Chapter 6

Advanced Crawling Techniques

Outline

• Selective Crawling
• Focused Crawling
• Distributed Crawling
• Web Dynamics

Web Crawler

• Program that autonomously navigates the web and downloads documents
• For a simple crawler
  – start with a seed URL, \( S_0 \)
  – download all reachable pages from \( S_0 \)
  – repeat the process for each new page
  – until a sufficient number of pages are retrieved
• Ideal crawler
  – recognize relevant pages
  – limit fetching to most relevant pages
Nature of Crawl

- Broadly categorized into
  - Exhaustive crawl
    - broad coverage
    - used by general purpose search engines
  - Selective crawl
    - fetch pages according to some criteria, for e.g., popular pages, similar pages
    - exploit semantic content, rich contextual aspects

Selective Crawling

- Retrieve web pages according to some criteria
- Page relevance is determined by a scoring function $s_{θ}(ξ)(u)$:
  - $ξ$ relevance criterion
  - $θ$ parameters
  - for e.g., a boolean relevance function
    - $s(ξ) = 1$ document is relevant
    - $s(ξ) = 0$ document is irrelevant

Selective Crawler

- Basic approach
  - sort the fetched URLs according to a relevance score
  - use best-first search to obtain pages with a high score first
  - search leads to most relevant pages
Examples of Scoring Function

- **Depth**
  - length of the path from the site homepage to the document
  - limit total number of levels retrieved from a site
  - maximize coverage breadth
  \[
  \delta_u = \begin{cases} 
  \text{if } \text{depth}(u) < \delta_x, & 1 \\
  0, & \text{otherwise}
  \end{cases}
  \]

- **Popularity**
  - assign relevance according to which pages are more important than others
  - estimate the number of backlinks
  \[
  \text{indegree}(u) = \begin{cases} 
  \text{if } \text{indegree}(u) < \delta_x, & 1 \\
  0, & \text{otherwise}
  \end{cases}
  \]

Examples of Scoring Function

- **PageRank**
  - assign value of importance
  - value is proportional to the popularity of the source document
  - estimated by a measure of indegree of a page

Efficiency of Selective Crawler

- BFS crawler
- Crawler using backlinks
- Crawler using PageRank
**Focused Crawling**

- Fetch pages within a certain topic
- Relevance function
  - use text categorization techniques
  - $s^{\text{topic}}(u) = P(c|d(u), \theta)$
- Parent based method
  - score of parent is extended to children URL
- Anchor based method
  - anchor text is used for scoring pages

**Focused Crawler**

- Basic approach
  - classify crawled pages into categories
  - use a topic taxonomy, provide example URLs, and mark categories of interest
  - use a Bayesian classifier to find $P(c|p)$
  - compute relevance score for each page
  - $R(p) = \sum_{c \in \text{good}} P(c|p)$

**Focused Crawler**

- Soft Focusing
  - compute score for a fetched document, $S_0$
  - extend the score to all URL in $S_0$
  - $s_{\text{finc}}(u) = P(c|d(v), \theta)$
  - if same URL is fetched from multiple parents, update $s(u)$
- Hard Focusing
  - for a crawled page $d$, find leaf node with highest probability ($c^*$)
  - if some ancestor of $c^*$ is marked good, extract URLs from $d$
  - else the crawl is pruned at $d$
Efficiency of a Focused Crawler

Context Focused Crawlers

- Classifiers are trained
  - to estimate the link distance between a crawled page and the relevant pages
  - use context graph of $L$ layers for each seed page

Context Graphs

- Seed page forms layer 0
- Layer $i$ contains all the parents of the nodes in layer $i-1$
Context Graphs

• To compute the relevance function
  – set of Naïve Bayes classifiers are built for each layer
  – compute \( P(t|c_1) \) from the pages in each layer
  – compute \( P(c_1|p) \)
  – class with highest probability is assigned the page
  – if \( P(c_1|p) < \tau \), then page is assigned to ‘other’ class

• Maintain a queue for each layer
• Sort queue by probability scores \( P(c_1|p) \)
• For the next URL in the crawler
  – pick top page from the queue with smallest \( l \)
  – results in pages that are closer to the relevant page first
  – explore outlink of such pages

Reinforcement Learning

• Learning what action yields maximum rewards
• To maximize rewards
  – learning agent uses previously tried actions that produced effective rewards
  – explore better action selections in future
• Properties
  – trial and error method
  – delayed rewards
Elements of Reinforcement Learning

- **Policy** $\pi(s,a)$
  - probability of taking an action $a$ in state $s$
- **Rewards function** $r(a)$
  - maps state-action pairs to a single number
  - indicate immediate desirability of the state
- **Value Function** $V^\pi(s)$
  - indicate long-term desirability of states
  - takes into account the states that are likely to follow, and the rewards available in those states

Reinforcement Learning

- **Optimal policy** $\pi^*$ maximizes value function over all states:
  
  $V^*(s) = \max_{\pi \in \Pi} V^\pi(s) = \max_{\pi \in \Pi} \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$
  
- **LASER** uses reinforcement learning for indexing of web pages
  - for a user query, determine relevance using TFIDF
  - propagate rewards into the web
  - discounting them at each step, by value iteration
  - after convergence, documents at distance $k$ from $u$ provides a contribution $\gamma^k$ times their relevance to the relevance of $u$

Fish Search

- **Web agents** are like the fishes in sea
  - gain energy when a relevant document found
  - search for more relevant documents
  - lose energy when exploring irrelevant pages
- **Limitations**
  - assigns discrete relevance scores
    - 1 – relevant, 0 or 0.5 for irrelevant
  - low discrimination of the priority of pages
Shark Search Algorithm

• Introduces real-valued relevance scores based on
  – ancestral relevance score
  – anchor text
  – textual context of the link

Distributed Crawling

• A single crawling process
  – insufficient for large-scale engines
  – data fetched through single physical link
• Distributed crawling
  – scalable system
  – divide and conquer
  – decrease hardware requirements
  – increase overall download speed and reliability

Parallelization

• Physical links reflect geographical neighborhoods
• Edges of the Web graph associated with “communities” across geographical borders
• Hence, significant overlap among collections of fetched documents
• Performance of parallelization
  – communication overhead
  – overlap
  – coverage
  – quality
Performance of Parallelization

- Communication overhead
  - fraction of bandwidth spent to coordinate the activity of the separate processes, with respect to the bandwidth usefully spent to document fetching
- Overlap
  - fraction of duplicate documents
- Coverage
  - fraction of documents reachable from the seeds that are actually downloaded
- Quality
  - e.g. some of the scoring functions depend on link structure, which can be partially lost

Crawler Interaction

- Recent study by Cho and Garcia-Molina (2002)
- Defined framework to characterize interaction among a set of crawlers
- Several dimensions
  - coordination
  - confinement
  - partitioning

Coordination

- The way different processes agree about the subset of pages to crawl
- Independent processes
  - degree of overlap controlled only by seeds
  - significant overlap expected
  - picking good seed sets is a challenge
- Coordinate a pool of crawlers
  - partition the Web into subgraphs
  - static coordination
    - partition decided before crawling, not changed thereafter
  - dynamic coordination
    - partition modified during crawling (reassignment policy must be controlled by an external supervisor)
Confinement

- Specifies how strictly each (statically coordinated) crawler should operate within its own partition
  - Firewall mode
    - each process remains strictly within its partition
    - zero overlap, poor coverage
  - Crossover mode
    - a process follows interpartition links when its queue does not contain any more URLs in its own partition
    - good coverage, potentially high overlap
  - Exchange mode
    - a process never follows interpartition links
    - can periodically dispatch the foreign URLs to appropriate processes
    - no overlap, perfect coverage, communication overhead

Crawler Coordination

Let $A_{ij}$ be the set of documents belonging to partition $i$ that can be reached from the seeds $S_j$

Partitioning

- A strategy to split URLs into non-overlapping subsets to be assigned to each process
  - compute a hash function of the IP address in the URL
  - e.g. $n \in \{0, \ldots, 2^{32}-1\}$ corresponds to IP address
  - $m$ is the number of processes
  - documents with $n \mod m = i$ assigned to process $i$
  - take to account geographical dislocation of networks
Web Dynamics

- Rate of change of information on the Web
- Used by search engines for updating index
- Notion of recency proposed by Brewington and Cybenko (2000)
- The index entry for a document indexed at time \( t_0 \) is \( \beta \)-current at time \( t \) if the document has not changed during \([t_0, t-\beta]\)
- \( \beta \) is a grace period
- A search engine for a given collection is \((\alpha, \beta)\)-current if the probability that a document is \( \beta \)-current is at least \( \alpha \)

Lifetime of Documents

- \( T \) – lifetime of a component
  - the time when a component breaks down
  - a continuous random variable
- The cumulative distribution function (cfd) of lifetime \( F(t) = P(T \leq t) \)
- Reliability (survivorship function) - the probability that the component will be functioning at time \( t \)
  \[ S(t) = 1 - F(t) = P(T > t) \]

Aging of Components

- Age of a component – time elapsed since the last replacement
- Cdf of age – the expected fraction of components that are still operating at \( t \)
  \[ G(t) = P(\text{age} < t) = \frac{\int_0^t S(\tau)\,d\tau}{\int_0^\infty S(\tau)\,d\tau} \]
- The age probability density function (pdf) \( g(t) \) is proportional to the survivorship function
Aging of Documents

- \( S(t) \) is the probability that a document last changed at time zero will remain unmodified at time \( t \)
- \( G(t) \) is the expected fraction of documents that are older than \( t \)
- Probability that a document will be modified before an additional time \( h \) has passed is the conditional probability \( P(t < T \leq t + h \mid T > t) \)
- The change rate
  \[
  \lambda(t) = \lim_{h \to 0} \frac{P(t < T \leq t + h \mid T > t)}{h}
  \]
  - where \( f(t) \) is the lifetime pdf

Age and Lifetime on the Web

- Assuming constant change rate, estimate lifetime from observed age
  \[
  F(t) = 1 - e^{-\lambda t}, \quad g(t) = f(t) = \lambda e^{-\lambda t}
  \]
  - use Last-Modified timestamp in the HTTP header
- Brewington and Cybenko (2000)
  - 7 million Web pages
  - observed between 1999 and 2000
Empirical Distributions

Growth with Time
• Essential property that the Web is growing with time – not captured by the model
• Most documents are young
• Suppose exponential distribution for growth
  – if documents were never edited, their age is the time since their creation
  – trivial growth model yields an exponential age distribution
• Realistic model should take to account both Web growth and document refreshing
  – use hybrid model

Meaningful Timestamps
• Often servers do not return meaningful timestamps
  – especially for dynamic Web pages
• Thus, estimate of change rates using lifetimes rather than ages
• Brewington and Cybenko (2000)
  – an improved model that explicitly takes to account the probability of observing a change, given the change rate and the timespan
Improved Model

- Document changes are controlled by an underlying Poisson process
  - probability of observing a change is independent of previous changes
- Mean lifetimes are Weibull distributed
- Change rates and timespans are independent
- The resulting lifetime distribution

\[ f(t) = \int_0^\infty e^{-\lambda t} \hat{w}(1/\lambda) d(1/\lambda) \]

where \( \hat{w}(1/\lambda) \) is an estimate of the mean lifetime.

Estimated Mean Lifetime Distribution

Refresh Interval

- Estimate how often a crawler should refresh the index of a search engine to guarantee that it remains \((\alpha, \beta)\)-current
- Consider a single document
- Let \( t=0 \) be the time the document was last fetched
- Let \( I \) be the interval between two consecutive visits
Refresh Interval

- The probability that for a particular time $t$ the document is unmodified in $[0, t-\beta]$ is
  \[e^{-(t-\beta)}, \quad t \in [\beta, \lambda/\beta]\]
  \[1, \quad t \in (0, \beta)\]
- The probability that a collection of documents is $\beta$-current is
  \[\alpha = \lambda \int_{0}^{\beta} \left( \frac{\beta}{t} + \frac{1 - e^{-(t-\beta)}}{t} \right) dt\]

Example

- Brewington and Cybenko (2000)
  - reindexing period of about 18 days
  - assuming a Web size of 800 million pages
  - to guarantee that 95% of repository was current up to one week ago

Freshness

- Another measure of recency
- Freshness $\phi(t)$ at time $t$ of a document
  - binary function indicating whether the document is up-to-date in the index at time $t$
- Expected freshness – probability that the document did not change in $(0, t)$
  \[E(\phi(t)) = e^{-\lambda t}\]
- Corresponds to $\beta$-currency for $\beta = 0$
- If document $d$ is refreshed regularly each $I$ time units, average freshness is
  \[\phi = \frac{1 - e^{-\lambda I}}{2I}\]
Index Age

- Index age \( a(t) \) of a document is the age of the document if the local copy is outdated, or zero if the local copy is fresh.
- Expected index age at time \( t \):
  \[
  E[a(t)] = \int_{0}^{t} (t-s)e^{-s} \, ds = t - \frac{1-e^{-t}}{\lambda}
  \]
- Index age average in \((0, I)\):
  \[
  \bar{a} = \frac{1-e^{-1}}{\lambda I} - \frac{1}{2} \frac{1}{I} \]

Recency and Synchronization Policies

- Not all sites change their pages at the same rate.
- Cho and Garcia-Molina (1999)
  - monitored about 720,000 popular Web pages
  - a vast fraction of this Web is dynamic
  - dramatically different results for top-level domains

Study Results

- Average change interval found in a study.
- 270 popular sites were monitored for changes from 17 February to 24 June 1999. A sample of 3000 pages was collected from each site by visiting in breadth first order from the homepage.
Optimal Synchronization Policy

- Resource allocation policy should consider specific site dynamics
- Simplifying assumptions:
  - $N$ documents of interest
  - Estimated change rate $\lambda_i$, $i = 1,...,N$
  - A crawler regularly fetches each document $i$ with a refresh interval $I_i$
  - Fetch each document in constant time
  - $B$ – the available bandwidth – the number of documents that can be fetched in a time unit

Optimal Resource Allocation

- Limited bandwidth constraint $\sum_{i=1}^{N} \frac{1}{I_i} \leq N$
- The problem of optimal resource allocation
  - select the refresh intervals $I_i$ to maximize a recency measure of the resulting index
- E.g. maximize freshness

$$(I'_1,\ldots,I'_N) = \arg \max_{(I_1,\ldots,I_N)} \sum_{i=1}^{N} \phi(\lambda_i, I_i) \text{ subject to } \sum_{i=1}^{N} \frac{1}{I'_i} \leq N$$

Example

- WebFountain (Edwards et al. 2001)
- A fully distributed and incremental crawler
  - no central control
  - repository entry of a document is updated as soon as it is fetched
  - crawling process is never regarded complete
  - changes are detected when a document is re-fetched
  - documents are grouped by similar rates of change
  - trade-off between re-fetching (freshness) and exploring (coverage) controlled by optimizing the number of old and new URLs