DR.BABASAHEB AMBEDKAR MARATHWADA UNIVERSITY,AURANGABAD

Sagar B.C.A. College, Jalna

A Project Report On

Information Retrieval on the Internet

Submitted To

UNDER THE GUIDANCE OF

AJITKUMAR

Submitted by

KHARAT GANESH SUDAM

M.LiB SY

Year 2022-23

Department of Master Of Library and Information Science

(M.Lib. SY)

CERTIFICATE

This is to certify that, the following student

Kharat Ganesh Sudam

Has successfully completed the summer internship project

Information Retrieval on the Internet

In the partial fulfillment of the requirement of Master Of Library and Information Science course as expected by **Dr. Babasaheb Ambedkar Marathwada University, Aurangabad** for Academic Year **2022-23**.

[illegible]
External Examiner

[illegible]
Internal Examiner
Head
Department of Humanities
Sagar BCA College, Jalna.

[illegible]
Principal
N. D. NAJARDHANE

STUDENT DECLARATION

This is to declare that this Summer Training Project report on **"Information Retrieval on the Internet"** is a record of genuine work done by me under the guidance of **AJITKUMAR** in the partial fulfillment to the requirement for Master Of Library and Information Science I declare that this Summer Training project report is original and not submitted to anyother university before.

[illegible]

Signature of the Student:

Student's Name: Kharat Ganesh Sudam

PRN No. 2021015200526883

Exam Seat No. CMLD401230

ACKNOWLEDGEMENT

While conducting this report, I got support in many ways from many people .First I am deeply grateful to my project guide, **AJITKUMAR** who helped me with full devotion and always supported me earnestly whenever it was needed. Without his guidance, mental &moral support and academic inputs this report was not possible.

This Training report could never have seen the light of the day without his co- operation of those Clients who participated in this. I am thankful to all of them for giving me their valuable time.

My friends have been biggest support for me at every juncture of life. They manifested their great interest in my research work also and always tried to make thingseasy for me.

A word of gratitude goes to my family members whose love; affection and understanding have enabled me to complete this end with ease.

At the end, I thank to Almighty for giving me courage and strength to conduct this project report.

(KHARAT GANESH)

Information Retrieval on the Internet

Outline

1. Introduction, 1-1 Search Engines, 1-2 Search Engine History, 1-3 Search Engine Features and Services, 1-4 Search Engine Architectures
2. Web Crawling, 2-1 Robots Exclusion Protocol, 2-2 Robots Meta Tag, 2-3 Multi-Threaded Spidering , 2-4 Focused Spidering, 2-5 Keeping Spidered Pages Up-to-Date
3. The Information Retrieval System, 3-1 Preprocessing the Document Collection, 3-2 Information Retrieval Models, 3-2-1 The Boolean Model, 3-2-2 The Vector Space Model, 3-2-3 Latent Semantic Indexing, 3-2-4 The Probabilistic Model, 3-4 Relevance Feedback
4. Evaluation of Information Retrieval Systems, 4-1 Precision and Recall, 4-2 F-measure and E-measure, 4-3 Mean Average Precision, 4-4 Novelty Ratio and Coverage Ratio
5. Practical Considerations, 5-1 Efficient Indexing, 5-2 Examples of Information Retrieval Systems, 5-2-1 SMART, 5-2-2 Lucene, 5-2-3 Lemur, 5-2-4 OKAPI, 5-3 Example of Large-Scale Search Engine Architecture: Google, 5-4 Page Ranking Algorithms, 5-4-1 Hubs and Authorities, 5-4-2 Google's PageRank, 5-5 Commercial Aspects of Search Engines: Sponsored Links
6. The Invisible Web, 6-1 Searchable Databases, 6-2 Excluded Pages
7. Other Types of Information Retrieval Systems, 7-1 Multimedia Information Retrieval, 7-2 Digital Libraries, 7-3 Distributed Information Retrieval Systems
8. Conclusion and Future Directions, 8-1 Natural Language Queries, 8-2 The Semantic Web and Use of Meta-Data, 8-3 Visualization and Categorization of Results
9. Glossary, Bibliography

Abstract

The main components of a search engine are the Web crawler which has the task of collecting webpages and the Information Retrieval system which has the task of retrieving text documents that answer a user query. In this chapter we present approached to Web crawling, Information Retrieval models, and methods used to evaluate the retrieval performance. Practical considerations include information about existing IR systems and a detailed example of a large-scale search engine (Google), including the idea of ranking webpages by their importance (the Hubs an Authorities algorithm, and Google's PageRank algorithm). Then we discuss the Invisible Web, the part of the Web that is not indexed by search engines. We briefly present other types of IR systems: digital libraries, multimedia retrieval systems (music, video, etc.), and distributed IR systems. We conclude with a discussion of the Semantic Web and future trends in visualizing search results and inputting queries in natural language.

INTRODUCTION

There is a huge quantity of text, audio, video, and other documents available on the Internet, on about any subject. Users need to be able to find relevant information to satisfy their particular information needs. There are two ways of searching for information: to use a search engines or to browse directories organized by categories (such as Yahoo Directories). There is still a large part of the Internet that is not accessible (for example private databases and intranets).

Information retrieval (IR) is the task of representing, storing, organizing, and offering access to information items. IR is different from data retrieval, which is about finding precise data in databases with a given structure. In IR systems, the information is not structured, it is contained in free form in text (webpages or other documents) or in multimedia content. The first IR systems implemented in 1970's were designed to work with small collections of text (for example legal documents). Some of these techniques are now used in search engines.

In this chapter we describe information retrieval techniques, focusing on the challenges faced by search engines. One particular challenge is the large scale, given by the huge number of webpages available on the Internet (for example, about 8 billion webpages were indexed by Google in 2005). Another challenge is inherent to any information retrieval system that deals with text: the ambiguity of the natural language (English or other languages) that makes it difficult to have perfect matches between documents and user queries.

The organization of this chapter is as follows. We briefly mention the search engines history, features, and services. We present the generic architecture of a search engine. We discuss its Web crawling component, which has the task to collect webpages to be indexed. Then we focus on the Information Retrieval component which has the task of retrieving documents (mainly text documents) that answer a user query. We present current methods used to evaluate the performance of the Information Retrieval component. Practical considerations include information about existing IR systems and a detailed example of a large-scale search engine (Google); we present methods for ranking webpages by their importance (the Hubs an Authorities algorithm and Google's PageRank algorithm). In another section, we discuss the Invisible Web, the part of the Web that is not indexed by search engines. We briefly present other types of IR systems: digital libraries, multimedia IR systems, and distributed IR systems. We conclude with a discussion of the Semantic Web and other future trends.

Search engines

There are many general-purpose search engines available on the Web. A resource containing up-to-date information on the most used search engines is: http://www.searchenginewatch.com.

Here are some popular search engines (in alphabetic order):

AllTheWeb http://www.alltheweb.com/
AltaVista http://www.altavista.com/
Excite http://www.excite.com/
Google http://www.google.com/
Hotbot http://www.hotbot.com/
Lycos http://www.lycos.com/
MSN Search http://search.msn.com/
Teoma http://teoma.com/
WiseNut http://www.wisenut.com/
Yahoo! http://search.yahoo.com/

Meta-search engines combine several existing search engines in order to provide documents relevant to a user query. Their task is reduced to ranking results from the different search engines and eliminating duplicates. Some examples are: http://www.metacrawler.com/, http://www.mamma.com/, and http://www.dogpile.com/.

Search Engine History

The very first tool used for searching on the Internet was called Archie (the name stands for "archive"). It was created in 1990 by Alan Emtage, a student at McGill University in Montreal. The program downloaded the directory listings of all the files located on public anonymous FTP sites, creating a searchable database of filenames. Gopher was created in 1991 by Mark McCahill at the University of Minnesota. While Archie indexed file names, Gopher indexed plain text documents. Two other programs, Veronica and Jughead, searched the files stored in Gopher index systems.

The first Web search engine used Wandex, a now-defunct index collected by the World Wide Web Wanderer, a web crawler developed by Matthew Gray at MIT in 1993. Another very early search engine, Aliweb, also appeared in 1993, and still runs today. The first "full text" crawler-based search engine was WebCrawler, 1994. Unlike its predecessors, it let users search for any word in any web page; this became the standard for all major search engines ever since. It was also the first one to be widely known to the public. Also in 1994, Lycos (which started at Carnegie Mellon University) came out, and became a major commercial endeavor.

Soon after, many search engines appeared and became popular. These included Excite, Infoseek, Inktomi, Northern Light, and AltaVista. In some ways, they competed with popular directories such as Yahoo!. Later, the directories integrated or added on search engine technology for greater functionality.

Search engines were also known for the Internet investing frenzy that occurred in the late 1990s. Several companies entered the market spectacularly, with record gains during their initial public offerings. Some have taken down their public search engine, and are marketing enterprise-only editions, such as Northern Light.

Around 2001, the Google search engine rose to prominence (Page and Brin, 1998). Its success was based in part on the concept of link popularity and PageRank, that uses the premise that good or desirable pages are pointed to by more pages than others. Google's minimalist user interface was very popular with users, and has since spawned a number of imitators. Google is currently the most popular search engine. In 2005, it indexed approximately 8 billion pages, more than any other search engine. It also offers a growing range of Web services, such as Google Maps and online automatic translation tools.

In 2002, Yahoo! acquired Inktomi and in 2003, Yahoo! acquired Overture, which owned AlltheWeb and AltaVista. Despite owning its own search engine, Yahoo initially kept using Google to provide its users with search results. In 2004, Yahoo! launched its own search engine based on the combined technologies of its acquisitions and providing a service that gave pre-eminence to the Web search engine over its manually-maintained subject directory.

MSN Search is a search engine owned by Microsoft, which previously relied on others for its search engine listings. In early 2005 it started showing its own results, collected by its own crawler. Many other search engines tend to be *portals* that merely show the results from another company's search engine. For more details and search engine timelines see, for example, http://en.wikipedia.org/wiki/Search_engine.

Search Engine Features and Services

Search engines allow a user to input keywords that describe an information need. The also offer advanced search capabilities. Although they lead to more precise, they are less utilized by users. We briefly discuss some advanced search features. Boolean features (AND, OR, NOT) that allow retrieval of documents that contain all the keywords (AND), any of the keywords (OR), exclude some words (NOT), or combinations of these Boolean operators. The proximity feature searches for phrases or consecutive words (usually simple search can do this if the words are surrounded by double quotes). The search can be done only in particular fields, such as URLs or titles. Limits can be imposed on the type of retrieved pages: date, language, file types, etc.

Some search engines also offer services: news directories, image search, maps (such as Google Maps), language tools (such as automatic translation tools or interfaces in particular languages), newsgroup search, and other specialized searches.

Search Engine Architectures

The components of a search engine are: Web crawling (gathering webpages), indexing (representing and storing the information), retrieval (being able to retrieve documents relevant to user queries), and ranking the results in their order of relevance. Figure 1 presents a simplified view of the components of a search engine. More details about the main module, the IR system, will follow in the next sections.

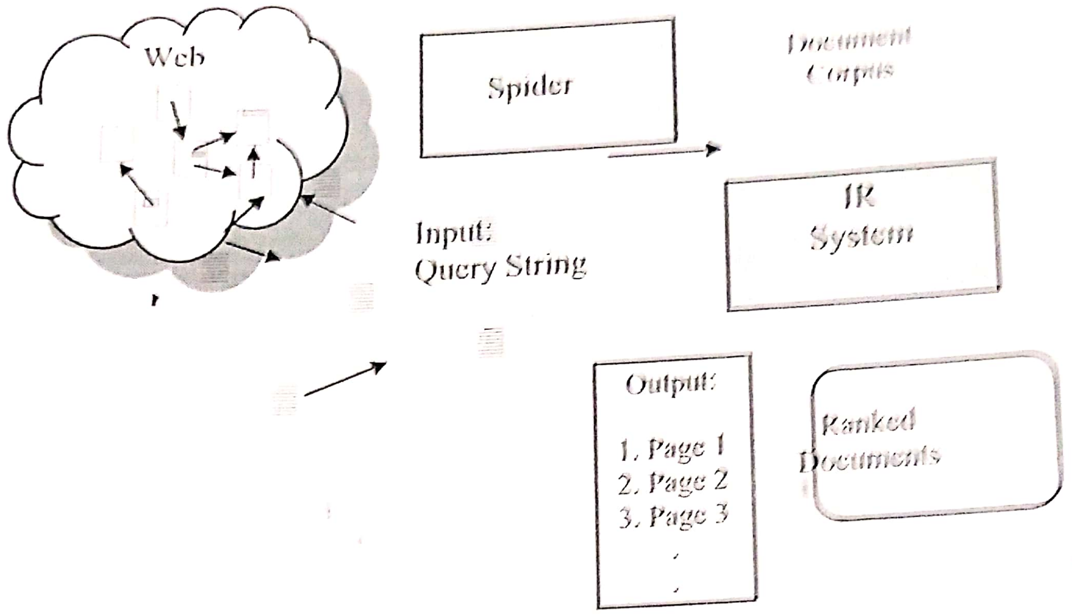


Figure 1. The simplified architecture of a search engine.

WEB CRAWLING

Web crawlers, also known as spiders or robots, have the task to collect webpages to build the text collection for the IR system. The text is extracted from the HTML code of the webpages. Some information related to the HTML format may be stored too. For example, text in headings or in bold font can be given higher weight than the rest of the text.

A crawler starts with one or more http addresses (a set of root URLs), and follows all the links on these pages recursively, to find additional pages. It can proceed by depth-first searching (follow the first link in a page and all the links in the new pages that it leads to, then come back o follow the rest of the links in the current page) or by breadth-first searching (follow all the inks in the page for one step, then the links in the pages they point to, for one step, etc.). 3readth-first has the advantage that it explores uniformly outward from the root page but equires memory to store all the links waiting to be traversed on the previous level (exponential n the depth of the links structure). It is the standard spidering method. Depth-first requires less nemory (linear in depth) but it might get "lost" pursuing a single thread. Both strategies can be asily implemented using a queue of links (URLs) to be visited next. How new links are added to ne queue determines the search strategy. FIFO (first in first out, append to end of the queue) ives breadth-first search. LIFO (last in first out, add to front of queue) gives depth-first search. euristically ordering the queue gives a "focused crawler" that directs its search towards nteresting" pages. A spider needs to avoid visiting the same pages again when the links are rcular; it needs to keep track of pages that were already visited.

To extract links from a webpage in order to collect candidate links to follow, HTML yperlink fields are parsed. Here are two examples of hyperlinks:

```
<a href="http:/www.site.uottawa.ca/~diana/csi4107">
<frame src="site-index.html">
```

If the URL is not specified, like in the last example, the link is relative to the current base URL. If a file name is not specified, a default name is used (such as index.hml). The links are put into canonical form: the ending slash is removed, if there is one; internal references within the same page are removed, etc. Once the pages are collected, the text is extracted from the HTML documents, to be processed by the IR system.

Robot exclusion protocols are used to prevent certain sites or webpages from being indexed by Web crawlers. Web sites and pages can specify that robots should not crawl or index certain areas, by using the Robots Exclusion Protocol or the robots meta tag. The second one is newer and less well-adopted than the first one. These standards are conventions to be followed by "good robots". They cannot be enforced, but companies have been prosecuted for "disobeying" these conventions and "trespassing" on private cyberspace.

The Robots Exclusion Protocol

The Robots Exclusion Protocol is a site-wide specification of excluded directories. The site administrator has to put a "robots.txt" file at the root of the host's Web directory. See for example http://www.ebay.com/robots.txt. The file "robots.txt" is a list of excluded directories for a given robot (user-agent). This file contains blank lines to separate different user-agent disallowed directories, with one directory per "Disallow" line. No regular expression can be used as patterns for directories.

To exclude all robots from the entire site, the file would contain:

```
User-agent: *
Disallow: /
```

To exclude specific directories:

```
User-agent: *
Disallow: /tmp/
Disallow: /cgi-bin/
Disallow: /users/paranoid/
```

To exclude a specific robot:

```
User-agent: GoogleBot
Disallow: /
```

To allow a specific robot:

```
User-agent: GoogleBot
Disallow:
```

The Robots Meta Tag

An individual document tag can be used to exclude indexing or following links in a particular webpage. The HEAD section of a specific HTML document can include a robots meta tag, such as `<meta name="robots" content="none">`. The content value can be a pair of values for two aspects: `index` or `noindex` for allowing or disallowing the indexing of this page, and `follow` or `nofollow` for allowing or disallowing following the links in this page. There are two special values: `all = index,follow` and `none = noindex,nofollow`. Examples:

```
<meta name="robots" content="noindex,follow">
<meta name="robots" content="index,nofollow">
<meta name="robots" content="none">
```

Multi-Threaded Spidering

Network delays are frequent when downloading individual pages. It is best to have multiple threads running in parallel, each requesting a page from a different host. The URL's can be distributed to threads, to guarantee equitable distribution of requests across different hosts, in order to maximize through-put and avoid overloading any single server. For example, early Google spiders had multiple coordinated crawlers with about 300 threads each, together being able to download over 100 pages per second.

Focused Spidering

More "interesting" pages could be explored first. There are two styles of focus: topic-directed and link-directed. For the former, if the desired topic description or sample pages of interest are given, the spidering algorithm could sort the queue of links by the similarity (e.g. cosine metric) of their source pages and/or anchor text to this topic description. For the latter, the spider could keep track of in-degree and out-degree of each encountered page, and sort the queue to prefer popular pages with many in-coming links (*authorities*), or to prefer summary pages with many out-going links (*hubs*). See the section on page ranking algorithms for more details.

Keeping Spidered Pages Up-to-Date

The Web is very dynamic: there are many new pages, updated pages, deleted pages, etc. A search engine needs to periodically check spidered pages for updates and deletions. A spider could look in the HTML head information (e.g. meta tags on the last update) to determine if the page has changed, and only reload the entire the page if needed. It could track how often each page is updated and preferentially return to pages which are historically more dynamic. It could preferentially update pages that are accessed more often to optimize the freshness of popular pages.

THE INFORMATION RETRIEVAL SYSTEM

Figure 2 presents a more detailed view of the architecture of an IR system (Baeza-Yates and Berthier Ribeiro-Neto, 1999). Text Operations are used to preprocess the documents collections and to extract index words. The indexing module constructs an inverted index from words to document pointers. The searching module retrieves documents that contain given query words, using the inverted index. The ranking module scores all the retrieved documents according to a relevance metric. The user interface manages interaction with the user: the input of the query and the output of the ranked documents, including the visualization of results. The query operations can transform the query in order to improve retrieval (query expansion using synonyms from a thesaurus, query transformation using relevance feedback).

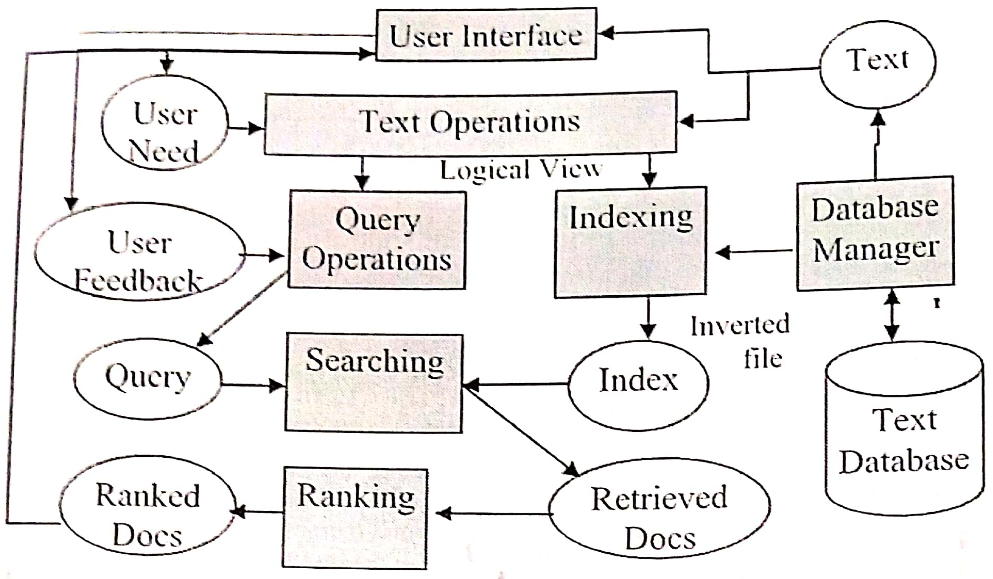


Figure 2. The architecture of an IR system: Text operations are applied of the text of the documents and on the description of the user information need in order to transform them in a simplified form needed for computation. The documents are indexed and the index is used to execute the search. After ranked documents are retrieved, the user can provide feedback which can be used to refine the query and restart the search for improved results.

Preprocessing the document collection

There are several preprocessing steps needed to prepare the document collection for the IR task. The first step is to filter out unwanted characters and markup (e.g. HTML tags, punctuation, numbers, etc.). Then the text needs to be broken into tokens (keywords) by using as delimiters white space and punctuation characters. This it not quite as straightforward as it seems, since words in texts are not always clearly delimited (for example, if the text is *You can't do this*, you can consider the apostrophe as a word separator to get two words *can* and *t*, or ignore it as separator and consider one word *can't*, or expand the contacted form into two words *can* and *not* and use the white space as separator).

The keywords can be used as they are, or they can be transformed into a base form, for example nouns in the singular form, verbs in the infinitive form, etc. (e.g., *books* becomes *book*, *talked* becomes *talk*). A common approach is to stem the tokens to "stem" forms. For example, *computational* becomes *comput* and *computing* becomes *comput*. Stemming the terms before building the inverted index has the advantage that it reduces the size of the index, and allows for retrieval of webpages with various inflected forms of a word (for example, when searching for webpages with the word *computation*, the results will include webpages with *computations* and *computing*). Stemming is easier to do than computing base forms, because stemmers remove suffixes, without needing a full dictionary of words in a language. A popular and fast stemmer is Porter's stemmer.

Another useful preprocessing step is to remove very frequent words that appear in most of the documents and do not bear any meaningful content. They are called stopwords (e.g., *a*, *the*, *it*, *of*, *could*, etc.). An example of stopwords list can be found at:
http://www.lextek.com/manuals/onix/stopwords1.html.

Important phrases composed of two or more words could also be detected to be used as keywords (possibly using a domain specific dictionary, or using a statistical method for analyzing the text collection in order to detect sequences of words that appear together very often).

Now the text is ready for the next step, building the inverted index that stores for each keyword a list of documents that contain it, in order to allow for fast access during the retrieval step.

Information Retrieval Models

This section presents information retrieval models that can be applied on any text collection. Not all the IR models are easily scaled up to be able to deal with a very large collection, such as pages collected from the Web. The most important IR models are: the Boolean Model, the Vector Space Model, and the Probabilistic Model. Various extensions of these models are possible. We discuss one of them here, Latent Semantic Indexing, which is an extension of the Vector Space Model.

The Boolean Model

The Boolean model is the simplest to implement. A document is represented as a set of keywords. Queries are Boolean expressions of keywords, connected by AND, OR, and NOT, including the use of brackets to indicate the scope of these operators. For example, the query "all the hotels in Rio Brazil or Hilo Hawaii, but not Hilton" is typed by the user as:

[[Rio & Brazil] | [Hilo & Hawaii]] & hotel & !Hilton]

The output of the system is a list of documents that are relevant, but there will be no partial matches or ranking. The Boolean model is very rigid: AND means "all"; OR means "any". All matched documents will be returned, making it difficult to control the number of documents retrieved. All the matched documents satisfy the query to the same degree; that makes it difficult to rank the output. Another disadvantage of this model is that is it not easy for the users to express complex queries.

The Vector Space Model

The vector space model of information retrieval is a very successful statistical method proposed by Salton (1989). It generates weighted term vectors for each document in the collection, and for the user query. Then the retrieval is based on the *similarity* between the query vector and document vectors. The output documents are ranked according to this similarity. The similarity is based on the occurrence *frequencies* of the keywords in the query and in the documents.

Let's assume that t distinct terms remain after preprocessing; let's call them index terms or the vocabulary. These terms form a vector space with dimensionality t, the size of the vocabulary. Each term i, in a document j, is given a weight w_{ij}. Both the documents and the queries are expressed as t-dimensional vectors: $d_i = (w_{1j}, w_{2j}, \ldots, w_{tj})$.

A collection of N documents can be represented in the vector space model by a documents-by-terms matrix. An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document; it simply doesn't appear in the document. The matrix tends to contain lots of zeros.

$$\begin{array}{c|cccc}
 & T_1 & T_2 & \cdots & T_t \\
\hline
d_1 & w_{11} & w_{21} & \cdots & w_{t1} \\
d_2 & w_{12} & w_{22} & \cdots & w_{t2} \\
\vdots & \vdots & \vdots & & \vdots \\
\vdots & \vdots & \vdots & & \vdots \\
d_N & w_{1N} & w_{2N} & \cdots & w_{tN}
\end{array}$$

The weights in the matrix can be 1 if the term occurs in the document and 0 if it does not (binary weights); but the more frequent terms in a document are more important, i.e., more indicative of the topic. Therefore it is good to use the frequencies of the terms as weights.
Let f_{ij} be the frequency of the term T_i in the document d_j
We can normalize the *term frequency* (*tf*) across the entire corpus: $tf_{ij} = f_{ij} / max\{f_{ij}\}$. Terms that appear in many *different* documents are *less* indicative of overall topic.
Let df_i be the document frequency of term T_i – the number of documents containing the term i, and let idf_i be the inverse document frequency of term T_i:

$$idf_i = log\ (N/df_i)$$

(where N is the total number of documents). The *idf* value is an indication of a term's *discrimination* power. The logarithm is used to dampen the effect relative to *tf*. A typical combined term importance indicator is *tf-idf* weighting:

$$w_{ij} = tf_{ij} \cdot idf_i = tf_{ij}\ log\ (N/df_i).$$

A term occurring frequently in the document but rarely in the rest of the collection is given high weight. Many other ways of determining term weights have been proposed. Experimentally, *tf-idf* has been found to work well.

The query is also transformed into a vector. It is typically treated as a document and also *tf-idf* weighted. Alternatively, the user could supply weights for the given query terms.

The similarity between vectors for the document d_j and the query q can be computed as the vector inner product:

$$\text{sim}(d_j, q) = \vec{d_j} \bullet \vec{q} = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

where w_{ij} is the weight of term i in document j and w_{iq} is the weight of term i in the query

For binary vectors, the inner product is the number of matched query terms in the document (the size of the intersection). For weighted term vectors, it is the sum of the products of the weights of the matched terms. There are several problems with the inner product: it does not have a bounded range of values; it favors long documents with a large number of unique terms; it measures how many terms are matched but not how many terms are *not* matched.

The cosine similarity measure tends to work better. The formula is the same as the inner product, but it is normalized by the length of the documents and the length of the query (the length of a vector is the square root of the sum of the squares of its components).

$$\text{cosSim}(\vec{d_j}, \vec{q}) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

The cosine measures the angles between the two vectors (the higher the cosine value closer to 1, the smaller the angle between the vector of the document and the vector of the query, therefore a more relevant document). Because we consider only the angle, the length of the documents is not a problem anymore.

A naïve implementation of the vector space retrieval is straightforward but impractical: convert all the documents in collection C to *tf-idf* weighted vectors, for all the keywords in the vocabulary *V*; convert the query to a *tf-idf*-weighted vector *q*; then for each document *d* in *C* compute cosSim(*d*, *q*); sort the documents by decreasing score and present top-ranked documents to the user. The time complexity would be O(|*V*|·|C|). It would take very long for large *V* and *C* (for example, if |*V*| = 10,000 and |C| = 100,000 then |*V*|·|C| = 1,000,000,000).

A practical implementation is based on the observation that documents containing none of the query words do not affect the final ranking. Identifying those documents that contain at least one query word is easily done by using an inverted index. The numerator in the cosine similarity formula will be calculated much faster because the multiplications where one of the terms is zero will not be executed.

The steps of a practical implementation are as follows. Step 1, pre-processing (tokenization, stopword removal, stemming), determines the keywords in the vocabulary to be used as index terms. Step 2 is the building of the inverted index, with an entry for each keyword in the vocabulary (see Figure 3). The index is a data structure that will allow fast access in the retrieval step (hash table, B-tree, sparse list, etc.) For each keyword, the index keeps a list of all the documents where it appears together with the corresponding term frequency (*tf*). It also keeps the total number of occurrences in all documents (for the *idf* score). So the *tf-idf* scores can be computed in one pass trough the collection. The cosine similarity also requires document lengths: a second pass to is needed to compute document vector lengths. The time complexity of indexing a document of *n* tokens is O(*n*). So indexing *m* such documents takes O(*m n*). Computing *idf* scores can be done during the same first pass. Therefore computing the vector lengths is also O(*m n*). Completing the process takes O(*m n*), which is also the complexity of just reading in the collection of documents.

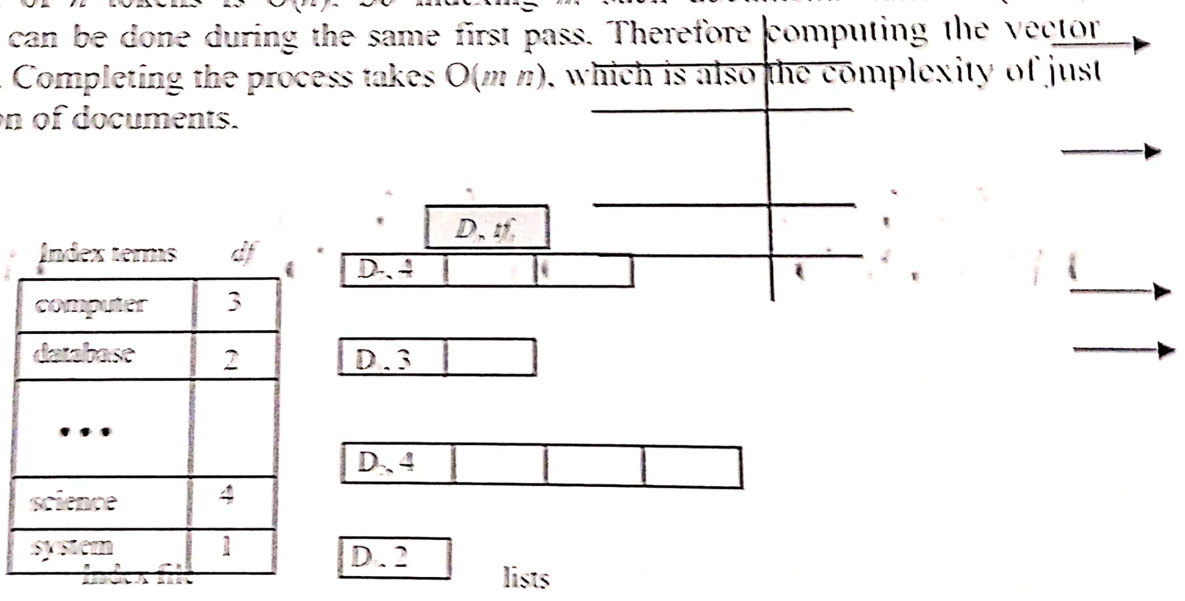


Figure 3 Example of inverted index: for each term, *df* is the number of documents in which it occurred; each list element records the document where the term occurred and how many times.

The last step is the retrieval process. The inverted index from Step 2 is used to find the limited set of documents that contain at least one of the query words. Then the cosine similarity of those documents to the query is computed. The retrieved documents are presented to the user in reversed order of their similarity to the query.

The main advantages of the vector space model are: it is simple and based on clear mathematical theory; it considers both local (*tf*) and global (*idf*) word occurrence frequencies; it provides partial matching and ranked results; it tends to work quite well in practice and allows efficient implementation for large document collections.

The main weaknesses are: it does not account for semantic information (e.g. word sense) and syntactic information (e.g. phrase structure, word order, proximity information); it lacks the control of a Boolean model (e.g., *requiring* a term to appear in a document; for example, given a two-term query "A B", it may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently).

Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an extension of the vector space retrieval method (Deerwester *et al.*, 1990). LSI can retrieve relevant documents even when they do not share any words with the query. Keywords are replaced by concepts ("latent" concepts, not explicit ones). Therefore if only a synonym of the keyword is present in a document, the document will be still found relevant. The idea behind LSI is to transform the matrix of documents by terms in a more concentrated matrix by reducing the dimension of the vector space. The number of dimensions becomes much lower, there is no longer a dimension for each term, but rather a dimension for each "latent" concept or group of synonyms (though it is not clear what is the desired number of concepts). The dimensionality of the matrix is reduced by a mathematical process called singular value decomposition. For more details see, for example, http://lsi.research.telcordia.com/lsi/LSIpapers.html

The advantage of LSI is its strong formal framework that can be applied for text collections in any language and the capacity to retrieve many relevant documents. But the calculation of LSI is expensive, so in practice it works only for relatively small text collections. The main disadvantage of LSI is that it does not allow for fast retrieval, an inverted index cannot be used since now we cannot locate documents by index keywords.

The Probabilistic Model

The probabilistic framework, initially proposed by Robertson and Sparck-Jones (1976), is based on the following idea. Given a user query, there is a set of documents which contain exactly the relevant documents and no other documents, called the ideal answer set. The query is a process for specifying the properties of the answer set, but we don't know what these properties are. Therefore an effort has to be made to guess a description of the answer set and retrieve an initial set of documents. Then the user inspects the top retrieved documents, looking for the relevant ones. The IR system uses this information to refine the description of the ideal answer set. By repeating this process, it is expected that the description of the ideal answer set will improve.

The description of ideal answer set is modeled in probabilistic terms. Given a user query q and a document d_i, the probabilistic model tries to estimate the probability that the user will find the document d_i relevant. The model assumes that this probability of relevance depends only on the query and the document representations. The ideal answer set is referred to as R and should maximize the probability of relevance. Documents in the set R are predicted to be relevant.

The probabilistic ranking is computed as:

$$sim(d_j,q) = P(R \mid d_j) / P(\neg R \mid d_j)$$

This is the ratio of the probability that the document d_j is relevant and the probability that it is not relevant. It reflects the odds of the document d_j being relevant, and minimizes the probability of an erroneous judgment. Using Bayes rule (for two events A and B, the probability of A given B is P(A|B) = P(B|A) P(A) / P(B)) we expand the formula:

$$sim(d_j,q) = \frac{P(d_j \mid R) \cdot P(R)}{P(d_j \mid \neg R) \cdot P(\neg R)} \cong \frac{P(d_j \mid R)}{P(d_j \mid \neg R)}$$

$P(d_j \mid R)$ is the probability of randomly selecting the document d_j from the set R of relevant documents. $P(R)$ stands for the probability that a document randomly selected from the document collection is relevant. The meanings attached to $P(d_j \mid \neg R)$ and $P(\neg R)$ are analogous and complementary. $R(R)$ and $P(\neg R)$ are the same for all the documents relative to the query.

We replace the probability of each document by the product of the probabilities of the terms it contains. We assume the terms occur in a document independent of each other; this is a simplifying assumption that works well in practice, even if in reality terms are not independent, the presence of a term might trigger the presence of a closely related term. We obtain:

$$sim(d_j,q) \cong \frac{\prod_i P(k_i \mid R) \cdot \prod_i P(k_i \mid \neg R)}{\prod_i P(\neg k_i \mid R) \cdot \prod_i P(\neg k_i \mid \neg R)}$$

where $P(k_i \mid R)$ is probability that the index term k_i is present in a document randomly selected from the set R of relevant documents and $P(\neg k_i \mid R)$ is the probability that k_i is not present . The probabilities for $\neg R$ have analogous meanings. Taking logarithms and ignoring factors that are constant for all the documents in the context of the same query we obtain:

$$sim(d_j,q) \cong \sum_i w_{iq} \, w_{ij} \left(\log \frac{P(k_i \mid R)}{P(\neg k_i \mid R)} + \log \frac{P(k_i \mid \neg R)}{P(\neg k_i \mid \neg R)} \right)$$

Where w are binary weights, 1 if the index term is in the document or in the query, 0 if not.

$P(\neg k_i \mid R) = 1 - P(k_i \mid R)$ and $P(\neg k_i \mid \neg R) = 1 - P(k_i \mid \neg R)$.

The probabilities left to estimate are: $P(k_i \mid R)$ and $P(k_i \mid \neg R)$. They can have initial guesses: $P(k_i \mid R) = 0.5$ and $P(k_i \mid \neg R) = df_i / N$, where df_i is the number of documents that contain k_i. This initial guess is used to retrieve an initial set of document V, from which the subset V_i contains the index term k_i. The estimates are re-evaluated:

$P(k_i \mid R) = V_i / V$ and $P(k_i \mid \neg R) = (df_i - V_i) / (N - V)$

This process can be repeated recursively. By doing so, the guess of the probabilities can be improved without the need of the user intervention (contrary to what we mentioned above).

The last formulas pose problems for small values of V and V_i (such as V=1 and V_i= 0). To circumvent these problems, an adjustment factor is added, for example:

$P(k_i \mid R) = (V_i+0.5) / (V+1)$ and $P(k_i \mid \neg R) = (df_i - V_i+0.5) / (N - V+1)$

A popular variant of the probabilistic model is the Okapi formula (Robertson *et. al*, 2000).

Relevance Feedback

The users tend to ask short queries, even when the information need is complex. Irrelevant documents are retrieved as answers because on the ambiguity of the natural language (words have multiple senses). If we know that some of retrieved documents were relevant to the query, terms from those documents can be added to the query in order to be able to retrieve more relevant documents. This is called *relevance feedback*. Often, it is not possible to ask the user to judge the relevance of the retrieved documents. In this case *pseudo-relevance feedback* methods can be used. They assume the first few retrieved documents are relevant and use the most important terms from them to expand the query.

EVALUATION OF INFORMATION RETRIEVAL SYSTEMS

To compare the performance of information retrieval systems there is a need for standard test collections and benchmarks. The TREC forum (Text Retrieval Conference, http://trec.nist.gov/) provides test collections and organizes competition between IR systems every year, since 1992. In order to compute evaluation scores, we need to know the expected solution. Relevance judgments are produced by human judges and included in the standard test collections. CLEF (Cross-Language Evaluation Forum) is another evaluation forum that organizes competition between IR systems that allow queries or documents in multiple languages (http://www.clef-campaign.org/), since the year 2000.

In order to evaluate the performance of an IR system we need to measure how far down the ranked list of results will a user need to look to find some or all the relevant documents.
The typical evaluation process starts with finding a collection of documents. A set of queries needs to be formulated. Then one or more human experts are needed to exhaustively label the relevant documents for each query. This assumes binary relevance judgments: a document is relevant or not to a query. This is simplification, because the relevancy is continuous: a document can be relevant to a certain degree. Even if relevancy is binary, it can be a difficult judgment to make. Relevancy, from a human standpoint, is subjective because it depends on a specific user's judgment; it is situational, it relates to the user's current needs; it depends on uman perception and behavior; and it might be dynamic, it can change over time.

Once the test suite is assembled, we can compute numerical evaluation measures, for ach query, and then average over all the queries in the test set.

Precision and Recall

recision (P) measures the ability to retrieve top-ranked documents that are mostly relevant. ecall (R) measures the ability of the search to find all of the relevant items in the corpus.

$$P = \frac{\textit{Number of relevant documents retrieved}}{\textit{Total number of documents retrieved}}$$

$$R = \frac{\textit{Number of relevant documents retrieved}}{\textit{Total number of relevant documents}}$$

F-measure and E-measure

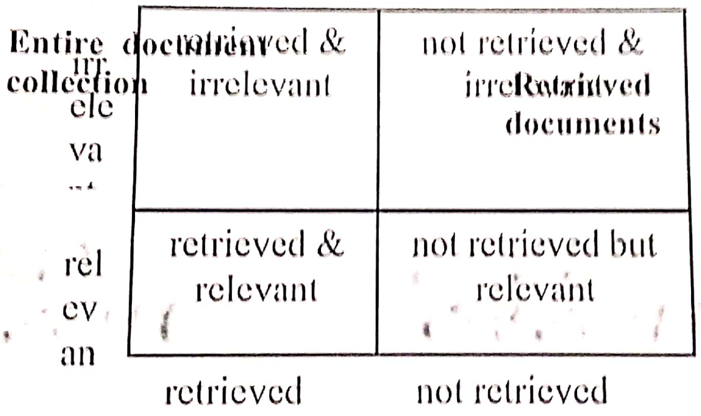
The F-measure combines precision and recall, taking their harmonic mean. The F-measure is high when both precision and recall are high.

$$F = \frac{2PR}{P+R} = \frac{2}{\frac{1}{R}+\frac{1}{P}}$$

A generalization of the F-measure is the E-measure, which allows emphasis on precision over recall or vice-versa. The value of the parameter β controls this trade-off: if β = 1 precision and recall are weighted equally (E=F), if β < 1 precision weights more, and if β > 1 recall weights more,

$$E = \frac{(1+\beta^2)PR}{\beta^2 P + R} = \frac{(1+\beta^2)}{\frac{\beta^2}{R}+\frac{1}{P}}$$

Figure 4 shows the distribution of the retrieved documents versus the relevant documents. n the upper part of the figure, the intersection of the two circles is the part that needs to be naximized by an IR system. In the lower part of the figure, the number of documents that need o be maximized is in the lower left corner and the upper right corner. The other two corners vould contain zeros for an ideal IR system.



Entire document collection		
irrelevant	retrieved & irrelevant	not retrieved & irrelevant Retrieved documents
relevant	retrieved & relevant	not retrieved but relevant
	retrieved	not retrieved

Figure 4. Retrieved versus relevant documents.

Sometimes, for very large text collections or the Web, it is difficult to find the total ber of relevant items. In such cases, one solution is to sample across the collection and to orm relevance judgment on the sampled items. Another idea is to apply different retrieval ithms or different IR systems to the same collection for the same query, then aggregate the ant items from all of them and perform relevance judgments on this set.

Mean Average Precision

Usually precision is more important than recall in IR systems, if the user is looking for an answer to a query, not for all the possible answers. Recall can be important when a user needs to know all the relevant information on a topic. A system can increase precision by decreasing recall and vice-versa; there is a precision-recall tradeoff (for example, recall can be increased by simply retrieving more documents, but the precision will go down, since many retrieved documents will not be relevant). Precision-recall curves can be used to compare two IR systems for all values of precision and recall.

In fact precision and recall are not enough for evaluating IR systems. For example, if we have two systems that retrieve 10 documents each, 5 relevant and 5 not relevant, both have precision 0.5, but a system that has the first 5 retrieved documents relevant and the next 5 irrelevant is much better than a system that has the first 5 retrieved documents irrelevant and the next 5 relevant (because the user will be annoyed to have to check the irrelevant documents first). Modified measures that combine precision and recall and consider the order of the retrieved documents are needed.

Some good measures are: precision at 5 retrieved documents, precision at 10 retrieved documents or some other cut-off point; the R-Precision; the interpolated average precision; and the mean average precision. The trec_eval script can be used to compute many evaluation measures (http://trec.nist.gov/trec_eval/).

The *R-precision* is the precision at the R-th position in the ranking of the results for a query that has R known relevant documents.

The *interpolated average precision* computes precision at fixed recall intervals (11 points), to allow fair average over all the queries in the test set at the same recall levels. See Chapter 3 of (Baeza-Yates and Berthier Ribeiro-Neto, 1999) for more details. This measure is not much used use lately in evaluating IR systems.

The most widely-used measure is the *mean average precision* (MAP score). It computes precision at each point in the ranking where a relevant document was found, divided by the number of existing relevant documents (and then averages over all queries). Here is a simple example of computing this measure.

Given a query q, for which the relevant documents are d1, d6, d10, d15, d22, d26, an IR system retrieves the following ranking: d6, d2, d11, d3, d10, d1, d14, d15, d7, d23. We compute the precision and recall for this ranking at each retrieved document.

Rank	Document	Recall	Precision
1	**d6**	1/6 = 0.166	1/1 = 1.00
2	d2	1/6 = 0.166	1/2 = 0.50
3	d11	1/6 = 0.166	1/3 = 0.33
4	d3	1/6 = 0.166	1/4 = 0.25
5	**d10**	2/6 = 0.33	2/5 = 0.40
6	**d1**	3/6 = 0.50	3/6 = 0.50
7	d14	3/6 = 0.50	3/7 = 0.42
8	**d15**	4/6 = 0.66	4/8 = 0.50
9	d7	4/6 = 0.66	4/9 = 0.44
10	d23	4/6 = 0.66	4/10 = 0.40

In this table, the relevant documents are marked in bold in the ranked list of retrieved documents (the second column). At each position in the list, recall is computed as how many relevant documents were retrieved until this point out of the 6 known to be relevant, and precision is how many correct documents are among these retrieved documents up till this point. At position 1, one correct document is retrieved out of 6 (recall is 1/6) and 1 document is correct (precision is 1/1). At position 2, 1 out of 6 relevant documents is retrieved (recall is 1/6) and 1 out of 2 is correct (precision is 1/2). At position 5 one more relevant document is found; recall becomes 2 out of 6 and precision 2 out of 5. The average precision over positions 1, 5, 6, and 8 where relevant documents were found is (1.0+0.40+0.50+0.50)/6=0.40. The R-precision is the precision at position 6, which is 3/6=0.50.

Novelty Ratio and Coverage Ratio

Other aspects of the retrieved results could be evaluated. The *novelty ratio* is the proportion of documents retrieved and judged relevant by the user and of which the user was previously unaware; it measures the ability to find *new* information on a topic. The *coverage ratio* is the proportion of relevant items retrieved out of the total relevant documents *known* to a user prior to the search. It is useful when the user wants to locate documents which they have seen before. The *user effort* measures the amount of work required from the user in formulating queries, conducting the search, and screening the output. The *response time* is the time interval between the reception of a user query and the presentation of system responses. The *form of presentation* is the influence of the search output format on the user's ability to utilize the retrieved materials. The *collection coverage* measures the extent to which any/all relevant items are included in the document collection.

PRACTICAL CONSIDERATIONS

Efficient Indexing

Several implementation issues were addressed in the section about the vector space retrieval, including how to build an inverted index for fast retrieval. Figure 3 presented an example of inverted index. This idea can be also used in the implementations of the Boolean model and of the probabilistic model. In order to allow searching for exact phrases (two or more consecutive words) the positions of terms in documents can also be stored in the inverted index. Other information can be stored in the index as well (for example if the word was found in a title, heading, bold font, etc.)

THE INFORMATION RETRIEVAL SYSTEM

Figure 2 presents a more detailed view of the architecture of an IR system (Baeza-Yates and Berthier Ribeiro-Neto, 1999). Text Operations are used to preprocess the documents collections and to extract index words. The indexing module constructs an inverted index from words to document pointers. The searching module retrieves documents that contain given query words, using the inverted index. The ranking module scores all the retrieved documents according to a relevance metric. The user interface manages interaction with the user: the input of the query and the output of the ranked documents, including the visualization of results. The query operations can transform the query in order to improve retrieval (query expansion using synonyms from a

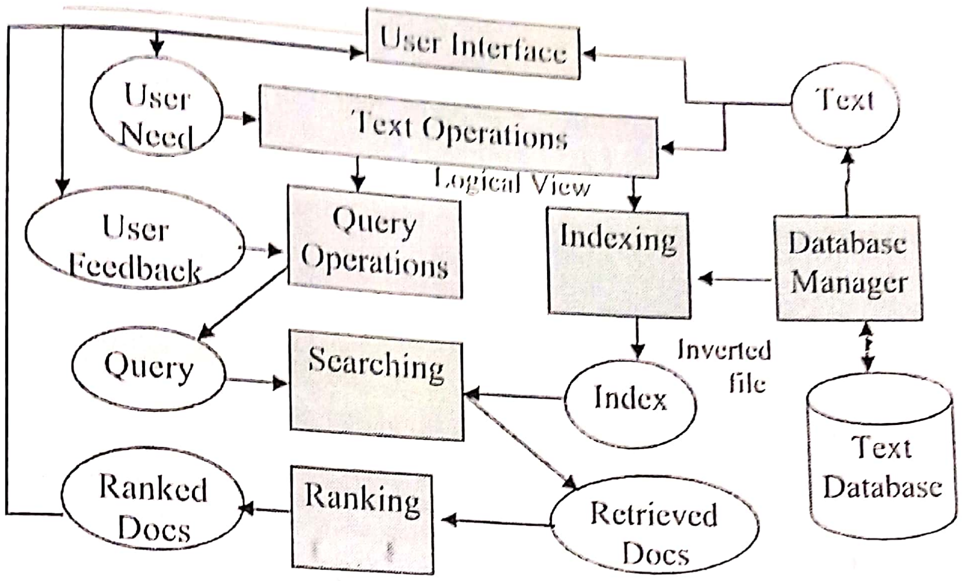


Figure 2. The architecture of an IR system: Text operations are applied of the text of the documents and on the description of the user information need in order to transform them in a simplified form needed for computation. The documents are indexed and the index is used to execute the search. After ranked documents are retrieved, the user can provide feedback which can be used to refine the query and restart the search for improved results.

Preprocessing the document collection

There are several preprocessing steps needed to prepare the document collection for the IR task. The first step is to filter out unwanted characters and markup (e.g. HTML tags, punctuation, numbers, etc.). Then the text needs to be broken into tokens (keywords) by using as delimiters white space and punctuation characters. This it not quite as straightforward as it seems, since words in texts are not always clearly delimited (for example, if the text is *You can't do this*, you can consider the apostrophe as a word separator to get two words *can* and *t*, or ignore it as separator and consider one word *can't*, or expand the contacted form into two words *can* and *not* and use the white space as separator).

The keywords can be used as they are, or they can be transformed into a base form, for example nouns in the singular form, verbs in the infinitive form, etc. (e.g., *books* becomes *book*, *talked* becomes *talk*). A common approach is to stem the tokens to "stem" forms. For example, *computational* becomes *comput* and *computing* becomes *comput*. Stemming the terms before building the inverted index has the advantage that it reduces the size of the index, and allows for retrieval of webpages with various inflected forms of a word (for example, when searching for webpages with the word *computation*, the results will include webpages with *computations* and *computing*). Stemming is easier to do than computing base forms, because stemmers remove suffixes, without needing a full dictionary of words in a language. A popular and fast stemmer is Porter's stemmer.

Another useful preprocessing step is to remove very frequent words that appear in most of the documents and do not bear any meaningful content. They are called stopwords (e.g., *a*, *the*, *it*, *of*, *could*, etc.). An example of stopwords list can be found at: http://www.lextek.com/manuals/onix/stopwords1.html.

Important phrases composed of two or more words could also be detected to be used as keywords (possibly using a domain specific dictionary, or using a statistical method for analyzing the text collection in order to detect sequences of words that appear together very often).

Now the text is ready for the next step, building the inverted index that stores for each keyword a list of documents that contain it, in order to allow for fast access during the retrieval step.

Information Retrieval Models

This section presents information retrieval models that can be applied on any text collection. Not all the IR models are easily scaled up to be able to deal with a very large collection, such as pages collected from the Web. The most important IR models are: the Boolean Model, the Vector Space Model, and the Probabilistic Model. Various extensions of these models are possible. We discuss one of them here, Latent Semantic Indexing, which is an extension of the Vector Space Model.

The Boolean Model

The Boolean model is the simplest to implement. A document is represented as a set of keywords. Queries are Boolean expressions of keywords, connected by AND, OR, and NOT, including the use of brackets to indicate the scope of these operators. For example, the query "all the hotels in Rio Brazil or Hilo Hawaii, but not Hilton" is typed by the user as:

[[Rio & Brazil] | [Hilo & Hawaii]] & hotel & !Hilton]

The output of the system is a list of documents that are relevant, but there will be no partial matches or ranking. The Boolean model is very rigid: AND means "all"; OR means "any". All matched documents will be returned, making it difficult to control the number of documents retrieved. All the matched documents satisfy the query to the same degree; that makes it difficult to rank the output. Another disadvantage of this model is that is it not easy for the users to express complex queries.

The Vector Space Model

The vector space model of information retrieval is a very successful statistical method proposed by Salton (1989). It generates weighted term vectors for each document in the collection, and for the user query. Then the retrieval is based on the *similarity* between the query vector and document vectors. The output documents are ranked according to this similarity. The similarity is based on the occurrence *frequencies* of the keywords in the query and in the documents.

Let's assume that t distinct terms remain after preprocessing; let's call them index terms or the vocabulary. These terms form a vector space with dimensionality t, the size of the vocabulary. Each term i, in a document j, is given a weight w_{ij}. Both the documents and the queries are expressed as t-dimensional vectors: $d_j = (w_{1j}, w_{2j}, ..., w_{tj})$.

A collection of N documents can be represented in the vector space model by a documents-by-terms matrix. An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document; it simply doesn't appear in the document. The matrix tends to contain lots of zeros.

$$\begin{array}{lcccc} & T_1 & T_2 & \dots & T_t \\ d_1 & w_{11} & w_{21} & \dots & w_{t1} \\ d_2 & w_{12} & w_{22} & \dots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ d_N & w_{1N} & w_{2N} & \dots & w_{tN} \end{array}$$

The weights in the matrix can be 1 if the term occurs in the document and 0 if it does not (binary weights); but the more frequent terms in a document are more important, i.e.. more indicative of the topic. Therefore it is good to use the frequencies of the terms as weights.
Let f_{ij} be the frequency of the term T_i in the document d_j
We can normalize the *term frequency* (*tf*) across the entire corpus: $tf_{ij} = f_{ij} / max\{f_{ij}\}$.Terms that appear in many *different* documents are *less* indicative of overall topic.
Let df_i be the document frequency of term T_i – the number of documents containing the term i, and let idf_i be the inverse document frequency of term T_i:

$$idf_i = log\ (N/df_i)$$

(where N is the total number of documents). The *idf* value is an indication of a term's *discrimination* power. The logarithm is used to dampen the effect relative to *tf*. A typical combined term importance indicator is *tf-idf* weighting:

$$w_{ij} = tf_{ij} \cdot idf_i = tf_{ij}\ log\ (N/df_i).$$

A term occurring frequently in the document but rarely in the rest of the collection is given high weight. Many other ways of determining term weights have been proposed. Experimentally, *tf-idf* has been found to work well.

The query is also transformed into a vector. It is typically treated as a document and also *tf-idf* weighted. Alternatively, the user could supply weights for the given query terms.

The similarity between vectors for the document d_j and the query q can be computed as the vector inner product:

$$\mathrm{sim}(\vec{d_j},\vec{q}) = \vec{d_j} \bullet \vec{q} = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

where w_{ij} is the weight of term i in document j and w_{iq} is the weight of term i in the query

For binary vectors, the inner product is the number of matched query terms in th document (the size of the intersection). For weighted term vectors, it is the sum of the produc of the weights of the matched terms. There are several problems with the inner product: it do not have a bounded range of values; it favors long documents with a large number of uniq terms; it measures how many terms are matched but not how many terms are *not* matched.

The cosine similarity measure tends to work better. The formula is the same as the in product, but it is normalized by the length of the documents and the length of the query (length of a vector is the square root of the sum of the squares of its components).

$$\mathrm{cosSim}(\vec{d_j},\vec{q}) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

The cosine measures the angles between the two vectors (the higher the cosine value closer to 1, the smaller the angle between the vector of the document and the vector of the query, therefore a more relevant document). Because we consider only the angle, the length of the documents is not a problem anymore.

A naïve implementation of the vector space retrieval is straightforward but impractical: convert all the documents in collection C to *tf-idf* weighted vectors, for all the keywords in the vocabulary V; convert the query to a *tf-idf*-weighted vector q; then for each document d_i in C compute cosSim(d_i, q); sort the documents by decreasing score and present top-ranked documents to the user. The time complexity would be $O(|V|\cdot|C|)$. It would take very long for large V and C (for example, if $|V| = 10{,}000$ and $|C| = 100{,}000$ then $|V|\cdot|C| = 1{,}000{,}000{,}000$).

A practical implementation is based on the observation that documents containing none of the query words do not affect the final ranking. Identifying those documents that contain at least one query word is easily done by using an inverted index. The numerator in the cosine similarity formula will be calculated much faster because the multiplications where one of the terms is zero will not be executed.

The steps of a practical implementation are as follows. Step 1, pre-processing (tokenization, stopword removal, stemming), determines the keywords in the vocabulary to be used as index terms. Step 2 is the building of the inverted index, with an entry for each keyword in the vocabulary (see Figure 3). The index is a data structure that will allow fast access in the retrieval step (hash table, B-tree, sparse list, etc.) For each keyword, the index keeps a list of all the documents where it appears together with the corresponding term frequency (*tf*). It also keeps the total number of occurrences in all documents (for the *idf* score). So the *tf-idf* scores can be computed in one pass trough the collection. The cosine similarity also requires document lengths; a second pass to is needed to compute document vector lengths. The time complexity of indexing a document of n tokens is $O(n)$. So indexing m such documents takes $O(m\ n)$. Computing *idf* scores can be done during the same first pass. Therefore computing the vector lengths is also $O(m\ n)$. Completing the process takes $O(m\ n)$, which is also the complexity of just reading in the collection of documents.

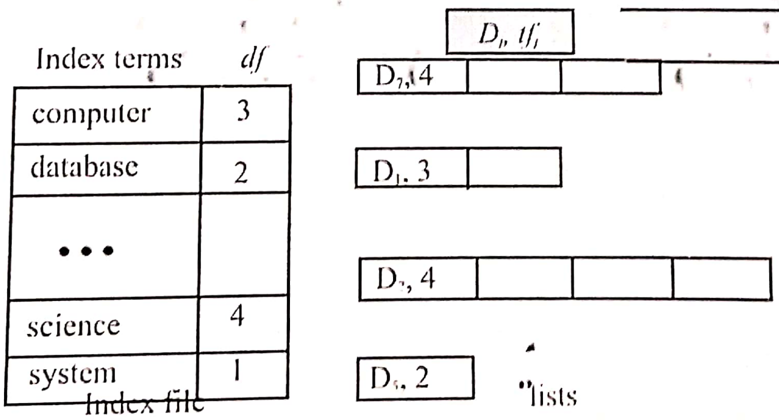


Figure 3 Example of inverted index: for each term, *df* is the number of documents in which it occurred; each list element records the document where the term occurred and how many times.

The last step is the retrieval process. The inverted index from Step 2 is used to find the limited set of documents that contain at least one of the query words. Then the cosine similarity of those documents to the query is computed. The retrieved documents are presented to the user in reversed order of their similarity to the query.

The main advantages of the vector space model are: it is simple and based on clear mathematical theory; it considers both local (*tf*) and global (*idf*) word occurrence frequencies; it provides partial matching and ranked results; it tends to work quite well in practice and allows efficient implementation for large document collections.

The main weaknesses are: it does not account for semantic information (e.g. word sense) and syntactic information (e.g. phrase structure, word order, proximity information); it lacks the control of a Boolean model (e.g., *requiring* a term to appear in a document; for example, given a two-term query "A B", it may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently).

Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an extension of the vector space retrieval method (Deerwester *et al.*, 1990). LSI can retrieve relevant documents even when they do not share any words with the query. Keywords are replaced by concepts ("latent" concepts, not explicit ones). Therefore if only a synonym of the keyword is present in a document, the document will be still found relevant. The idea behind LSI is to transform the matrix of documents by terms in a more concentrated matrix by reducing the dimension of the vector space. The number of dimensions becomes much lower, there is no longer a dimension for each term, but rather a dimension for each "latent" concept or group of synonyms (though it is not clear what is the desired number of concepts). The dimensionality of the matrix is reduced by a mathematical process called singular value decomposition. For more details see, for example,
http://lsi.research.telcordia.com/lsi/LSIpapers.html

The advantage of LSI is its strong formal framework that can be applied for text collections in any language and the capacity to retrieve many relevant documents. But the calculation of LSI is expensive, so in practice it works only for relatively small text collections. The main disadvantage of LSI is that it does not allow for fast retrieval, an inverted index cannot be used since now we cannot locate documents by index keywords.

The Probabilistic Model

The probabilistic framework, initially proposed by Robertson and Sparck-Jones (1976), is based on the following idea. Given a user query, there is a set of documents which contain exactly the relevant documents and no other documents, called the ideal answer set. The query is a process for specifying the properties of the answer set, but we don't know what these properties are. Therefore an effort has to be made to guess a description of the answer set and retrieve an initial set of documents. Then the user inspects the top retrieved documents, looking for the relevant ones. The IR system uses this information to refine the description of the ideal answer set. By repeating this process, it is expected that the description of the ideal answer set will improve.

The description of ideal answer set is modeled in probabilistic terms. Given a user query q and a document d_i, the probabilistic model tries to estimate the probability that the user will find the document d_i relevant. The model assumes that this probability of relevance depends only on the query and the document representations. The ideal answer set is referred to as R and should maximize the probability of relevance. Documents in the set R are predicted to be relevant.

The probabilistic ranking is computed as:

$$sim(d_j,q) = P(R \mid d_j) \,/\, P(\neg R \mid d_j)$$

This is the ratio of the probability that the document d_j is relevant and the probability that it is not relevant. It reflects the odds of the document d_j being relevant, and minimizes the probability of an erroneous judgment. Using Bayes rule (for two events A and B, the probability of A given B is P(A|B) = P(B|A) P(A) / P(B)) we expand the formula:

$$sim(d_j,q) = \frac{P(d_j \mid R) \cdot P(R)}{P(d_j \mid \neg R) \cdot P(\neg R)} \cong \frac{P(d_j \mid R)}{P(d_j \mid \neg R)}$$

$P(d_j \mid R)$ is the probability of randomly selecting the document d_j from the set R of relevant documents. $P(R)$ stands for the probability that a document randomly selected from the document collection is relevant. The meanings attached to $P(d_j \mid \neg R)$ and $P(\neg R)$ are analogous and complementary. $P(R)$ and $P(\neg R)$ are the same for all the documents relative to the query.

We replace the probability of each document by the product of the probabilities of the terms it contains. We assume the terms occur in a document independent of each other; this is a simplifying assumption that works well in practice, even if in reality terms are not independent, the presence of a term might trigger the presence of a closely related term. We obtain:

$$sim(d_j,q) \cong \frac{\prod_i P(k_i \mid R) \cdot \prod_i P(k_i \mid \neg R)}{\prod_i P(\neg k_i \mid R) \cdot \prod_i P(\neg k_i \mid \neg R)}$$

where $P(k_i \mid R)$ is probability that the index term k_i is present in a document randomly selected from the set R of relevant documents and $P(\neg k_i \mid R)$ is the probability that k_i is not present . The probabilities for $\neg R$ have analogous meanings. Taking logarithms and ignoring factors that are constant for all the documents in the context of the same query we obtain:

$$sim(d_j,q) \cong \sum_i w_{iq}\, w_{ij} \left(\log \frac{P(k_i \mid R)}{P(\neg k_i \mid R)} + \log \frac{P(k_i \mid \neg R)}{P(\neg k_i \mid \neg R)}\right)$$

Where w are binary weights, 1 if the index term is in the document or in the query, 0 if not.

$P(\neg k_i \mid R) = 1 - P(k_i \mid R)$ and $P(\neg k_i \mid \neg R) = 1 - P(k_i \mid \neg R)$.

The probabilities left to estimate are: $P(k_i \mid R)$ and $P(k_i \mid \neg R)$. They can have initial guesses: $P(k_i \mid R) = 0.5$ and $P(k_i \mid \neg R) = df_i / N$, where df_i is the number of documents that contain k_i. This initial guess is used to retrieve an initial set of document V, from which the subset V_i contains the index term k_i. The estimates are re-evaluated:

$P(k_i \mid R) = V_i / V$ and $P(k_i \mid \neg R) = (df_i - V_i) / (N - V)$

This process can be repeated recursively. By doing so, the guess of the probabilities can be improved without the need of the user intervention (contrary to what we mentioned above).

The last formulas pose problems for small values of V and V_i (such as V=1 and V_i= 0). To circumvent these problems, an adjustment factor is added, for example:

$P(k_i \mid R) = (V_i+0.5) / (V+1)$ and $P(k_i \mid \neg R) = (df_i - V_i+0.5) / (N - V+1)$

A popular variant of the probabilistic model is the Okapi formula (Robertson *et. al*, 2000).

Relevance Feedback

The users tend to ask short queries, even when the information need is complex. Irrelevant documents are retrieved as answers because on the ambiguity of the natural language (words have multiple senses). If we know that some of retrieved documents were relevant to the query, terms from those documents can be added to the query in order to be able to retrieve more relevant documents. This is called *relevance feedback*. Often, it is not possible to ask the user to judge the relevance of the retrieved documents. In this case *pseudo-relevance feedback* methods can be used. They assume the first few retrieved documents are relevant and use the most important terms from them to expand the query.

EVALUATION OF INFORMATION RETRIEVAL SYSTEMS

To compare the performance of information retrieval systems there is a need for standard test collections and benchmarks. The TREC forum (Text Retrieval Conference, http://trec.nist.gov/) provides test collections and organizes competition between IR systems every year, since 1992. In order to compute evaluation scores, we need to know the expected solution. Relevance judgments are produced by human judges and included in the standard test collections. CLEF (Cross-Language Evaluation Forum) is another evaluation forum that organizes competition between IR systems that allow queries or documents in multiple languages (http://www.clef-campaign.org/), since the year 2000.

In order to evaluate the performance of an IR system we need to measure how far down he ranked list of results will a user need to look to find some or all the relevant documents. he typical evaluation process starts with finding a collection of documents. A set of queries eeds to be formulated. Then one or more human experts are needed to exhaustively label the elevant documents for each query. This assumes binary relevance judgments: a document is elevant or not to a query. This is simplification, because the relevancy is continuous: a ocument can be relevant to a certain degree. Even if relevancy is binary, it can be a difficult dgment to make. Relevancy, from a human standpoint, is subjective because it depends on a ecific user's judgment; it is situational, it relates to the user's current needs; it depends on uman perception and behavior; and it might be dynamic, it can change over time.

Once the test suite is assembled, we can compute numerical evaluation measures, for ch query, and then average over all the queries in the test set.

ecision and Recall

ecision (P) measures the ability to retrieve top-ranked documents that are mostly relevant. call (R) measures the ability of the search to find all of the relevant items in the corpus.

$$P = \frac{\textit{Number of relevant documents retrieved}}{\textit{Total number of documents retrieved}}$$

trieved

F-measure and E-measure

The F-measure combines precision and recall, taking their harmonic mean. The F-measure is high when both precision and recall are high.

$$F = \frac{2PR}{P+R} = \frac{2}{\frac{1}{R}+\frac{1}{P}}$$

A generalization of the F-measure is the E-measure, which allows emphasis on precision over recall or vice-versa. The value of the parameter β controls this trade-off: if β = 1 precision and recall are weighted equally (E=F), if β < 1 precision weights more, and if β > 1 recall weights more.

$$E = \frac{(1+\beta^2)PR}{\beta^2 P + R} = \frac{(1+\beta^2)}{\frac{\beta^2}{R}+\frac{1}{P}}$$

Figure 4 shows the distribution of the retrieved documents versus the relevant documents. In the upper part of the figure, the intersection of the two circles is the part that needs to be maximized by an IR system. In the lower part of the figure, the number of documents that need to be maximized is in the lower left corner and the upper right corner. The other two corners would contain zeros for an ideal IR system.

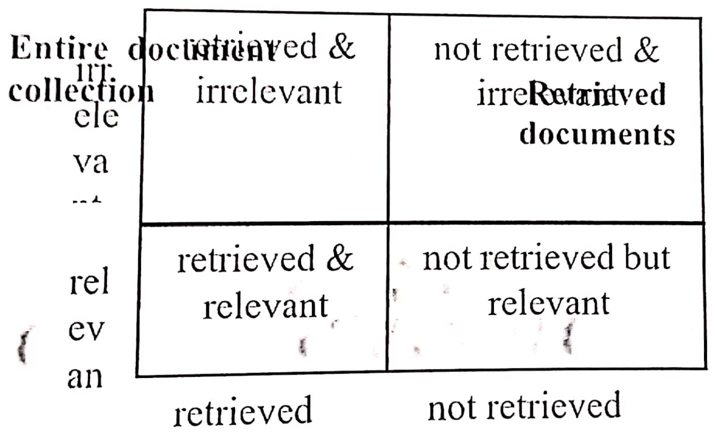


Figure 4. Retrieved versus relevant documents.

Sometimes, for very large text collections or the Web, it is difficult to find the total umber of relevant items. In such cases, one solution is to sample across the collection and to erform relevance judgment on the sampled items. Another idea is to apply different retrieval gorithms or different IR systems to the same collection for the same query, then aggregate the levant items from all of them and perform relevance judgments on this set.

Mean

Usually precision is more important than recall in IR systems, if the user is looking for an answer to a query, not for all the possible answers. Recall can be important when a user needs to know all the relevant information on a topic. A system can increase precision by decreasing recall and vice-versa; there is a precision-recall tradeoff (for example, recall can be increased by simply retrieving more documents, but the precision will go down, since many retrieved documents will not be relevant). Precision-recall curves can be used to compare two IR systems for all values of precision and recall.

In fact precision and recall are not enough for evaluating IR systems. For example, if we have two systems that retrieve 10 documents each, 5 relevant and 5 not relevant, both have precision 0.5, but a system that has the first 5 retrieved documents relevant and the next 5 irrelevant is much better than a system that has the first 5 retrieved documents irrelevant and the next 5 relevant (because the user will be annoyed to have to check the irrelevant documents first). Modified measures that combine precision and recall and consider the order of the retrieved documents are needed.

Some good measures are: precision at 5 retrieved documents, precision at 10 retrieved documents or some other cut-off point; the R-Precision; the interpolated average precision; and the mean average precision. The trec_eval script can be used to compute many evaluation measures (http://trec.nist.gov/trec_eval/).

The *R-precision* is the precision at the R-th position in the ranking of the results for a query that has R known relevant documents.

The *interpolated average precision* computes precision at fixed recall intervals (11 points), to allow fair average over all the queries in the test set at the same recall levels. See Chapter 3 of (Baeza-Yates and Berthier Ribeiro-Neto, 1999) for more details. This measure is not much used use lately in evaluating IR systems.

The most widely-used measure is the *mean average precision* (MAP score). It computes precision at each point in the ranking where a relevant document was found, divided by the number of existing relevant documents (and then averages over all queries). Here is a simple example of computing this measure.

Given a query q, for which the relevant documents are d1, d6, d10, d15, d22, d26, an IR system retrieves the following ranking: d6, d2, d11, d3, d10, d1, d14, d15, d7, d23. We compute the precision and recall for this ranking at each retrieved document.

Rank	Document	Recall	Precision
1	**d6**	1/6 = 0.166	1/1 = 1.00
2	d2	1/6 = 0.166	1/2 = 0.50
3	d11	1/6 = 0.166	1/3 = 0.33
4	d3	1/6 = 0.166	1/4 = 0.25
5	**d10**	2/6 = 0.33	2/5 = 0.40
6	**d1**	3/6 = 0.50	3/6 = 0.50
7	d14	3/6 = 0.50	3/7 = 0.42
8	**d15**	4/6 = 0.66	4/8 = 0.50
9	d7	4/6 = 0.66	4/9 = 0.44
10	d23	4/6 = 0.66	4/10 = 0.40

In this table, the relevant documents are marked in bold in the ranked list of retrieved documents (the second column). At each position in the list, recall is computed as how many relevant documents were retrieved until this point out of the 6 known to be relevant, and precision is how many correct documents are among these retrieved documents up till this point. At position 1, one correct document is retrieved out of 6 (recall is 1/6) and 1 document is correct (precision is 1/1). At position 2, 1 out of 6 relevant documents is retrieved (recall is 1/6) and 1 out of 2 is correct (precision is 1/2). At position 5 one more relevant document is found; recall becomes 2 out of 6 and precision 2 out of 5. The average precision over positions 1, 5, 6, and 8 where relevant documents were found is (1.0+0.40+0.50+0.50)/6=0.40. The R-precision is the precision at position 6, which is 3/6=0.50.

xamples of Information Retrieval Systems

here are many prototype systems (SMART, Lucene, Lemur, JFeret, Inquire, etc.); some of them re available for download, for research purposes.

MART

he SMART Information Retrieval system, originally developed at Cornell University in the)60s. SMART is based on the vector space model of information retrieval (Salton, 1989). It nerates weighted term vectors for the document collection. SMART preprocesses the documents by tokenizing the text into words, removing common words that appear on its stop-list, and performing stemming on the remaining words to derive a set of terms. When the IR system executes a user query, the query terms are also converted into weighted term vectors. A vector inner-product similarity computation is then used to rank documents in decreasing order of their similarity to the user query. The newest version of SMART (version 11) offers many state-of-the-art options for weighting the terms in the vectors. Each term-weighting scheme is described as a combination of term frequency, collection frequency, and length normalization components (Salton and Buckley, 1988). The system is implemented in C for Unix operating systems and it is available for download at ftp://ftp.cs.cornell.edu/pub/smart/.

Lucene

Lucene is a Java-based open source toolkit for text indexing and searching. It is easy to use and flexible. In includes a full-featured text search engine library, suitable for application that requires full-text search, especially cross-platform. It is available for download at http://lucene.apache.org/java/docs/.

Lemur

The Lemur Toolkit is designed to facilitate research in language modeling and information retrieval, where IR is broadly interpreted to include such technologies as ad hoc and distributed retrieval, cross-language IR, summarization, filtering, and classification. The toolkit supports indexing of large-scale text databases, the construction of simple probabilistic language models for documents, queries, or subcollections, and the implementation of retrieval systems based on language models as well as a variety of other retrieval models. The system is written in the C and C++ languages, and is designed as a research system to run under Unix operating systems, although it can also run under Windows. The toolkit is being developed as part of the Lemur Project, a collaboration between the Computer Science Department at the University of Massachusetts and the School of Computer Science at Carnegie Mellon University. It is available for download at http://www.lemurproject.org.

Okapi

The Okapi IR system was developed at the Centre for Interactive Systems Research at City University London. The system is based on a formula referring to some half a dozen variables (Robertson *et. al*, 2000), which estimate the probability that a given document is relevant to a given query. The system has a simple interface and several layers of complex software, which support both probabilistic and non-probabilistic retrieval functions, and combinations of them. It is implemented in C/C++ and Tcl/Tk for Linux and Solaris, and it is available for research purposes only, for a nominal fee, at http://www.soi.city.ac.uk/~andym/OKAPI-PACK/.

Example of large-scale search engine architecture: Google

Web search engines have to deal with very large collections of documents. In addition to the problems of typical IR systems, they have to deal with scalability and efficiency issues.

Figure 5 presents the architecture of Google search engine (Brin and Page, 1998). See http://www.google.ca/intl/en/corporate/tech.html for more up-to-date but very generic information. The main components of the architecture accomplish the three functions: crawling, indexing, and searching.

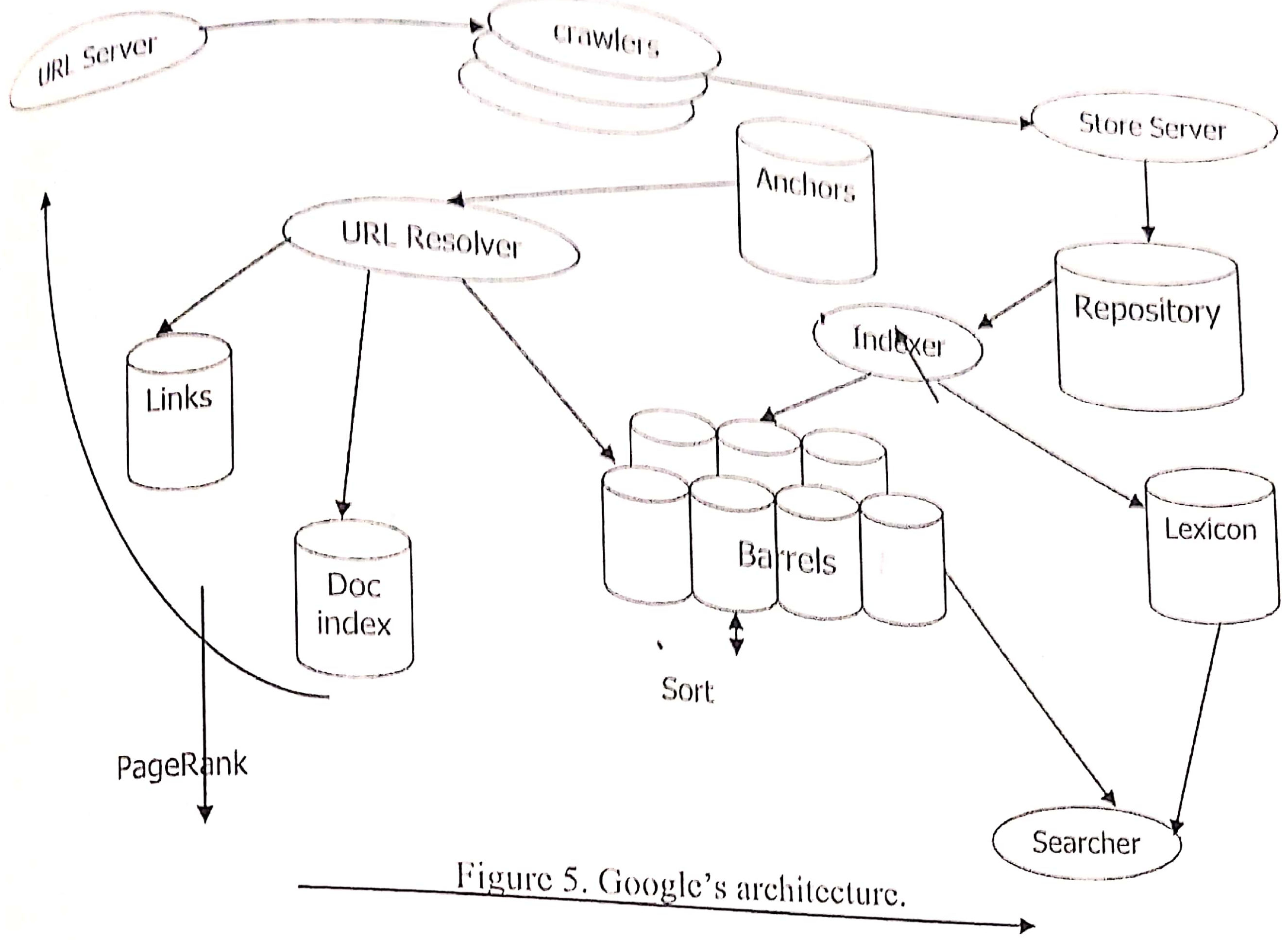


Figure 5. Google's architecture.

In order to deal with many small files in an efficient way (in both space requirements and access time), the system uses big virtual files addressable by 64 bit integers, and it supports compression. The "Repository" contains the full HTML code of every webpage (compressed), a document identifier, its length, and its URL. The "Document Index" keeps information about each document (the document identifier, the current document status, a pointer into the repository, a document checksum, and various statistics). The "Lexicon" is a repository of words, implemented as a list and a hash table of pointers. The list stores occurrences of a particular word in a particular document. It also records the types of hit: Plain (simple text), Fancy (in ~~special~~ HTML format such as bold or heading) and Anchor (text on a link).

There are two indexes: The Forward Index (for fast access to words using word identifiers, and to documents using document identifiers) and ~~the Inverted Index (for the actual~~ retrieval and similarity calculation).

Google associates the text of a link with the page of the link and the page where the link points to. The ~~advantages~~ of doing this are: the anchors often provide accurate descriptions; anchors may exist for documents which cannot be indexed (i.e., images, programs, and databases); propagating anchor text improves the quality of the results.

Page Ranking Algorithms

In addition to how relevant the retrieved webpages are to the user query, they can also be ranked by their importance. A webpage is important, for example, if there are many webpages that have

This section presents the Hubs and Authorities algorithm (Kleinberg, 1999) and gle's PageRank algorithm (Brin & Page, 1998), with examples.

ubs and Authorities

horities are pages that are recognized as providing significant, trustworthy, and useful mation on a topic. The *in-degree* of a page (the number of links that point to the page) is a ple measure of authority. *Hubs* are index pages that provide lots of useful links to relevant ent pages (topic authorities). The algorithm developed by Kleinberg in 1998 attempts to putationally determine hubs and authorities on a particular topic through analysis of a vant subgraph of the Web. It is based on mutually recursive facts: hubs point to lots of orities and authorities are pointed to by lots of hubs. Together they tend to form a bipartite ph, as depicted in Figure 6.

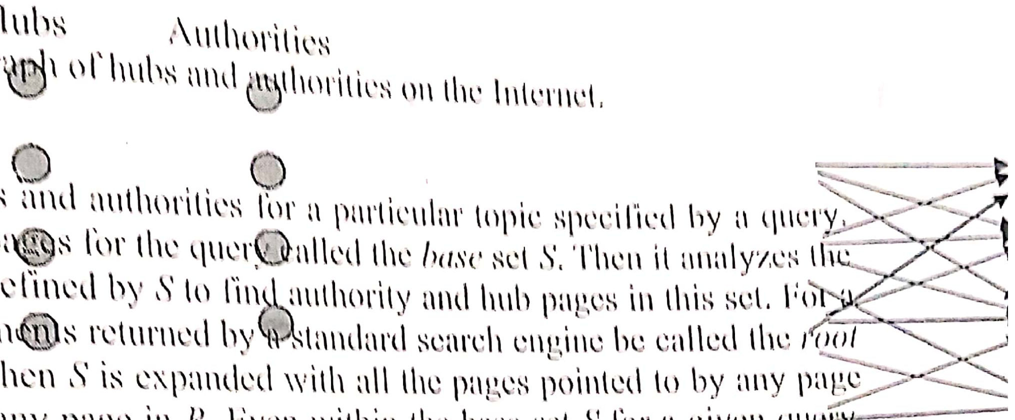


Figure 6. Bipartite graph of hubs and authorities on the Internet.

The algorithm computes hubs and authorities for a particular topic specified by a query. ist, it determines a set of relevant pages for the query called the *base* set *S*. Then it analyzes the k structure of the Web subgraph defined by *S* to find authority and hub pages in this set. For a ecific query *Q*, let the set of documents returned by a standard search engine be called the *root* et *R*. The set *S* is initialized to *R*. Then *S* is expanded with all the pages pointed to by any page n *R* and all the pages that point to any page in *R*. Even within the base set *S* for a given query, he nodes with highest in-degree are not necessarily authorities (they may be generally popular ges like Yahoo or Amazon).

The algorithm slowly converges on a mutually reinforcing set of hubs and authorities. For ach page $p \in S$, an authority score a_p and a hub score h_p are maintained. All a_p and h_p are itialized with 1. The algorithm is repeated several times. At each iteration, the scores are maintained normalized, and the new scores use the values from the previous iteration.

$$\sum_{p\in S}(a_p)^2 = \sum_{p\in S}(h_p)^2 = 1$$

Authorities are pointed to by lots of good hubs (all pages q that point to p):

$$a_p = \sum_{q:q\to p} h_q$$

Hubs point to lots of good authorities (all pages q that p points at):

$$h_p = \sum_{q:p\to q} a_q$$

For example, if pages 1, 2, and 3 point to page 4, and page 4 points to pages 5, 6, and 7, es are computed as exemplified in Figure 7.

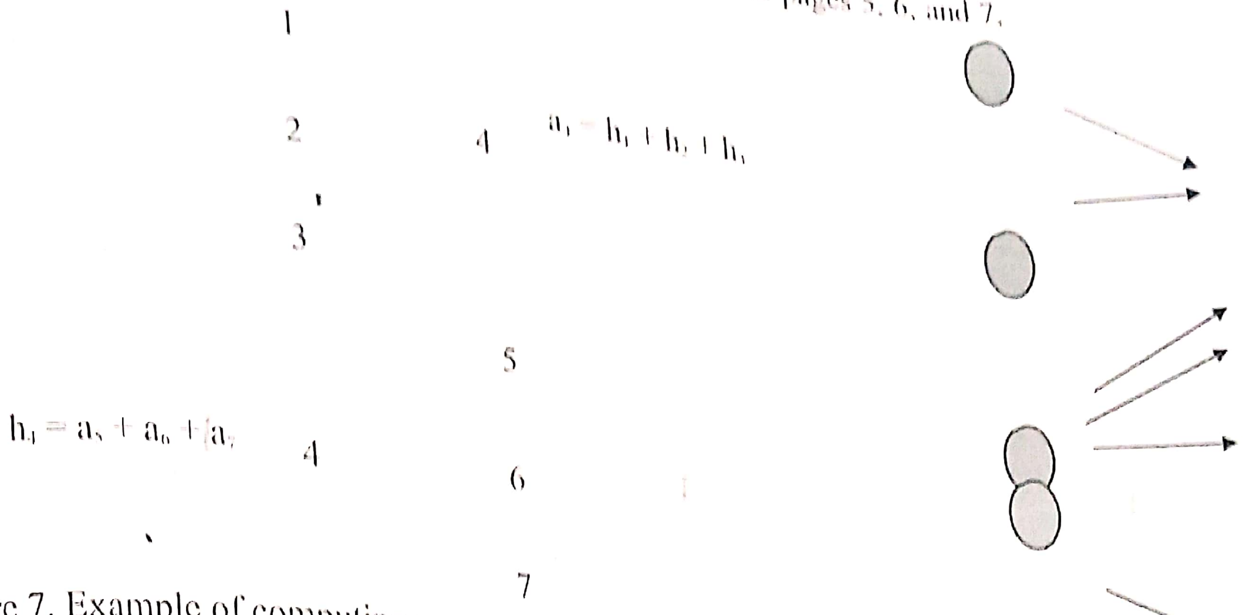


Figure 7. Example of computing scores for authorities and for hubs.

The hubs and authorities algorithm can be summarized as follows:

```
Initialize for all $p \in S$: $a_p = h_p = 1$
Repeat k times (where k is the number of iterations):
    For all $p \in S$:    $a_p = \sum_{q: q \to p} h_q$    (update authority scores)
    For all $p \in S$:    $h_p = \sum_{q: p \to q} a_q$    (update hub scores)
    For all $p \in S$: $a_p = a_p / c$ where $c$ is a constant such that: $\sum_{p \in S} (a_p / c)^2 = 1$
    For all $p \in S$: $h_p = h_p / c$ where $c$ is a constant such that: $\sum_{p \in S} (h_p / c)^2 = 1$
```

The algorithm converges to a *fix-point*, where the scores do not change at the next iteration. In practice, 20 iterations produce fairly stable results.

Google's PageRank

An alternative link-analysis method is PageRank, used by Google (the actual formula currently used by Google might be slightly different). PageRank does not attempt to capture the distinction between hubs and authorities, it ranks pages only by authority. It is applied to the entire Web rather than a local neighborhood of pages surrounding the results of a query.

If p is a given page and $q_1, \ldots, q_n$ are the pages that point to the page p, the page rank PR of p is given by the sum of the page ranks of all the pages $q_1, \ldots, q_n$ each of them divided by its number of outgoing links:

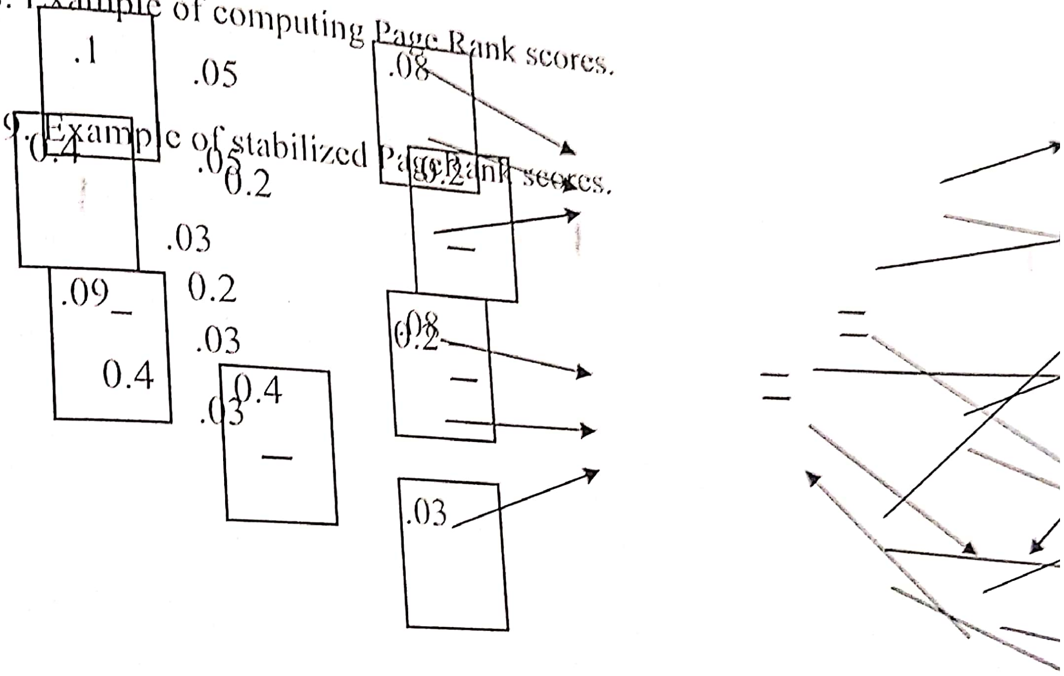
$$PR(p) = (1-d) + d \cdot (PR(q_1)/C(q_1) + \ldots + PR(q_n)/C(q_n))$$

$C(q_i)$ is number of links going out of the page q_i, and d is damping factor which can be en 0 and 1 (usually d is set to 0.85).

that the sum of all ranks of all the webpages needs to add up to 1. In fact the links that go f any page equally share its rank (due to the division of $PR(q_i)$ by $C(q_i)$). The page rank of a p is the sum of the weights of all its incoming links. Figure 8 shows a simplified example the PageRank values "flow" from pages to the pages they cite. After several iterations the values stabilize. Figure 9 shows an example of stable fix point.

Figure 8. Example of computing Page Rank scores.

Figure 9. Example of stabilized PageRank scores.



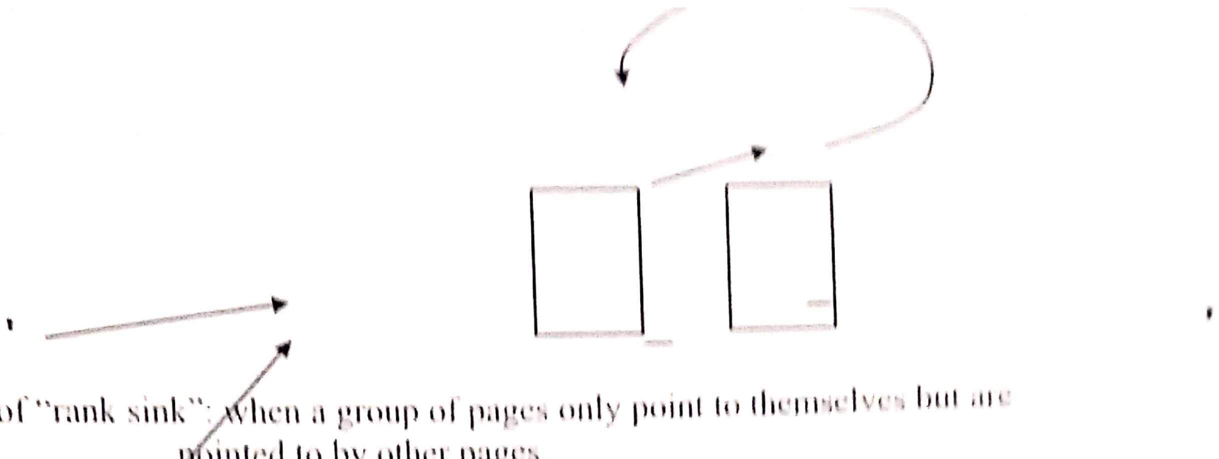


Figure 10. Example of "rank sink": when a group of pages only point to themselves but are pointed to by other pages.

There are complications when a group of pages only point to themselves but are pointed to by other pages; they act as a "rank sink" and absorb all the rank in the system (see Figure 10). That's why the additional factor *d* in the formula is needed. The PageRank algorithm models a "random surfer", that visits a page *p* with a probability given by its page rank *PR(p)*. The term *(1-d)* is the probability at each page that the "random surfer" will get bored and randomly jump somewhere else, allowing the surfer to get out of possible dead ends.

Commercial Aspects of Search Engines: Sponsored Links

Businesses pay to advertise on the major search engines. This would lead customers to their websites, if the customers following links returned by search engines as results of searching for specific terms. A business can bid for particular keywords. This is called *term leasing*. The links returned by search engines for commercial purposes are called *sponsored links*. They usually appear separate from other results, at the top, bottom or right side of the search engine results page. The *pay-per-click* advertising method allows search engines companies to charge a business proportional to the number of times users follow a sponsored link.

THE INVISIBLE WEB

The *Invisible Web* is the part of the Web that cannot be retrieved (seen) in the result pages from general Web search engines and in almost all subject directories.

Searchable Databases

Most of the Invisible Web is made up of the contents of thousands of specialized *searchable databases* that can be searched via the Web. The search results from many of these databases are delivered to the user in webpages that are generated just in answer to the search – *dynamic pages*. Such pages very often are not stored anywhere: it is easier and cheaper to dynamically generate the answer page for each query than to store all the possible pages containing all the possible answers to all the possible queries people could make to the database. Search engines cannot find these pages. If the only way to access webpages requires the user to *type something* or *select a combination of options*, spiders are unable to index the pages, because they cannot type or select options. Also, spiders crawl or navigate the Web by following the links in the webpages that are already in pages collected by their parent search engine. If there is no link to a page, a spider cannot "see" it.

Some searchable databases require a fee, and the users log in using passwords. Many are free; here are a few examples of free searchable databases:

http://www.freepint.com/gary/direct.htm
http://opcit.eprints.org/explorearchives.shtml
http://www.deepwebresearch.info/

Of particular value in academic research are:

Librarians Index http://lii.org/
AcademicInfo http://www.academicinfo.net/
Infomine http://infomine.ucr.edu/

Excluded Pages

There are also some types of pages that search engine companies *exclude by policy*. There is no technical reason they could not include them if they wanted. It's a matter of selecting what to include in indexes that are already huge and expensive to operate.

Some search engines may choose not to include pages because the *format of the document* would be infrequently or unsuccessfully searched by the users of the search engine. Pages formatted in PDF or pages that have very little HTML text might be excluded (though lately Google and other search engines index PDF files by transforming them into plain text with minimal HTML markup). Search engines also have a hard time indexing the contents of documents in Flash, Shockwave, and other programs like Word, WordPerfect, etc. Pages consisting almost entirely of images are often excluded as well. *Script-based pages* are usually excluded. HTML links containing a ? lead to script-based pages. A script is a type of programming language that can be used to fetch and display webpages. They can be used to create all or part of a webpage, and to communicate with searchable databases. Most search engines are instructed not to crawl sites or include pages that use script technology, although it is often technically possible for them to do so. This is a policy decision. If spiders encounter a ? in a URL or link, they are programmed to back off, because they could encounter poorly written scripts or intentional "spider traps" designed to ensnare spiders, sometimes bogging them down in infinite loops that run up the cost and time it takes for spiders to do their work. This may result in the contents of an entire site using scripts being excluded from a search engine, or a search engine may crawl a safe part of a site and omit others.

OTHER TYPES OF INFORMATION RETRIEVAL SYSTEMS

Multimedia Information Retrieval

There is a lot of multimedia content on the Web. The information retrieval systems described above were adapted to work with collections of images, video, or music. A query can be expressed as text, or as a sample image, or by humming a melody. If the query is in text form, the IR system can use the text in the caption of the images, or the text description of the music (composer, singer, album, etc.) in order to find the information. In this case the traditional IR technology described above is used. If the query is an image or a piece of music, it can be treated as a digital signal. Techniques such as vector space model can be extended to compute the milarity between two signals, where the features in the vectors will not be frequencies of ccurrence in text, but features extracted by digital signal processing techniques.

A multimedia IR system differs from a traditional IR system in several ways. First, the ructure of the multimedia objects is more complex than the structure of textual data; this quires integration of multimedia database management systems to adequately represent, anage, and store multimedia objects. Second, the similarity measure needs to be extended. The milarity measure is needed to match a query to a multimedia document, and to rank the rieved multimedia documents. Third, query languages are more complex. Depending on the pe of query, the search can be done only by content (image, music, etc.), only by text scriptions, or a combination of both. See chapters 11 and 12 of (Baeza-Yates and Berthier beiro-Neto, 1999) for more details.

gital Libraries

aditional libraries are among the first institutions to use IR systems, to create catalogs of cords for the material from the library. The catalogs can be search by users in the library or er the Web (online public access catalogs). These catalogs use database technology; the cords are structured according to standards such as MARC (title, a few subject headings, and a ssification number).

Modern libraries are being transformed to digital libraries as a result of the growth in ectronic publishing, which makes information available in a digital form. Through the Web, a ngle interface provides access to local resources, as well as remote access to databases in the sciences, humanities, and business, including full-text journals, newspapers, and directories. pecial collections, in multimedia not only in text format, become available through the same iteway. For more details about the technology of digital libraries see, for example, (Lesk, 1997).

Many libraries, particularly academic and large public libraries, have undertaken digital prary project to achieve interoperability and ease of use and access. Two such projects are the os Angeles Public Library's Virtual Electronic Library project (http://www.lapl.org), and niversity of Pennsylvania's Digital Library (http://www.library.upenn.edu). A digital library ould have no connection to an actual library, for example the ACM Digital Library http://portal.acm.org/dl.cfm) that contains journal and conference publications in Computer cience.

Digital libraries are more than complex IR systems. They are social systems centered round various communities of users. They also have component for building, cataloging, naintaining, and preserving repositories. There are many international or national digital library projects. One such project is the Digital Libraries Initiative (DLI) (phase one 1994-1998, phase two in progress) supported by the National Science Foundation (NSF), the Department of Defense Advanced Research Projects Agency (DARPA) and the National Aeronautics and Space Administration (NASA). The DLI phase one contained large research projects at six universities: University of Illinois Urbana-Champaign, Carnegie-Mellon University, Stanford University, University of California at Berkeley, University of California at Santa Barbara and University of Michigan. These projects are developing the next generation of tools for information discovery, management, retrieval and analysis.

Distributed Information Retrieval Systems

When the collection of documents is huge, it can be distributed over many computers. Parallel computing can be used to speed up processing. Document partitioning can be used to divide the search task into multiple, self-contained tasks that each involve extensive computation and data processing with little communication between them. Collections can be divided by topics, or for administrative purposes. When the collection is distributed, an index can be built for each artition, but a centralized index is still needed in order to direct the search for the terms in the ser's query. To build a distributed IR system, algorithmic IR issues need to be considered ogether with engineering issues common to distributed systems in general. The main ngineering issues involve: defining a search protocol for transmitting requests and results; lesigning servers that can efficiently accept requests, initiate subprocesses or threads to service equests, and exploit any locality inherent in the processing using appropriate caching echniques: designing a broker that can submit asynchronous search requests to multiple servers n parallel and combine the intermediate results into a final end user result. The algorithmic ssues involve: how to distribute documents across the distributed search servers, how to decide vhich servers should receive a particular search request, and how to combine the results from the lifferent servers.

A special type of distributed IR systems are Peer-to-Peer IR systems (P2P), when the nformation can be repeated on several computers, and there is no centralized access control. In a P2P system, the nodes (servers) are independent; each node can leave or enter the system any ime. Examples of P2P systems are Gnutella and Napster. See chapter 9 of (Baeza-Yates and Berthier Ribeiro-Neto, 1999) for more details about distributed systems.

CONCLUSION AND FUTURE DIRECTIONS

This chapter presented an overview of the methods used in information retrieval and search engines. The technology of search engines is a very dynamic field, always looking for improvements and new ideas in order to satisfy user needs. Future trends in search engines include technology that is yet in the stage of research prototypes. Multimedia IR systems on the Web are becoming more important, as more video, music, and other types of data are available on the Web and fast Internet access becomes common.

Natural Language Queries

Text-based IR systems will also evolve. Users could express their queries in natural language, not just as keywords. This requires deeper syntactic and semantic analysis of the queries and the documents. Allowing the user to orally describe the information need into a microphone is a more natural way to interact with a search engine (Crestani, 2002). Spoken queries need to be translated into text queries using a Speech Recognition system (though current speech recognition technology would introduce recognition errors that might hurt retrieval performance). Cross-language Information Retrieval systems become available (Savoy, 2003). The queries can be a language in which the user feels comfortable, while the documents are in another language. This requires automatic translation of the queries before matching them to documents for retrieval.

Information Extraction techniques look at retrieving specific pieces of information from ocuments rather than showing to the user long lists of links to documents. These systems tend o work only in specific domains, such as biomedical text (Mooney and Bunescu, 2005) or ewspaper text describing terrorist attacks (Rillof, 1999). Questions Answering techniques return concise answer to a query expressed in natural language. They require deep semantic analysis n order to match queries to selected sentences in the documents (Harabagiu *et al.*, 2000). impler methods looked at exploring redundancy on the Web: extract answers from many ebpages, and even if some answers are wrong, selecting the answer that has a majority of votes ften leads to a correct response (Clarke *et al.*, 2001).

he Semantic Web and Use of Meta-Data

Most of the current forms of Web content are designed to be presented to humans; they e not understandable by computers. The Semantic Web aims at enhancing existing Web ntent with semantic structure in order to make it meaningful to computers as well as to humans. e Semantic Web project (http://www.semanticweb.org/) provides support for adding semantic notations (meta-data that describes their content) to webpages or multimedia objects. To press the meta-data there is a need for standardized vocabularies and constructs explicitly and mally defined in domain *ontologies* (sets of domain concepts and relations between them).

The performance of current search engines and IR systems suffers because of the biguity of the natural language: words in documents and queries have multiple meanings and e retrieval results often include the wrong meanings in addition to the desired meanings. Better sults will be achieved if webpages contain precise semantic annotations. This will allow search ents to navigate, collect, and utilize information on the Web in more reliable ways.

sualization and Categorization of Results

arch engines tend to retrieve many documents in answer to a user query. Often users look only the first 10 documents. When recall in important to the user, a long list is not a good way of splaying the results. The list does not show the distribution of the different categories of swers. Various ideas are tried in order to present the results in more manageable ways, for ample 2-dimensional maps or 3-dimentional visualizations (Chen *et al.*, 1998). Automatic ustering techniques can be used to discover clusters of similar documents. Each cluster will en be an object in the visual representation.

GLOSSARY

oolean Model: a classic model of document retrieval based on classic set theory. Uses Boolean operator such as AND, OR, NOT.

igital Library: a complex system composed of: a repository of heterogeneous digital objects; descriptions of these objects (meta-data); a set of users; systems for capturing, indexing, cataloging, searching, browsing, delivery, archiving, and preserving the repository.

Distributed Information Retrieval: IR systems that distribute the data collection and the computation over multiple servers.

ndex: a data structure built to speed up the search. For each keyword, it records the number of occurrences in documents and possibly other information.

nformation Retrieval (IR): part of Computer Science that studies the retrieval of information (not data) from a collection of written documents. The retrieved documents aim at satisfying a user information need usually expressed in natural language.

nvisible Web: the part of the Web not indexed by search engines, mostly composed of searchable databases. These databases produce dynamic HTML pages as results to queries; therefore the pages cannot be indexed by search engines.

atent Semantic Indexing: a model of information retrieval that extends the classic vector space model; it reduces the dimension of the vector space; the dimensions are no longer the index terms, they approximate concepts.

ean Average Precision: an information retrieval performance measure that combines precision and recall and rewards relevant documents ranked higher in the list of retrieved documents. Computed as the average of the precision values for each relevant document in the ranked results.

eta-data: description of the data (in XML or other description language)

eta-search: a search technique where a single entry point is provided to multiple heterogeneous search engines. The user query is sent to these search engines and a unified list of results is presented to the user.

ultimedia Information Retrieval: IR systems that deal with images, video, audio, music or other multimedia objects.

age Ranking: methods for ranking webpages by their popularity, for example based on the number of links that point to a page.

eer-to-Peer Information Retrieval: Distributed IR systems where the nodes are independent computers that can leave or join the system any time.

Precision: an information retrieval performance measure that quantifies the fraction of the retrieved documents which are relevant.

Probabilistic Model: a model of information retrieval based on a probabilistic interpretation of document relevance to a user query.

Query: the expression of the user information need in the input language of the information system. Usually keywords and sometimes a few Boolean connectives (AND, OR, NOT).

Recall: an information retrieval performance measure that quantifies the fraction of know relevant documents that are among the retrieved documents.

vance Feedback: an interactive process of obtaining feedback from he user about the levance or non-relevance of the retrieved documents.

ch Engine: An IR system designed to find information on the Web, to index webpages in der to be able to retrieve them as result of a user query.

mming: a technique for reducing a word to its root form.

pwords: words that occur frequently in texts, for example articles, prepositions, and onjunctions.

r Information Need: a natural language declaration of the informational need of a user.

ctor Space Model: a classic model of document retrieval based on representing documents nd queries as vectors of index terms.

b Crawler (Web Spider or Robot): a program that collects HTML pages from the Web by ollowing links from the collected webpage.

BIBLIOGRAPHY

cardo Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*, Addison-Wesley Publishing Company.

ergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. WWW7 / *Computer Networks* 30(1-7): 107-117.

hris Buckley, Gerard Salton, and James Allan. 1993. Automatic retrieval with locality information using SMART. In *Proceedings of the First Text REtrieval Conference* (TREC-1), pages 59–72. NIST Special Publication 500-207.

Charles L. A. Clarke and Gordon V. Cormack and Thomas R. Lynam. 2001. Exploiting Redundancy in Question Answering. In *Research and Development in Information Retrieval*, pages 358-365.

Hsinchun Chen, Andrea L. Houston, Robin R. Sewell, and Bruce L. Schatz. 1998, Internet browsing and searching: User evaluations of category maps and concept space techniques *Journal of the American Society of Information Science*, 49(7):582-608.

Fabio Crestani. 2002. Spoken Query Processing for Interactive Information Retrieval. Data ar Knowledge Engineering, 41(1):105-124.

Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard rshman. 1990. Indexing by latent semantic analysis. *Journal of the Society for Information ence*, 41(6):391-407.

al Library Initiative. 2006. http://www.dli2.nsf.gov/

da Harabagiu, Marius Pasca and Steven Maiorano. 2000. Experiments with Open-Domain Textual Question Answering, in *Proceedings of COLING-2000*, August 2000, Saarbruken, Germany. pages 292-298.

n Kleinberg. 1999. Hubs, Authorities, and Communities. In *ACM Computing Surveys*, 31(4).

ichael Lesk. 1997. *Practical Digital Libraries; Books, Bytes, and Bucks*. Morgan Kaufmann.

aymond Mooney and Razvan Bunescu. 2005. Mining Knowledge from Text Using Information Extraction. *SIGKDD Explorations, Special issue on Text Mining and Natural Language Processing*, 7(1):p.3-10.

Martin F. Porter. 1980. An algorithm for suffix stripping. Program, 14(3): 130-137. http://www.tartarus.org/martin/PorterStemmer/

Ellen Riloff. 1999. Information Extraction as a Stepping Stone toward Story Understanding. In *Computational Models of Reading and Understanding*, Ashwin Ram and Kenneth Moorman, eds., The MIT Press.

Stephen E. Robertson and Karen Sparck-Jones. 1976. Relevance weighting of search terms. Journal of the American Society for Information Sciences, 27(3):129-146.

Stephen E. Robertson, Steve Walker, and Micheline Hancock-Beaulieu. 2000. Experimentation as a way of life: Okapi at TREC. Information Processing and Management 36(1):95-108.

Gerard Salton and Chris Buckley. 1988. Term-weighting approaches in automatic retrieval. *Information Processing and Management*, 24(5):513-523.

Gerard Salton. 1989. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley Publishing Company.

Jacques Savoy. 2003. Cross-language information retrieval: experiments based on CLEF 200 corpora. *Information Processing and Management* 39(1).

Semantic Web project. 2006. http://www.semanticweb.org/