will define $n$ as the Importance Vector of all nodes in $G’$ that the algorithm will compute
On-Line Algorithm for Computing Page Importance – Notations

The algorithm assigns each page $v$ the following variables:
1. Cash money $C(v)$
2. Bank money $B(v)$

In addition, let $L(v)$ denote the outlinks of page $v$, and let $G$ denote the global amount of money in the bank.

On-Line Algorithm for Computing Page Importance

The algorithm itself proceeds as follows:
- Initialization: for all pages, let $C(v)\leftarrow 1/N$, $B(v)\leftarrow 0$; $G\leftarrow 0$
- Repeat forever:
  - Pick a page $v$ to visit according to the Visit Strategy*
  - Distribute (evenly) an amount equal to $C(v)$ among $v$'s children, i.e. $\forall j \in L(v)$, $C(j) += C(v) / |L(v)|$
  - Deposit $v$'s cash in the bank:
    $B(v) += C(v)$, $G += C(v)$, $C(v)\leftarrow 0$
- Fairness condition*: every page is visited infinitely often
Analysis of the Algorithm

**Lemma 1:** at all times, $\Sigma_v C(v) = 1$ (proof by easy induction)

**Lemma 2:** at all times, $B(v) + C(v) = 1/N + \Sigma_{j:j\rightarrow v}M(G'), B(j)$
Proof: also by induction, which is trivial at time zero. The step analyzes three distinct cases of which page is visited:
1. $v$ is visited, and then nothing changes on the RHS while money is just moved from $C(v)$ to $B(v)$ on the LHS
2. A page $j$ that links to $v$ is visited. The LHS grows by $C(j)/|L(j)|$, which is the exact increment of the RHS
3. Otherwise – neither side changes

Analysis of the Algorithm (cont.)

**Lemma 3:** $G$ goes to infinity as the algorithm proceeds
Proof: at any time $t$ there is at least one page $x$ whose cash amount is at least $1/N$. Since each page is visited infinitely often, there is a finite $t'>t$ in which $x$ is visited, thus $G$ will increase by at least $1/N$ by time $t'$
**Analysis of the Algorithm – Main Theorem**

Lemma 1: at all times, $\sum_v C(v) = 1$ (proof by easy induction)

Lemma 2: at all times, $B(v) + C(v) = 1/N + \sum_{j:j \rightarrow v} M(G')_{j,v} B(j)$

Lemma 3: $G$ goes to infinity as the algorithm proceeds

Let $B$ be the normalized bank vector, i.e. $B_v = B(v)/G$. Thus, $B$ is a distribution vector.

Theorem: $|B^* M(G') - B| \rightarrow 0$ as the algorithm proceeds

Proof: examine the $v$'th coordinate:

$|B^* M(G') - B|_v = G^{-1} * |B(v) - \sum_j B(j) M(G')_{j,v}|$

$= G^{-1} * |B(v) + C(v) - C(v) - \sum_{j:j \rightarrow v} B(j) M(G')_{j,v}|$

$= G^{-1} * |1/N - C(v)| < G^{-1} \rightarrow 0$

Conclusion: $B \rightarrow \pi$, the stationary distribution of $M(G')$

---

**Possible Visit Strategies – Crawling Policies**

1. Round Robin (obviously fair)

2. Random – choose the next node to visit u.a.r., thus guaranteeing that the probability of each page being visited infinitely often is $1$

3. Greedy - visit the node with maximal cash, thus increasing $G$ in the fastest possible manner
   - Why are all nodes are visited infinitely often?
Additional Notes

1. There are adaptations of the algorithm to the case of evolving graphs and to a distributed implementation.

2. Estimating v’s importance by \( \frac{(B(v)+C(v))}{(G+1)} \) is slightly better than just using \( \frac{B(v)}{G} \).

3. If one can keep the cash values for all pages in RAM, then each step in the algorithm involves a single I/O operation to update \( B(v) \) (beyond normal crawling I/O).

Focused Crawlers

- Attempt to collect pages pertaining to a specific topic or theme
  - Recall: collecting as many pages on the topic as possible
  - Precision: avoiding the download of off-topic pages
- Main observation: on-topic pages may link to other on-topic pages, whereas off-topic pages most likely will not
- The challenge is how to prune the crawl
Pruning a Focused Crawl

- Identify whether a crawled page is on/off topic
  - Topical Classifiers are put to work
- Identify which outlinks of an on-topic page are worth fetching
  - Using the vicinity around the link and of course its anchor-text
- The evidence on the potential of a target URL may come from multiple source pages
  - The crawl priority of a page should be allowed to change - complicates the crawl queue mechanism

Web Page Update Statistics

“A Large Scale Study of the Evolution of Web Pages”, Fetterly et al., WWW 2003:

- 151M pages were crawled every week for 11 consecutive weeks in late 2002 through early 2003
  - A four-node Mercator crawler using breadth-first search
- About half the pages were successfully downloaded in all 10 recrawl attempts. Another third was successfully downloaded in 9 out of 10 recrawls
  - In each individual crawl cycle over 85% of pages were successfully downloaded, and 88% were successfully downloaded in the last (11th) crawl cycle
Web Page Update Statistics

- Week-to-week comparisons of pages indicate that:
  - 65.2% of Web pages do not change at all within a week
  - Another 9.2% of pages change very slightly, only in some HTML element
  - Some rapidly changing pages are machine-generated spam pages
  - Past change rates of a page are indicative of future change rates, i.e. rate of change is a property of a document

Crawl Refresh Rate

- Downloaded pages should be refreshed periodically
- Possible refresh strategies:
  - Uniform: allocate crawl slots equally to each page
  - Proportional: allocate crawl slots in proportion to each page’s rate of change
- Freshness metrics:
  - Ratio of fresh pages: the proportion of pages whose crawled copy equals their up-to-date copy
  - Average stale age of pages: average time since the actual page has changed with respect to crawled page
  - Embarrassment probability: the probability of a stale page to (1) be returned for the query \( q \), (2) clicked by the user, and (3) found to be inconsistent with respect to \( q \)
Optimal Crawl Scheduling

“Optimal Crawling Strategies for Web Search Engines”, Wolf et al., WWW’2002:

- Assume there are \( N \) pages to be crawled in \( T \) time units, and that \( R > N \) fetches are possible during \( T \)
- Assume that page \( j \)’s inter-update times are independent and identically distributed with known mean
- There is an algorithm that determines an ideal crawling schedule that minimizes staleness/embarrassment by computing:
  - The number of times \( x_j \) to crawl page \( j \), such that
    \[
    \sum_{j=1}^{N} x_j = R
    \]
  - The times \( 0 < t_j(1) < \ldots < t_j(x_j) < T \) in which to crawl page \( j \)
- Complexity is \( O(N + N \log(R/N)) \)

Optimal Crawl Scheduling (cont.)

- Once the ideal crawl schedule has been computed, the \( R \) crawl tasks need to be assigned to \( C \) crawlers, with crawler \( k \) being able to carry out \( s_k \) of the \( R \) tasks
  - Assume that each crawl performed by crawler \( k \) takes \( T/s_k \) units
  - If the \( i \)'th crawl of page \( j \) is scheduled as the \( d \)'th task performed by crawler \( k \), the cost is defined as \( |t_j(i) - dT/s_k| \)
    - The difference between the ideal time of the task and its actual time
    - Can be expressed as a linear optimization problem (transportation)
- There is a polynomial-time algorithm that computes the min-cost achievable schedule
  - Minimizes a flow in a bipartite graph where the \( R \) source nodes are the ideal crawl times, the \( R \) sink nodes are the realizable crawl times, and each arc carries a penalty as described above
Optimal Crawl Scheduling (cont.)

Experiments and simulations were done for various inter-update rates and distributions (Poisson, Pareto and quasi-deterministic)

- Objective function: minimizing the average staleness of pages
- Non-intuitive(?) result: the uniform strategy is better than the proportional strategy
  - Means that on average, it is better not to try to keep up with a few frequently changing pages since that will degrade the freshness of the rest of the pages
- The refresh strategy that is computed based on the ideal crawl strategy outperformed the uniform strategy