Web Search

Jeremy Nicholson and Justin Zobel and Karin Verspoor, CIS

Semest

THE UNIVERSITY OF MELBOURNE

Elements of a web search engine

Web search involves four main technological components.

- **Crawling**: the data to be searched needs to be gathered from the web
- **Parsing**: the data then needs to be translated into a canonical form.

Practical search also involves an increasi technologies, such as:

- Snippet generation.
- As-you-type querying.
- Query correction.
- Answer consolidation. (cf. Product price lists)
- Info boxes. (cf. Google Knowledge Graph)

Crawling fundamentals

Before a document can be queried, the search engine must know that it exists. On the web, this is achieved by *crawling*.

(Web crawlers are also known as spiders, robots, and bots.)

Crawlers attempt to visit every page of interest and retrieve them for

- Some websites return the same cont
- Some pages never return status 'don
- Some websites are not intended to be c

- Much web content is generated on-the-fly from databases, which can be costly for the content provider, so excessive numbers of visits to a site are unwelcome.
- Some content has a short lifespan.
- Some regions and content providers have low bandwidth.

Crawling

The observation that allows effective harvesting of the web is that it is a highly linked graph.

Assumption: If a web page is of interest, there will be a link to it from another page.

g

1. Create a prioritised list L of URL s that have been visited and when.

2. Repeat forever:
   1. Choose a URL u from L and fe
   2. Parse and index $p(u)$, and extract URLs $\{u'\}$ from $p(u)$.
   3. Add u to V and remove it from L. Add $\{u'\} - V$ to L.
   4. Process V to move expired or ‘old’ URLs to L.

In practice, page processing is much faster than URL resolution, so numerous streams of pages should be processed simultaneously.

Challenges

The list of URLS L must be prioritised to ensure that

- Every page is visited eventually.
- Synonym URLs are disregarded.
- Significant or dynamic pages are visited sufficiently frequently.
-

on a calendar can potentially be followed unt

The Robots Exclusion Standard defines a p supposed to observe. It allows website ma crawlers while allowing web browsing.

Simple crawlers are now part of programming languages, for example Perl's `LibWWW`, and good crawlers are available as part of systems such as `Nutch`.

Page recognition

Once a document has been fetched, it must be *parsed*.

That is, the words in the document are extracted, then added to a data structure that records which documents contain which words.

At the same time, information such as links and anchors can be

The most basic element is the character enc
captured in the page's metadata.

- For the first decade or so of the web, most
  (Want to travel in time? Try the **Wa**
  `waybackmachine.org/19970501000000*/http://cs.mu.oz.au`)
- HTML markup was used to provide an extended character set.
- ISO-8859 and ISO-8859-* now provide extended Latin character sets (Cyrillic, Thai, Greek, . . .)
- UTF-8 is the dominant character set covering the large-alphabet languages, with codes from 8 to 32 bits. The first 128 of the 8-bit codes are ASCII.

Page analysis

Web pages are supposed to be in HTML or XML (or sometimes in other formats, hence `ftp://` and so on).

The format separates user-visible content from metadata.

a

Many, many websites are not in conforman
be accidental, or can be a deliberate attempt t
known behaviour of particular browsers.

Parsers therefore need to be robust and flex

Some applications also make use of *scraping*, where only some components of the page are retained. For example, the advertisements and comments on a blog website might be ignored, with only blog content retained for indexing.

Tokenisation

Tokenisation

```
<head>
<META NAME="keywords" CONTENT="science humor, science humour, science,
humor, humour, ig-nobel, ig nobel, ignobel, hotair, hot-air, hot air,
improbable research">
<META HTTP-EQUIV="expires" CONTENT="0">
<title>HotAIR - Rare and well-done tidbits from the Annals of
Improbable Research</title>
</head>

<hr>

<img src="/toplevel/banner-2004.gif" width="406" height="200
alt="The Annals of Improbable Research: HotAIR">

<tr>
<td colspan=2 align="center">
<b><br><b>NOTE THIS:  JoAnn O'Linger-Luscusk and Alasdair
Skelton have <a href="/projects/hair/hair-club-top.html#newest">joined</a>
the Hair Club</b>
</td> </tr>
```

hotair rare and well done tidbits from the annals of improbable research note this joann o linger luscusk and alasdair skelton have joined the hair club

Tokenisation

The aim of parsing is to reduce a web page, or a query, to a sequence of *tokens*.

If the tokenisation is successful, the tokens in a query will match those of the web page, allowing query evaluation to proceed without any form of approximate matching.

- [illegible] one word or two? ‘Re-initialize’? ‘Under-standing’?
- Compounding. Is ‘football’ one word
- Possessives. Is ‘Zadek’s’ meant to b about ‘Smiths’?

Sometimes it is possible to disambiguate word senses, for example to separate ‘listen to the wind’ from ‘wind up the clock’, but in practice the error rate obliviates any possible gains.

In any case, such corrections are typically difficult or impossible in queries.

Canonicalisation

Any indexing process that relies on fact extraction may need information in a canonical form.

- Dates. Consider 5/4/2011, 4/5/2011, April 5 2011, first Tuesday in April 2001.
-

-
- Dr Who under Google?)
- Variant punctuation, 'e.g.' versus 'e

Historically, search engines discarded b terms such as `the`, `or`, and so on), but they now generally appear to be indexed.

They also discarded terms that linguistic rules suggested were not reasonable query strings, but anecdotally it is reported that they index *all* tokens of up to 64 characters.

Stemming

The most significant form of canonicalisation (for English) is arguably stemming.

his stem

Inflectional morphology: how a word is derived from a stem, for example *in+expense+ive* → *inex*

Stemming is the process of stripping away a

It can be challenging, because every word has a different set of legal suffixes.

Stemming

Different stemmers have different strengths, but the Porter stemmer (`www.tartarus.org/~martin/PorterStemmer` has several implementations) is the most popular.

- [illegible]
- [illegible]
- `ational` → `ate`
- `tional` → `tion`
- `tion` → `-`

Some versions of the stemmer constrain it so that the final result, or the stem produced at each step, must be a known word (either in a dictionary, or in the corpus being indexes).

Stemming example

`glasses` → `glass`

`posies` → `posi`

Other alternatives, like **lemmatisati**
dictionary entry, and constrain intermedi

Zoning

Web documents can usually be segmented into discrete zones such as title, anchor text, headings, and so on.

,

and compute similarities for documents by the fly.

Hence the observed behaviour of web sear that have the query terms in titles.

Indexing

Fast query evaluation makes use of an *index*: a data structure that maps terms to the documents that contain them. For example, the index of a book maps a few key terms to page numbers.

The only practical index structure for text qu *inverted index*: a collection of lists, one per term, recor documents containing that term.

An inverted index can be seen as the transposition of document-term frequency matrix accessed by (d, t) pairs into one accessed by (t, d) pairs.

Inverted index components

Search structure

For each distinct word t, the search structure contains:

- A pointer to the start of the corresponding inverted list.
-

Inverted lists

For each distinct word t, the inverted lis

- The identifiers d of documents containing t, as ordinal numbers.
- The associated frequency $f_{d,t}$ of t in d. (We could instead store $w_{d,t}$ or $w_{d,t}/W_d$.)

Example inverted index

Together with an array of W_d values (stored separately), the search structure and inverted index provide all the information required for Boolean and ranked query evaluation.

Example inverted index

For example:

$$\langle\ ,\qquad ,\ldots,\qquad ,\ldots,\qquad ,\ldots,\qquad ,\ldots,\qquad \mathrm{y},\ldots\rangle$$

$$\langle 0, 0, \ldots, 1, \ldots, 1$$

Example inverted index

Inverted index (one document):

happy	→	1(1)
of	→	1(1)
we	→	1(3)

Example inverted index

Inverted index (multiple documents):

...

band ... $(d, f_{d,\text{band}})$ ...

,

...

happy → ... → $(d, f_{d,\text{happy}}$

...

of → ... → $(d, f_{d,\text{of}})$

...

we → ... → $(d, f_{d,\text{we}})$ → ...

...

Example inverted index

An inverted index allows for fast querying because:
(1) the terms in the query correspond to the search structure
(2) the index only indicates documents where the term is *present*

Inverted index size

In a simple representation, for (say) a gigabyte of newswire data

- 12 MB (say) for 400,000 words, pointers, counts.
- 280 MB for 70,000,000 document identifiers (4 bytes each).
-

For 100 GB of web data, the total size is about 21
of the original text. (Many web pages contain l
unindexed data such as markup.)

Index construction and index maintenan
subject. But it is straightforward to build an index for a terabyte of text data on a current laptop in about a day.

Boolean Querying using a TDM

A term–document matrix of binary values is compact to store (1b per term per document), and the bitwise comparisons are fast to perform.

much larger.

Also, most of the values in the matrix are 0, whic
many, many comparisons for documents t
the query.

Boolean Querying using an inverted index

To evaluate a general Boolean query using an inverted index,

- Fetch the inverted list for each query term.
- Ignore within-document frequenci

For strictly conjunctive queries, query pro
shortest list as a set of *candidates*, a
do not appear in the other lists, working from second shortest to longest.

Ranked Querying principles

To produce a document ranking for a typical TF-IDF model, using the cosine as a similarity measure, we need the following information:

- The frequency of each query term in each document (TF)
-
-

$$S(q,d) \quad \overline{}$$

We then calculate the dot product, and then d

A TDM (32 bits per term per document) is too large to contemplate.

The structure of the inverted index is not designed to compare documents one at a time.

Ranked Querying using an inverted index

To use an inverted index to evaluate a query under the cosine measure,

1. t $A_d \leftarrow 0$.
2.

Set $A_d \leftarrow A_d + w_{q,t} \times w_{d,t}$

3. Read the array of W_d values an
   Set $A_d \leftarrow A_d / W_d$.

4. Identify the r greatest A_d value documents.

Ranked Querying using an inverted index ...

That is, starting with a set of N zero update the accumulators term by term.

Then use the document lengths to normalize each non-zero accumulator.

Accumulator costs

With the standard query evaluation algorithm and long queries, most accumulators are non-zero and an array is the most space- and time-efficient structure.

If only low f_t (that is, rare) terms are allow number of accumulators is greatly reduce

A simple mechanism is to impose a limit accumulators. This is another example of a compromise that alters the set of documents returned, and may therefore impact on effectiveness.

The “limiting” approach

1. Create an empty set A of accumulators.
2. For each query term t, ordered by decreasing $w_{q,t}$
    -
    - d,t

      set $A_d \leftarrow A_d + w_{q,t}$
3. For each accumulator set A_d
4. Identify their greatest A_d value

There are many variations on these algorithms.

The “thresholding” approach

1. Create an empty set A of accumulators, and set a threshold S.
2. For each query term t, ordered by decreasing w
   - [illegible]
   - [illegible]

create an accumulator A_d for d.
If d has an accumulator
set $A_d \leftarrow A_d + w_{q,t}$

3. For each accumulator set A_d [illegible]
4. Identify the r greatest A_d value

Querying costs

Several resources must be considered.

Disk space: for the index, at 40% of the size of the data. (With unstemmed terms, the index can be around 80% of the size of the data.)

of

CPU time: for processing inverted lists and updating accumulators.

Disk traffic: to fetch inverted lists.

By judicious use of compression and carefu
can be dramatically reduced compared to this first implementation. The gains are so great that it makes no sense to implement without some use of compression.

Phrase queries

Around 7% of the queries in the Excite log have an explicit phrase, such as "the great flydini".

A question for information retrieval resear
this lecture) is how best to use phrases in simil

A question for research in efficient query eva
pages in which the words occur as a phrase.

Phrase queries

The number of distinct phrases grows far more rapidly than the number of distinct terms. A small web crawl could easily contain a billion distinct two-word pairs, let alone longer phrases of interest.

There are three main strategies for phrase query evaluation:

- Process queries as bag-of-words, so that the terms can occur
  te

-

- Use some form of phrase index or word
  be directly identified without using the i

In this lecture, inverted lists have been descr
index entries, each an $\langle d, f_{d,t} \rangle$ pair. It is straightforward to include the $f_{d,t}$ ordinal word positions p at which t occurs in d:

$$\langle d, f_{d,t}, p_1, \ldots, p_{f_{d,t}} \rangle$$

Positions are word counts, not byte counts, so that they can be used to determine adjacency.

Phrase query evaluation

A phrase in a ranked query can be treated as an ordinary term – a lexical entity that occurs in given documents with given frequencies.

Similarity can therefore be computed in the usual way, but it is first

each document, along with in-document fr

- Fetch the inverted lists for each term.
- Take their intersection to find locatio

A similar strategy can be used for the more general task of determining whether query terms are proximate in a document.

Phrase query evaluation

Many phrases include common words. The cost of phrase query processing on an inverted index is dominated by the cost of fetching and decoding lists for these words, which typically occur at the start of or in

could involve intersecting the lists for rne, looking for positions p of lord such melbourne is at $p + 3$.

False matches could be eliminated by post-processing, or could simply be ignored.

Phrase query evaluation ...

Alternatively, it is straightforward to build a complete index of two-word phrases (around 50% of the size of the "web" data). Then evaluation of the phrase query:

and of

Proximity is an a variant, imprecise form of ph

- Favour documents where the terms a
- Search for 'phrases' where the terms distance of each other.

Proximity search involves intersection of inverted lists with word positions.

Link analysis

In general search, each document in considered independently.

(This can be spoofed by use of link farms, but with the kinds of analysis used by current engines it is extremely hard t

The two major link analysis algorithms are H topic search, not discussed in this subject) a

Pagerank overview

Basic intuition of PageRank: each web document has a fixed number of credits associated with it, a portion of which it redistributes to documents it links to; in turn, it receives credits from pages that point to it.

The final number of credits the page is left with determines its pagerank

tion of

probability $\alpha \in (0, 1)$. In this, we make the following assumptions:

- Each page has the same probability of random walk.
- For both teleports and traversal of out pages have an equal probability of being visited.

Some implementations of PageRank assign a maximum, fixed score to trusted pages, to seed the process.

Pagerank in practice

PageRank has a reputation for being critical to the performance of Google, and has attracted a great deal of research interest.

Analyses of Google searches has shown that in most cases the

- web site.
- The Aerospace home page only cont
- About 9[illegible]% of the within-RMIT ‘aeros Aerospace home page.
- Most of the links to the home page contain the word ‘aerospace’.

Anchor text is treated as a form of zone.

A high-performance web search engine

Further heuristics.

- Note which pages people actually visit by counting click-throughs.
- Manually alter the behavior of common queries.
-

-
- index of its documents.
  Then have multiple collections of ide

- Have separate servers for crawling a
- Accept feeds from dynamic data prov newspapers, and microblogging sites.
- Integrate diverse data resources, such as maps and directories.

Summary

- Search involves crawling, parsing, indexing, and querying; practical search also involves a range of other technologies.
- Crawling is in principle a straightforward application of queuing, but
-

- each word, rather than the list of words o
- The same structure is used for Boolea

- Approximations can be used to reduc affect the answer set in unpredictable
- On the web, link and anchor information can be the dominant evidence of relevance.

Readings

Brin, Sergey and Lawrence Page (1998). “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. Computer Networks 30: 107–117.

b Search

Zobel, Justin and Alistair Moffatt (2006). “In Engines”. ACM Computing Surveys 38 (2). doi:10.1145/1132956.1132959

Manning, Christopher D., Prabhakar Rag e (2008). “Introduction to Information Retrieval”. Chapters 1–2, 20–21. Cambridge University Press.

Pagerank algorithm

Input: D = document set
Output: Π_T = set of pagerank scores for each document $d_i \in D$

```
for all $d_i \in D$ do                                  ▷ Initialise the starting probabilities
    $\pi(d_{(i,0)}) \leftarrow \frac{1}{N}$                          ▷ $N$ is the total number of documents
end for
for $t = 1..T$ do                                        ▷ Repeat over $T$ iterations
                                                                        nt probabilities
            for all $d_j \in D$ do                       ▷ EITHER teleport randomly
                $\pi(d_{(j,t)}) \leftarrow \pi(d_{(j,t)}) + \alpha \times$          −
            end for
            for all $d_j$ where $d_i \to d_j$ do
                                                                        f $m$)
                $\pi(d_{(j,t)}) \leftarrow \pi(d_{(j,t)}) + (1 -$               −
            end for
        else
            for all $d_j \in D$ do                       ▷ teleport to a random document
                $\pi(d_{(j,t)}) \leftarrow \pi(d_{(j,t)}) + \pi(d_{(i,t-1)}) \times \frac{1}{N}$
            end for
        end if
    end for
end for
```

Pagerank example

Assume a set of two documents, d_1 and d_2, with a link from d_1 to d_2.

0 0

$\times 0.8 +$

$. \times . \times . + . \times . = . . . + .$ $0.8 +$

0.38

3 $0.38 \times 0.2 \times 0.5 + 0.62 \times 0.5 =$ $+$

0.348

⋮