Search Engines for the Web

An Overview

- Brin and Page: *The Anatomy of a Large-Scale Hypertextual Web Search Engine*. 7th Int. WWW conference, 1998

Information Retrieval

- Process data, build index
- Query the index:
  - Find all documents relevant to query
  - Rank documents, show most relevant first

Classic Information Retrieval (IR):

Methods developed for small to medium sized homogeneous collections of text documents.

Examples: Scientific document collections, news collections, libraries.

IR on the Web

Difficulties:

- Documents not local.
- Documents very heterogeneous.
- Documents constantly changing in contents and number.
- Very large document collection (billions of documents, total size measured in Terabytes).
  - Storage and performance are important issues. Distribution and parallelism necessary.
  - Many (e.g. 100.000) relevant documents for most queries. Good ranking methods are essential.

Advantages:

- Extra structure on document collection: links.

Further Challenges of the Web

- Many near-duplicate documents (30%)
- Users heterogeneous and impatient. Advanced search interfaces not viable.
- How to search and index non-text documents.
  - Multimedia contents.
  - Database interfaces.

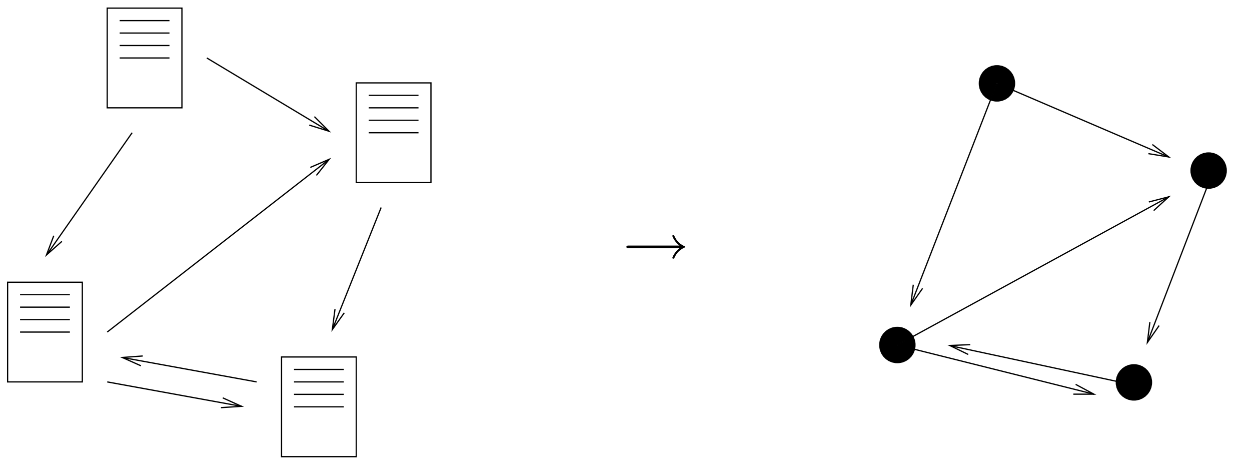
  This course: only considers text documents.

The Web as a Graph

Note:

$\boxed{\text{WWW = an oriented graph}}$

nodes	=	pages (URL's)
edges	=	links



Basic Tasks of Search Engines

Gather data:

- Web crawling (traversal of the web graph)

Index data:

- Parse documents
- Lexicon: index (dictionary) over all words encountered.
- Inverted file: for all words in lexicon, list in which documents they appear.

Search in data:

- Find all relevant documents (those containing the search phrases).
- Rank the documents.

Lexicon

For one billion documents:

Inverted files $\sim$ total number of words $\geq 100 \cdot 10^9$ Disk

Lexicon $\sim$ number of different words $\sim 10^6$ RAM

Can reside in RAM $\Rightarrow$ standard dictionary structures OK.
Examples:

- Binary search in sorted list of words.
- Hash tabels.
- Tries, suffix trees, suffix arrays.

Inverted File

- Simple (appearance of word in document):

  word$_1$: DocID, DocID, DocID
  word$_2$: DocID, DocID
  word$_3$: DocID, DocID, DocID, DocID, DocID,...
  $\vdots$

- Detailed (*all* appearances of word in document):

  word$_1$: DocID, Position, Position, DocID, Position...
  $\vdots$

- Even more detailed:

  Appearance annotated with info (heading, boldface, anchor text,...). Useful during ranking.

Constructing index

```
foreach document $D$ in collection
    Parse $D$ and identify words
    foreach word $w$
        output (DocID, $w$)
        if $w$ not in lexicon
            insert $w$ in lexicon
```

$$\Downarrow$$

$(1,2),(1,37),\ldots,(1,123)$, $(2,34),(2,37),\ldots,(2,101)$, $(3,486),\ldots$

External Sorting $\surd$ $\Downarrow$ Hashing $\div$

$(22,1),(77,1),\ldots,(198,1)$, $(1,2),(22,2),\ldots,(345,2)$, $(67,3),\ldots$

$\approx$ inverted file

Searching and Ranking

Query: computer AND science:

1. Look up computer and science in lexicon. This gives positions on disk where their lists start.
2. Scan these lists and merge them (find DocIDs which are included in both lists by doing simultaneous scans).

   computer: 12, 15, 117, 155, 256,. . .
   science: 5, 27, 117, 119, 256,. . .
3. Calculate rank of the returned DocIDs. Fetch the 10 highest ranked from the document collection, and return URL and some textual context from documents to the user.

OR and NOT works similarly. If lists have word positions, phrase-searches (“computer science”) and proximity searches (“computer” close to “science”) can also be done.

Text Based Ranking

Add weight to appearance of word in document according to e.g.

- Number of appearances of word in document.
- Typographic emphasis (boldface, headline,. . .)
- Appearance in META-tags.
- Appearance around links pointing to the document

Improves text based ranking, but still not good enough on the web (where ranking of e.g. 100.000 relevant documents is common).

Also: too easy to influence (spam) the ranking by adding keywords to the page.

Link Based Ranking

Idea 1: Link to page $\approx$ recommendation of page.

$\Rightarrow$ Rank of page: its indegree in the web graph.

Still very easy to spam (create lots of links to the page in question).

Linkbaseret ranking

Idea 1: Link to page $\approx$ recommendation of page.

Idea 2: Recommendations from important pages count more.

PageRank:

Find values r_j fulfilling $$r_j = \sum_{i \in B_j} r_i / N_i$$ for all j, where

r_j = PageRank of page j,
B_j = set of pages linking to page j,
N_i = links out of page i (i.e. its outdegree)

I.e. find $\vec{r} = (r_1, r_2, \ldots, r_n)$ such that $\vec{r} = \vec{r}A$,

where A = normalized adjacency matrix for the web graph
(normalized: entries in row i is $1/N_i$ instead of 1).

Calculation of PageRank

In short, the PageRank vector $\vec{r}$ is defined as an eigenvector for A, i.e. a vector fulfilling:

$$\vec{r} = \vec{r}A$$

From exising mathematical theory (the Ergodic Theorem on random walks) we get:

If A fulfills certain conditions, such a vector $\vec{r}$ does exist, and for any initial vector x we have:

$$\vec{x}A^k \rightarrow \vec{r} \quad \text{for} \quad k \rightarrow \infty$$

Calculation of PageRank

To fulfill the conditions, exchange A by A' defined as follows:

$$A' = 0.85A + 0.15E\ ,$$

where E is the normalized adjacency matrix for the graph containing all possible edges (i.e. the clique on the set of all nodes). The split 85–15% is not central, but is chosen because it has proven to work well in practice.

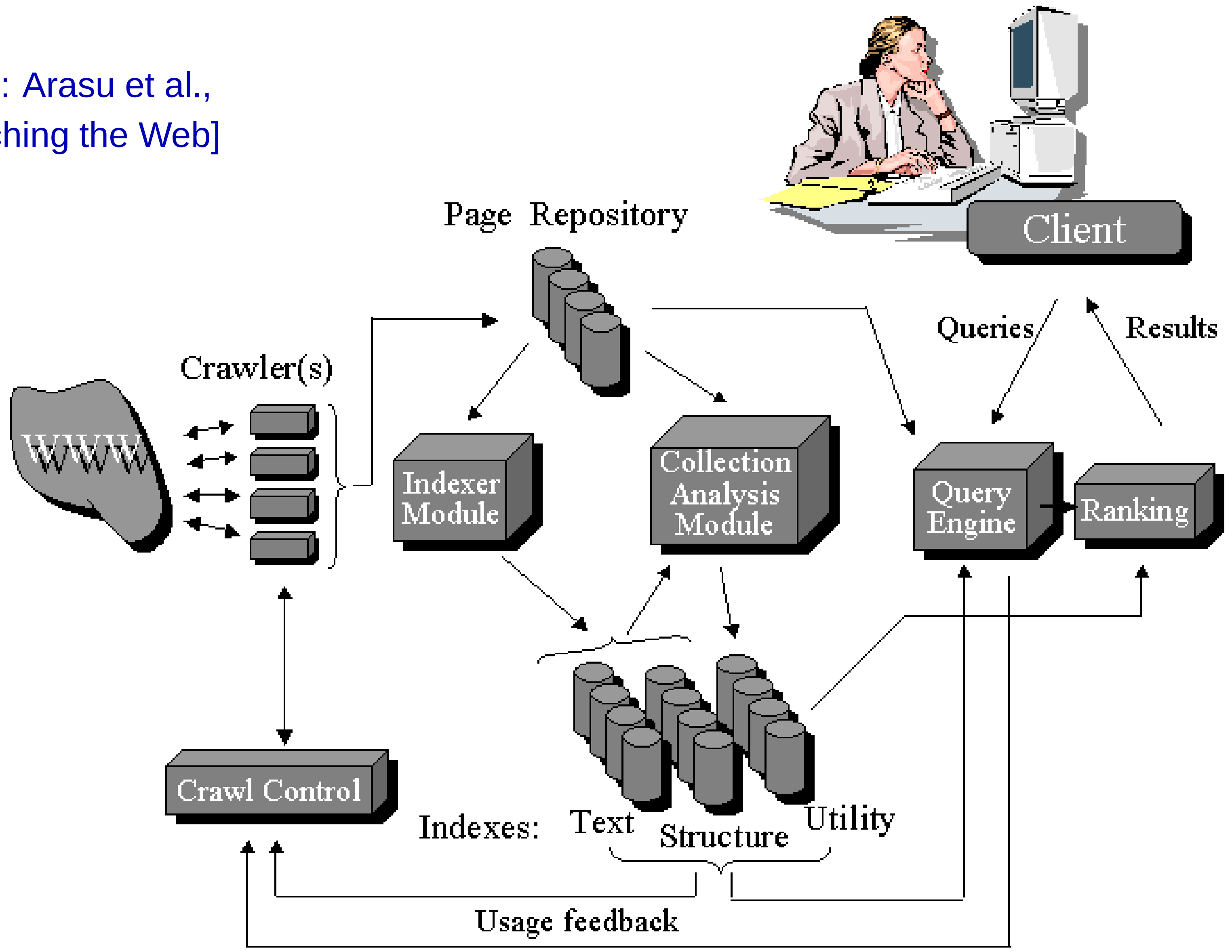
Calculation of PageRank: From some arbitrary start vector r (not null), repeat

$$\vec{r}_{\text{new}} = \vec{r}_{\text{old}} A'$$

In practice, convergence towards the eigenvector is fast: The value of $\vec{r}$ typically stabilizes after 20-50 iterations. Then the process is stopped and the resulting r used as the PageRank.

Search Engine, General Structure

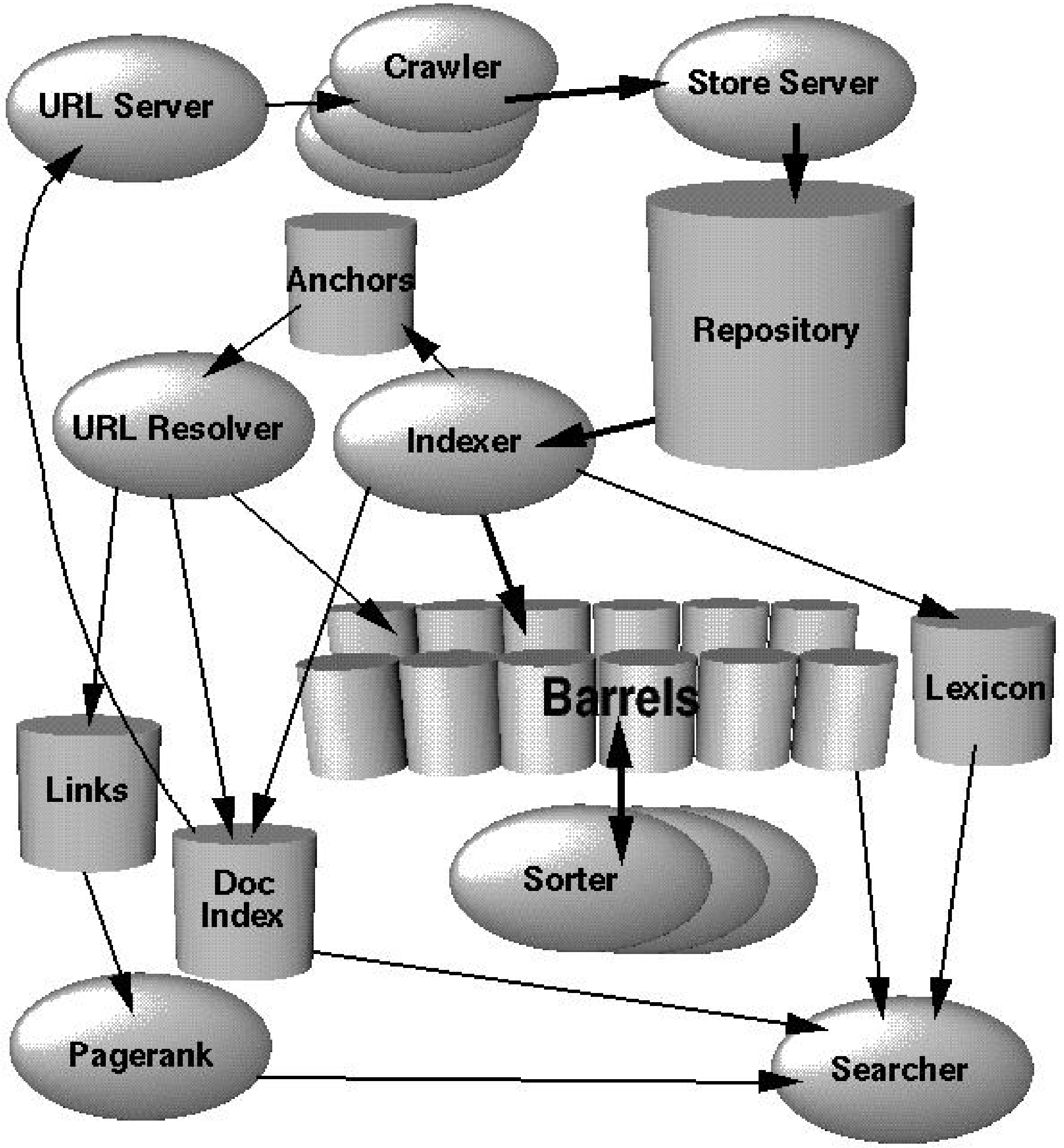
[From: Arasu et al., Searching the Web]



Specific Example

Google:
(1998)

[From:
Brin and Page,
Anatomy of. . .]


URL Server
Crawler
Store Server
Anchors
Repository
URL Resolver
Indexer
Barrels
Lexicon
Links
Doc Index
Sorter
Pagerank
Searcher