Dynamic Search
A practical perspective

Presented by:
Dan Pelleg

Overview

- What is Dynamic Search?
- Motivation
- Fish-Search
- Fish-Search Formalized
- Shark-Search (aka Mapuccino)
- Metrics
- Spider Ethics
- Hacking

What is Dynamic Search?

- User types in a query, resource limits (usually time)
- Dynamic search tool interacts directly with WWW
  - Can also consult static search engines
- When time is up, user gets "quality" results:
  - No "404 – doc not found"
  - Relevant documents, hopefully better than static search
  - More information (eg: Mapuccino — visualization of subgraph)
- CLEVER is one example

Dynamic Search – motivation

- Web is a temporal DB
  - If a server is down or a page has moved, why list it?
  - Average lifetime of a WWW document is 24 days
    - Halflife time anyone?
- Cognitive-Science approach: given results from a static search engine, most users will...
- Another way to look at a "traditional" spider in a static search-engine: maximize information, minimize time

Fish-Search

- “The search-engine is like a school of fish foraging for information in the WWW ocean”
- “Where food is abundant, they reproduce”
- “They die in polluted water” (this is low bandwidth)
- “They split to follow new paths”
- de Bra et al. ’94, not necessarily for WWW

Fish-Search formalized

- Keep a (priority, depth) queue of yet-to-be-explored documents
- Initialize it to some seed URL and depth = D
  - Getting this seed URL is an orthogonal problem
- Repeat while resources last:
  - Pop maximal priority document off queue
  - Do nothing if depth is zero
  - Read and parse document (this is the costly part)
  - If it is relevant, a reasonable number of its children enter the queue with priority 1 and depth D
  - Otherwise, some number of its children enter the queue with priority 0.5 and depth 1 less the parent's
- Post-processing here

- Fish-search is really “Best-First search”.
- Original fish-search uses boolean “relevance” measure. We refine it to include any similarity measure $\mathrm{sim}(d, q)$

  (d document, q query)
- Our similarity measure was cosine-angle

Refinements(1)

- Score decay:
  - Each node inherits similarity score from ancestors
  - Children of relevant nodes get similarity score of parent
  - Children of irrelevant nodes get inherited score of parent
    - Then we multiply by decay factor d (cf. Marchirori '97)
  - With inherited priorities, depth usually redundant

Refinements(2)

- Considering anchor text similarity
  - Instead of choosing children arbitrarily
  - "Neighborhood" of anchor text also included
- Priority now is a linear combination of inherited score and anchor score

The refined algorithm is called... "Shark-Search" (Maarek et al '98, `www.ibm.com/java/mapuccino`)

Shark-Search: complaints

- Cosine measure inaccurate:
  - Twice as good to have a word appear twice instead of once? 20 times instead of 10?
  - The ”bag of words” approach is problematic
- Age-old “search-engine persuasion” problem
- How to choose limits?
- Usually don’t want many results from single site
- Intra-site links less significant
- So are links to mega sites/advertisement/counter sites
- “Site” is a vague thing

We hope that combination of content and hyper-info will do the trick.

Metrics

Traditional IR approach: D docs (=WWW), $R \subseteq D$ relevant, S retrieved by our tool.

$$\begin{array}{rcl} \textit{recall:} & \frac{|S \cap R|}{|D \cap R|} & \text{“all the truth”} \\ \textit{precision:} & \frac{|S \cap R|}{|S|} & \text{“nothing but the truth”} \end{array}$$

Recall demands a test-site. The community still awaits...
(CMU-Web-KB effort?)

Precision sounds irrelevant, since user browses anyway (top 10 items are the only ones ever read)

Our Metrics

We used: $\sum_{d \in S} \mathrm{sim}(d, q)$

- "The more the merrier", bad docs don't harm us
- Potential scores (queue priorities) not counted — just actual score of fetched documents
- Hand-crafted queries to compare. Not very scientific
- Blind-test
- Hand-crafted queries used for CLEVER
- Other ways to measure?

Spider Ethics

- Identify yourself
  - Make sure you're known to the world
- Do not overload other party
  - No multiple connections
  - Work at off-load times
  - Use a caching proxy
  - Respect `robots.txt`

    also ROBOTS meta tag?

Algorithms

- How to do dot-product
- How to compute Hub/Authority score

Try and implement the following:

- Parse HTML tags: headings, lists, paragraph break...
- Multimedia indexing:
  - "the strip where Dilbert said..."
  - "find free RealAudio station that broadcasted at least 3 Beethovens this week"
  - Read PS,pdf,Word... files
- Getting rid of advertisements

  junkbuster proxies?