CHAPTER 31

WEB SEARCH TECHNOLOGIES FOR TEXT DOCUMENTS

Weiyi Meng

SUNY, BINGHAMTON

Clement Yu

UNIVERSITY OF ILLINOIS, CHICAGO

Abstract

The World Wide Web has become the largest information source in recent years, and search engines are indispensable tools for finding needed information from the Web. While modern search engine technology has its roots in text/information retrieval techniques, it also consists of solutions to unique problems arising from the Web such as web page crawling and utilizing linkage information to improve retrieval effectiveness. The coverage of the Web by a single search engine may be limited. One effective way to increase the search coverage of the Web is to combine the coverage of multiple search engines. Systems that do such combination are called metasearch engines. In this chapter, we describe the inner workings of text retrieval systems, search engines, and metasearch engines.

Key Words: search engine, Web, Surface Web, Deep Web, Web crawler, text retrieval, Vector space model, PageRank, hub and authority, metasearch engine, database selection, result merging

INTRODUCTION

The World Wide Web has emerged as the largest information source in recent years. People all over the world use the Web to find needed information on a regular basis. Students use the Web as a library to find references, and customers use the Web to purchase various products and services. It is safe to say that the Web has already become an important part in many people's daily lives.

The precise size of the Web is a moving target as the Web is expanding very quickly. The Web can be divided into the Surface Web and the Deep Web (or Hidden Web) (Bergman 2000). The former refers to the collection of web pages that are publicly and directly accessible without the need to go through a registration, login, or a search engine. Usually, each such page has a static logical address called the Uniform Resource Locator or URL. It was estimated that the Surface Web contained about 2 billion web pages in 2000 (Bergman 2000) but now (June 2009) it has more than 45 billion web pages according to www.worldwidewebsize.com. The Deep Web contains web pages that are not publicly accessible or are dynamically generated. As an example, a publisher may have accumulated many articles in digital format. If these articles are not placed on the Surface Web (i.e., there are no static URLs for them) but they are accessible by web users through the publisher’s search engine, then these articles belong to the Deep Web. Web pages that are dynamically generated using data stored in database systems also belong to the Deep Web. The size of the Deep Web was estimated to be about 500 billion pages in 2000 (Bergman 2000), and it is now about 1 trillion pages (Zillman 2009).

In the last several years, many search engines have been created to help users find desired information on the Web. Search engines are easy-to-use tools for searching the Web. Based on what type of data is searched, there are document-driven search engines and database-driven search engines. The former searches documents (web pages) while the latter searches data items from database systems through Web-based search interfaces. Database-driven search engines are mostly employed for e-commerce applications such as buying cars or books. This chapter concentrates on document-driven search engines only. When a user submits a query, which typically consists of one or more key words that reflect the user's information needs, to a search engine, the search engine returns a list of web pages (usually their URLs and snippets) from the set of web pages covered or indexed by the search engine. Retrieved web pages are displayed to the user based on how well they are deemed to match with the query, with better-matched ones displayed first. In 2008 July, Google (`www.google.com`), Yahoo (`www.yahoo.com`), MSN (`www.msn.com`), Ask (`www.ask.com`), and AOL (`www.aol.com`) were the most popular document-driven search engines for searches performed in the United States. Based on the 2008 August comScore report (comScore 2008), these five search engines have the largest search shares for 2008 July and their shares are 61.9 percent, 20.5 percent, 8.9 percent, 4.5 percent, and 4.2 percent, respectively. Search engines can generate revenues by placing advertisements on returned result pages, and a search engine's popularity has a direct impact on the amount of revenue it can generate. The Deep Web is usually accessed through Deep Web search engines (like the publisher's search engine we mentioned above). Each Deep Web search engine usually covers a small portion of the Deep Web.

While using a search engine is easy, building a good search engine is not. In order to build a good search engine, the following issues must be addressed. First, how does a search engine find the set of web pages it wants to cover? After these pages are found, they need to be preprocessed so that their approximate contents can be extracted and stored within the search engine. The approximate representation of web pages is often called the *index* of the search engine. So the second issue is how to construct such an index. Another issue is how to use the index to determine whether a web page matches well with a query. These issues will be discussed in the sections on Text Retrieval and Search Engine Technology of this chapter. More specifically, in the section on Text Retrieval, we will provide an overview of some basic concepts on text retrieval, including how to build an index for a given document collection and how to measure the similarity of a document to a query; in the section on Search Engine Technology, we will provide detailed descriptions about how a search engine finds the web pages it wants to cover, what are the new ways to measure how well a web page matches with a query, what are the techniques to organize the results of a query, and how advertisements are placed on result pages.

Due to the huge size and the fast expansion of the Web, each search engine can only cover a small portion of the Web. One of the largest search engines on the Web, Google (`www.google.com`), for example, has a collection of about 20 billion Web pages (`www.worldwidewebsize.com`). But the entire Web, including the Deep Web, is believed to have more than 900 billion pages. Current general search engines are mostly limited to the Surface Web due to the difficulty in crawling data from the Deep

Web. One effective way to increase the search coverage of the Web and to reach the Deep Web is to combine the coverage of multiple search engines, including Deep Web search engines. Systems that do such combination are called *metasearch engines*. A metasearch engine can be considered as a system that supports unified access to multiple existing search engines. A metasearch engine does not maintain its own collection of web pages, but it may maintain information about its component search engines in order to achieve higher efficiency and effectiveness. In a typical session using a metasearch engine, the user submits a query to the metasearch engine which passes the query to its component search engines; when the metasearch engine receives the search results returned from its component search engines, it merges these results into a single ranked list and displays them to the user. Techniques that can be used to build efficient and effective metasearch engines are reviewed in the section on Metasearch Engine Technology of this chapter.

TEXT RETRIEVAL

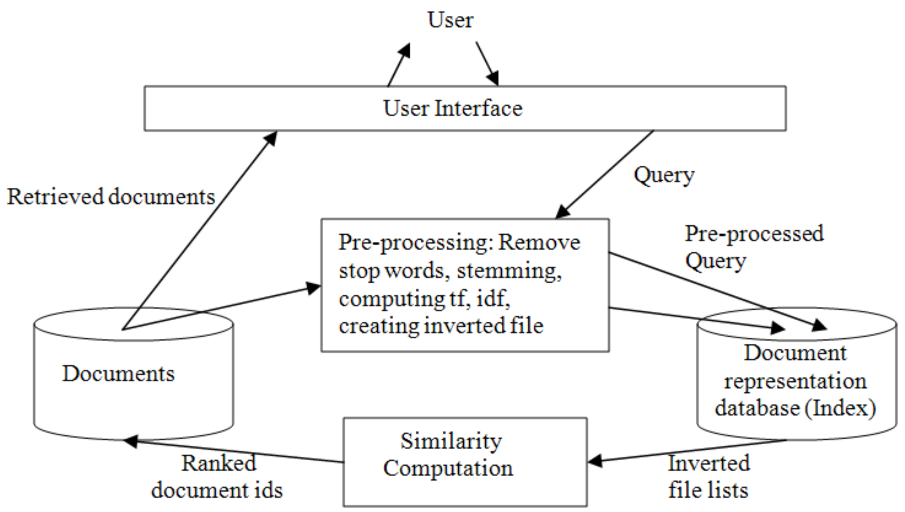
Text (information) retrieval deals with the problem of how to find relevant (useful) documents for any given query from a collection of text documents. Text retrieval technology has a profound and direct influence on Web search engines. In fact, the first-generation search engines (around 1995–1997) were built almost entirely based on traditional text retrieval technology with web pages as the text documents. In this section, we provide an overview of some basic concepts in classical text retrieval. The overview is primarily based on the *vector space model* where both documents and user queries are represented as vectors of terms with weights (Salton and McGill 1983). Several other popular models will also be very briefly mentioned. Text retrieval is a large discipline, and it is not possible to cover all major aspects of this discipline in reasonable details in this article. Readers who want to read more about this subject are referred to textbooks covering this subject, for example, Salton and McGill 1983) and Frakes and Baeza-Yates (1992).

In the subsection on System Architecture, the architecture of a basic text retrieval system is introduced. In the subsection on Document Representation, basic techniques for representing documents and queries to facilitate efficient and effective retrieval are discussed. In the subsection on Document-Query Matching, several methods that measure the degree of closeness (similarity) between documents and queries are reviewed. The similarity between a document and a query determines how high the document is ranked in the result list of the query. The next subsection briefly introduces basic ideas on efficient evaluation of queries. Then comes the subsection on basic concepts on measuring the effectiveness of text retrieval systems.

System Architecture

The architecture of a basic text retrieval system is shown in Figure 31.1. Documents in the document collection of a text retrieval system are preprocessed to identify terms representing each document, to collect certain statistics about the terms and to organize the information in a certain format (i.e., the *index* in Figure 31.1) to facilitate the fast computation of the similarity of each document with respect to any query. When a user query is received, it is first preprocessed to identify terms that represent the query and compute term statistics; then the text retrieval system computes the similarities of the documents with the query using the prebuilt index and ranks the documents in descending order of their similarities. More details about these concepts and operations will be provided in the next several subsections.

Figure 31.1 Architecture of a Basic Text Retrieval System



Document Representation

The contents of a document may be represented by the words contained in it. Some words such as “a,” “of,” and “is” do not contain topical content information. These words are called *stop words*. Many text retrieval systems remove stop words from documents for efficiency and matching precision considerations while some systems choose to keep them because they may be used in phrases. Variations of the same word may be mapped to the same term. For example, the words “calculate,” “calculation,” and “calculating” can be denoted by the term “calcul.” This can be achieved by a *stemming program*, which removes suffixes or replaces them by other characters. After removing stopwords and stemming, each document can be logically represented by a vector of *n* terms (Baeza-Yates and Ribeiro-Neto 1999), where *n* is the total number of distinct terms in the set of all documents in a document collection. It should be noted that while removing stop words and stemming are widely practiced in traditional text retrieval, different text retrieval systems often employ different stop word lists and/or stemming algorithms. Many present search engines do not actually remove stop words. Furthermore, a term does not necessarily mean a single word; it could be a phrase.

Suppose the document *d* is represented by the vector ($d_1, ..., d_i, ..., d_n$), where d_i is a number (weight) indicating the importance of the *i*-th term in representing the contents of the document *d*. Most entries in the vector will be zero because most terms do not appear in any single document. When a term is present in a document, the weight assigned to the term is usually based on two factors, namely the *term frequency* (*tf*) factor and the *document frequency* (*df*) factor. The term frequency of a term in a document is the number of times the term appears in the document. Intuitively, the higher the term frequency of a term is, the more important the term is in representing the contents of the document. Consequently, the *term frequency weight* (*tfw*) of a term in a document is usually a monotonically increasing function of its term frequency. The document frequency of a term is the number of documents containing the term in the entire document collection. Usually, the higher the document frequency of a term is, the less important the term is in differentiating different documents. Thus, the weight of a term with respect to document frequency is usually a monotonically decreasing function of its document frequency and is called the *inverse document frequency weight* (*idfw*). The weight of a term in a document can be the product of its term frequency weight and its inverse document frequency weight, that is, *tfw* * *idfw*. The weight of a term in a document may also be affected by other factors such as where it appears in the document; for example, the weight may be increased if the term appears in the title of the document. Please refer to the subsection on the Use of Tag Information to see some other factors in the context of web pages.

A typical query for text retrieval is also written in text. It can be internally transformed into an *n*-dimensional vector as well, following the same process for transforming documents to vectors described above.

Document-Query Matching

After all documents and a query have been represented as vectors of the same dimension, a similarity between the query vector and each document vector can be calculated using a *similarity function*. Then documents whose corresponding vectors have high similarities with the query vector are retrieved. Let $q = (q_1, ..., q_n)$ and $d = (d_1, ..., d_n)$ be the vectors of a query and a document, respectively. One simple similarity function is the *dot product function*, $dot(q, d) = \sum_{i=1}^{n} q_i * d_i$. Essentially, this function is a weighted sum of the terms in common between the two vectors. For a given query, the dot product function tends to favor longer documents over shorter ones because a longer document is more likely to have more terms in common with a given query than a shorter document. One way to remedy this bias is to employ the *cosine function* (Salton and McGill 1983), given by $dot(q,d)/(|q|\cdot|d|)$, where $|q|$ and $|d|$ denote, respectively, the lengths of the query vector and the document vector. The *cosine function* between two vectors has a geometric interpretation—it is the cosine of the angle between the two vectors. In other words, the *cosine function* measures the angular distance between a query vector and a document vector. When the weights in vectors are non-negative, the *cosine function* always returns a value between 0 and 1. It gets the value 0 if there is no term in common between the query and the document (i.e., when the angle is 90°); its value is 1 if the query and the document vectors are identical or one vector is a positive constant multiple of the other (i.e., when the angle is 0°). There are other similarity functions, and some of them also take into consideration how close the query terms appear in a document. The closer the query terms appear in a document, the higher the similarity between the query and the document should be. This is because when the query terms appear in close proximity, the meaning of the query is more likely to be retained by the document. To support proximity-based match, for any given document and a term, the positions of the term in the document need to be collected and stored. The ordering of terms in the query and their relative positions in a document can also be taken into consideration.

Other models for text retrieval also exist and we briefly mention some of them here. The basic *Boolean retrieval model* retrieves documents based on the presence and absence of query terms in a document and the weights of the terms are not considered. A Boolean query can contain one or more Boolean operators (AND, OR, and NOT). As an example, query "((t1 AND t2) OR t3) and NOT(t4)" is to retrieval all documents that contain either the two terms t1 and t2 or a term t3, but not term t4. Another popular model is the *probabilistic model*. In this model, the documents are ranked in descending order of the probability that a document will be relevant with respect to a query. The probabilities are estimated based on the distribution of the query terms in relevant and irrelevant documents. One of the most widely used similarity functions based on the probabilistic model is the *Okapi function* (Robertson and Walker 1999). In recent years, the *language model* has also been applied to information retrieval with good success (Ponte and Croft 1998; Zhai and Lafferty 2004). In this approach, for a given query, we estimate the probability that the query can be generated based on each document and then rank the documents in descending order of the probabilities.

Efficient Query Evaluation

Computing the similarity between a query and every document directly is inefficient because most documents do not have any term in common with a given query and computing the similarities for these documents is a waste of resources. This approach is also impractical when the document collection is large due to the high computational overhead required and the short response time desired. To illustrate this point more clearly, let us consider the *document-term matrix* of a document collection as shown in Figure 31.2, where N is the number of documents in the collection, n is the number of distinct terms of the collection, and w_{ij} is the weight of the j-th term in the i-th document. Each row in the matrix corresponds to a document while each column corresponds a term. If the j-th term does not appear in the i-th document, then w_{ij} will be zero. Most entries in the matrix will be zero because most documents will have a relatively small number of distinct terms in comparison to the total number of distinct terms in a large collection.

Figure 31.2 A Document-Term Matrix

	t_1	t_2	……	t_n
D_1	w_{11}	w_{12}	……	w_{1n}
D_2	w_{21}	w_{22}	……	w_{2n}
……	……	……	……	……
D_N	w_{N1}	w_{N2}	……	w_{Nn}

If each document were compared with the query directly to compute the similarity of the document, then all or most non-zero entries in the matrix would need to be accessed. This would be very inefficient considering that most queries contain only a few terms and only the entries of the matrix that correspond to the query terms are needed for the similarity computations.

To improve the efficiency, an inverted file index is created in advance. For each term t_j, an inverted list of the format $[(D_{j_1}, w_{j_1 j}), \ldots, (D_{j_k}, w_{j_k j})]$ with a header is generated and stored, where D_{j_i} is the identifier of a document containing t_j, $w_{j_i j}$ is the weight of t_j in D_{j_i}, $1 \le i \le k$, and k is the number documents containing t_j. Note that the inverted list of t_j corresponds to the j-th column in the document-term matrix. In addition, a hash table, which is a table-like data structure, is created to map each given term to the header of the inverted list of the term. These two data structures, namely the inverted file and the hash table, permit efficient calculation of the similarities of all those documents that have positive similarities with any query. Specifically, consider a query with m terms. For each query term, the hash table is used to quickly locate the inverted file list for the term. The m inverted file lists essentially contain all the data needed to calculate the similarity between the query and every document that contains at least one query term. Essentially, with this approach, only the non-zero entries in the m columns of the document-term matrix that correspond to the m query terms need to be accessed, leading to a very substantial saving of the processing cost over the method that requires all or nearly all entries in the matrix to be processed.

A widely used query evaluation strategy is the *document-at-a-time strategy* (Turtle and Flood 1995), which computes the similarity for one document at a time. The basic idea of this strategy is described as follows. In many situations, the inverted file of a text retrieval system is too large to fit in the main memory of the system and is thus stored on disk. If the inverted file is on disk, then during the evaluation of a query, we first bring the inverted file lists of all the query terms into the main memory at the same time. We then compute the similarities of the documents containing one or more of the query terms, one document at a time. Another well-known query evaluation strategy is the *term-at-a-time strategy* (Turtle and Flood 1995). This strategy processes the inverted file lists of the query terms one by one; after the inverted file list of a query term is processed, the contribution of this term to the overall similarity between the query and each document that contains this query term is computed and then added to the contribution by query terms that have already been processed. After the inverted file lists for all query terms are processed, the final similarity between the query and each document containing at least one query term is computed. There are a number of pruning strategies to reduce the computational overhead of the above two basic evaluation strategies.

Note that if proximity conditions between query terms need to be considered, the positions of each term in every document need to be saved in the inverted file. In this case, the inverted list for term t_j is extended to $[(D_{j_1}, w_{j_1 j}, L_{j_1}), ..., (D_{j_k}, w_{j_k j}, L_{j_k})]$, where L_{j_i} contains the list of positions of t_j in D_{j_i}. Since a proximity condition involves two or more terms and usually all the location lists of these terms need to be present in order to evaluate the condition, the *document-at-a-time strategy* is more appropriate than the *term-at-a-time strategy* for evaluating proximity queries.

Effectiveness Measures

The goal of text retrieval is to find documents relevant (useful) to any given query and rank them high in the result list. The retrieval effectiveness of a text retrieval system is often measured by a pair of quantities known as *recall* and *precision*. Suppose, for a given user query, the set of relevant documents in the document collection is known. Recall and precision are defined as follows:

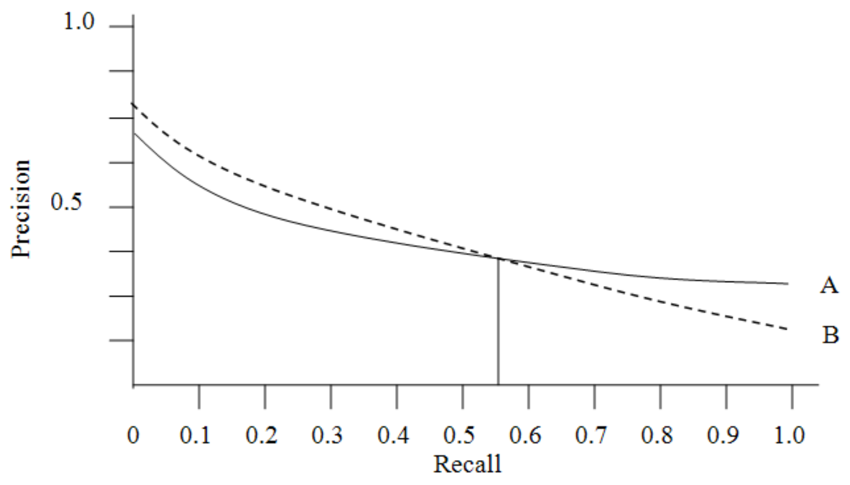
$$recall = \frac{the \quad number \quad of \quad retrieved \quad relevant \quad documents}{the \quad number \quad of \quad relevant \quad documents}$$

$$precision = \frac{the \quad number \quad of \quad retrieved \quad relevant \quad documents}{the \quad number \quad of \quad retrieved \quad documents}$$

As an example, suppose there are 10 relevant documents for a query and among the 20 retrieved documents, 4 are relevant. Then for this query, the recall is 4/10 = 0.4 and the precision is 4/20 = 0.2.

To evaluate the effectiveness of a text retrieval system, a set of test queries is often used. For each query, the set of relevant documents is identified in advance. For each test query, a precision value for each distinct recall value is obtained. Usually only eleven recall values, 0.0, 0.1, ..., 1.0, are considered. When the precision values at each recall value are averaged over all test queries, an average recall-precision curve is obtained. This curve is used as the measure of the effectiveness of the system. Figure 31.3 shows two 11-point recall-precision curves representing the performance of two text retrieval systems A and B. It can be seen that System B is better than System A when the recall is in interval [0.0, 0.56], but the reverse is true when the recall is in the interval [0.56, 1.0].

Figure 31.3 Comparing the Effectiveness of Two Text Retrieval Systems



Data sets for evaluating text retrieval systems are available from NIST (the National Institute of Standards and Technology) in the Text Retrieval Conference (TREC) series (`trec.nist.gov`). The current TREC data sets include relevance judgments for several hundreds of topics (queries) over several document collections.

There are also many other metrics for measuring the retrieval effectiveness of text retrieval systems. In the context of search engines, it is often impossible to know the complete set of relevant documents for test queries. Therefore, some precision-oriented metrics can be used to measure the effectiveness based on certain top-ranked results. For example, *precision at n* can be used to compute the precision of the top *n* ranked results for some *n*. Readers may refer to book by Voorhees and Harman (2005) for more information regarding the evaluation methodologies and the evaluation measures used in TREC.

SEARCH ENGINE TECHNOLOGY

First-generation web search engines are essentially text retrieval systems for web pages. However, the Web environment has some special characteristics that make building modern search engines significantly different from building traditional text retrieval systems. In this section, we review these special characteristics as well as techniques that address/explore these characteristics.

The following are some of the special characteristics of the Web environment.

Web pages are stored at numerous autonomous Web servers. A program known as *Web robot* or *Web crawler* is needed to gather them so that they can be processed for later search. Web crawlers are described in their own subsection.

Queries submitted to search engines have different characteristics from those submitted to traditional text retrieval systems due to the nature of the Web and the nature of the users. For example, Web users conduct not only searches but also business activities, such as purchasing products, on the Web. Therefore, Web users have more diverse needs when using a search engine compared to their counterparts when using traditional text retrieval systems. Some analyses about the characteristics of Web search queries are introduced in the section on Web Search Queries.

Web pages are highly tagged documents, that is, tags that control the presentation of contents on a Web browser and/or define the structures/semantics of the contents are embedded in web pages. At present, most web pages are in HTML (HyperText Markup Language) format. Tags in web pages often convey rich information regarding the terms in these pages. For example, a term appearing in the title of a page or a term that is highlighted by special font can provide a hint that the term is rather important in reflecting the contents of the page. Traditional text retrieval systems are usually based on plain text documents that are either not or rarely tagged. The section on the Use of Tag Information discusses some techniques that utilize these tags to improve retrieval effectiveness.

Web pages are linked to each other. A link from page A to page B allows a web user to navigate from page A to page B. Such a link also contains several pieces of information that are useful to improve retrieval effectiveness. First, the link indicates a good likelihood that the contents of the two pages are related (Davison 2000). Second, the author of page A considers page B to be of some value. Third, the clickable text associated with the link, called *anchor text* of the link, usually provides a short description of page B (Davison 2000). The subsection on the Use of Linkage Information discusses several methods for utilizing link information among web pages to improve the search engine retrieval performance.

Search engines have become important tools to generate revenue for search engine operators. The most common way to generate revenue is to display advertisements on the pages that show the search results to users. In the section on the Sponsored Search, a common method for placing advertisements on result pages is briefly reviewed.

In this section we also discuss two issues that may occur in both search engines and traditional text retrieval systems, but they have a special flavor in search engines. These two issues are the use of user profiles to support personalized search and the organization of search results on result pages. These two issues will be covered in the subsection on the Use of User Profiles and on Result Organization respectively.

Web Crawler

A Web crawler is a program for fetching Web pages from remote Web servers. Web crawlers are widely used to build the web page collection for a search engine.

The main idea behind the crawler program is quite simple. Note that each web page has a URL (universal resource locator) that identifies the location of the page on the Web. The program takes one or more seed URLs as input to form an initial URL list. The program then repeats the following steps until either no new URLs can be found or enough pages have been fetched: (1) take the next URL from the URL list and fetch the corresponding web page from its server using the HTTP (hypertext transfer protocol); (2) from each fetched web page, extract new URLs and add them to the list. A web crawler may follow a breadth-first crawling strategy or a depth-first crawling strategy. The former is achieved by implementing the URL list as a *queue* so that the new URLs are always added at the end of the list; the latter is realized by implementing the URL list as a *stack* so that the new URLs are always added at the beginning of the list. By adding only new URLs of the web pages that are related to a certain topic (say movies) to the URL list, the web crawler becomes a *focused crawler* for pages related to the topic. Focused crawlers are very useful for creating domain-specific search engines when only limited resources are available for crawling (Chakrabarti, van den Berg, and Dom 1999). The effectiveness of a focused crawler depends on how accurately the crawler can predict whether a URL will bring a page or pages related to the topic of interest. One technique useful for such prediction is based on whether the URL, the anchor text associated with the URL and the text adjacent to the anchor text contain terms related to the topic.

Web Search Queries

Traditional text retrieval systems are developed for the sole purpose of finding text documents that contain information relevant to users' information needs. Often the users are well trained to write high quality queries. In contrast, search engines are search tools on the Web where most users have little or no training about writing good queries and where users often conduct other activities that are not searches but can be helped by searches. For example, suppose a user wants to buy a book from the Web. If the user does not already know which web site(s) sell the book, she can use a search engine to find such a web site and then make the purchase at the web site.

An analysis of approximately 1 billion queries submitted to the AltaVista search engine (`www.altavista.com`) in 1998 revealed some interesting characteristics about Web search queries (Silverstein et al. 1999). For example, these queries were usually short: the average number of terms in a query was 2.35, about 26 percent of the queries had a single term, and more than 70 percent of the queries had less than three terms. Other findings included that about 80 percent of the queries did not use any operators (e.g., Boolean operators such as "AND," "OR," etc.), a small number of queries had very high repetition rate (the top 25 most common queries accounted for 1.5

percent of the total number of queries), and for about 85 percent of the queries, only the results in the first result page were viewed by the users. These findings suggest that it is very important for search engines to handle short queries that do not use any operators well and to achieve high precisions for top-ranked results.

In another study, Broder analyzed users' needs behind their Web search queries based on a user survey (Broder 2002). The study found that users often have different needs when submitting their queries to a search engine compared to the users of traditional text retrieval systems. Based on this study, Broder classified web queries into the following three classes according to the search needs.

Navigational queries. The purpose of this type of queries is to find a particular web page or web site the user has in mind. For example, for query "Binghamton University Computer Science," the goal is to reach the homepage of the Department of Computer Science of Binghamton University (i.e., `www.cs.binghamton.edu/).` Any page other than this homepage won't be considered as correct. Usually, users who submit such queries are aware of the web pages they seek, perhaps because they had visited the pages before. Link analysis and anchor text are considered to be critical for evaluating navigational queries.

Informational queries. The purpose of this type of queries is to find certain information from the Web, and it does not matter what pages contain the information. Queries for traditional text retrieval systems are mostly informational queries.

Transactional queries. The purpose of this type of queries is to find a web site to perform certain transactions interactively. Examples of transactions include shopping, downloading songs and movies, and signing up for certain services.

According to the analysis of 1,000 queries reported in (Broder 2002), 48 percent are informational queries, 30 percent are transactional queries and 20 percent are navigational queries. A user survey also reported in Broder (2002) suggested a lower percentage of informational queries and higher percentages of transactional and navigational queries than above.

Use of Tag Information

At present, most web pages are formatted in the HTML markup language. The HTML markup language contains a set of tags such as *title* and *header*. Most tags appear in pairs with one indicating the start and the other indicating the end. For example, in HTML, the starting and ending tags for title are <title> and </title>, respectively. Currently, in the context of search engine applications, tag information is primarily used to help computing the weights of index terms.

In the section on Text Retrieval, a method was introduced to compute the weight of a term in a document using its term frequency and its document frequency information. Tag information can also be used to influence the weight of a term. For example, authors of web pages frequently use emphatic fonts such as boldface, italics, and underscore to emphasize the importance of certain terms in a web page. Therefore, terms in emphatic fonts should be assigned higher weights. Conversely, terms in smaller fonts should be given lower weights. Other tags such as title and header can also be used to influence term weights. Many well-known search engines, including Google, have been known to assign higher weights to terms in titles.

A general approach to utilize tags to adjust term weights is as follows (Cutler et al. 1999). First, the set of HTML tags is partitioned into a number of subsets. For example, the title tag could be a subset by itself, all header tags (i.e., h1, ..., h6) could form a subset, all list tags (namely “ul” for unordered list, “ol” for ordered list and “dl” for descriptive list) could be grouped together, and all emphatic tags could form a subset and the rest of the tags can form yet another subset. Next, based on the partition of the tags, term occurrences in a page are partitioned into a number of classes, one class for each subset of tags. For example, all term occurrences appearing in the title form a class. In addition to these classes, two special classes can also be formed for each page P. The first contains term occurrences in plain text (i.e., with no tags), and the second contains the terms in the anchor text associated with the links that point to the page P. Let n be the number of classes formed. With these classes, for a given page, the term frequency of each term in the page can be represented as a *term frequency vector*: $tfv = (tf_1, ..., tf_n)$, where tf_i is the number of times the term appears in the i-th class, $i=1, ..., n$. Finally, different importance can be assigned to different classes. Let $civ = (civ_1, ..., civ_n)$ be the *class importance vector* such that civ_i is the importance assigned to the i-th class, $i=1, ..., n$. With vectors tfv and civ, the traditional term frequency weight formula can be extended into $tf_1*civ_1 + \ldots + tf_n*civ_n$. This formula takes both the frequencies of the term in different classes as well as the importance of different classes into consideration. Note that when the civ for the anchor term class is set 0 and all other civ's are set 1, $tf_1*civ_1 + \ldots + tf_n*civ_n$ becomes tf, the total frequency of the term in a page.

An interesting issue is how to find the optimal class importance vector that can yield the highest retrieval effectiveness. One method is to find an optimal or near optimal *civ* empirically based on a test dataset (Cutler et al. 1999). The main findings of this study were that the classes corresponding to the anchor texts and the emphatic tags are the most important (their *civ* values are the highest) to achieve high retrieval effectiveness.

Use of Linkage Information

There are several ways to make use of the linkage information between web pages to improve the retrieval quality of a search engine. One method is to use all anchor texts associated with the links that point to page B to represent the contents of B. Since these terms are chosen by human authors for the purpose of describing the contents of B, they are high quality terms for indicating B’s contents. Many search engines (e.g., Google) use anchor texts to index linked pages (e.g., page B). The most well-known uses of the linkage information in search engines treat links as votes to determine the importance of web pages. The PageRank method described below tries to find the overall importance of each web page regardless of the contents of the page.

PageRank Method

The Web can be viewed as a gigantic directed graph $G(V, E)$, where V is the set of pages (vertices) and E is the set of links (directed edges). Each page may have a number of outgoing edges (forwardlinks) and a number of incoming edges (backlinks). As mentioned at the beginning of the section on Search Engine Technology, when an author places a link in page A to point to page B, the author shows that she considers page B to be of some value. In other words, such a link can be viewed as a vote for page B. Each page may be pointed to by many other pages, and the corresponding links can be aggregated in some way to reflect the overall importance of the page. For a given page, *PageRank* is a measure of the relative importance of the page on the Web and this measure is computed based on the linkage information (Page et al. 1998). The following are the three main ideas behind the definition and computation of the PageRanks of web pages.

1. Pages that have more backlinks are likely to be more important. Intuitively, when a page has more backlinks, the page is considered to be of some value by more web page authors. In other words, the importance of a page should be reflected by the popularity of the page among all web page authors.
2. The importance of a page increases if it is pointed to by more important pages. Suppose page A and page B are pointed to by two sets of pages S_a and S_b, respectively, and the two sets have the same cardinality. If each page in S_a is more important than a corresponding page in S_b and they have the same number of outgoing pages (see the next item for the reason why the same number of outgoing pointers is needed), then page A is more important than page B. Intuitively, important pages are likely to be published by important authors or organizations, and the endorsement of these authors/organizations should have more weight in determining the importance of a page. To summarize, the importance of a page should be a weighted popularity measure of the page.
3. When a page has more forward links, the page has less influence over the importance of each of the linked pages. Item 2 above indicates that the importance of a page may be propagated to its child pages. If a page has multiple child pages, then these pages should share the importance of the parent page. In other words, if a page has more child pages, then it can only propagate a smaller fraction of its importance to each child page.

From the above discussion, it can be seen that it is natural to compute PageRanks of web pages in a recursive manner. The PageRank of a page u is defined more formally as follows. Let F_u denote the set of pages that are pointed by u and B_u denote the set of pages that point to u. For a set X, let $|X|$ denote the number of items in X. The PageRank of u, denoted $R(u)$, is defined by the formula below:

$$R(u) = \sum_{v \in B_u} \frac{R(v)}{|F_v|} \qquad (31.1)$$

Notice how the above formula incorporates the three ideas we discussed earlier. First, the sum reflects the idea that more backlinks can lead to a larger PageRank. Second, having $R(v)$ in the numerator indicates that the PageRank of u is increased more if page v is more important (i.e., has larger $R(v)$). Third, using $|F_v|$ in the denominator implies that the importance of a page is evenly divided and propagated to each of its child pages. Also notice that Formula (31.1) is recursive. The PageRanks of web pages can be computed as follows. First, an initial PageRank is assigned to all pages. Let N be the number of web pages. Then $1/N$ could be used as the initial PageRank of each page. Next, Formula (1) is applied to compute the PageRanks in a number of iterations. In each iteration, the PageRank of each page is computed using the PageRanks of its parent pages in the previous iteration. This process is repeated until the PageRanks of the pages converge within a given threshold. It should be noted that the actual PageRank formula is more complex than Formula (31.1) to ensure the convergence of PageRanks.

The PageRanks of web pages can be used to help retrieve better web pages in a search engine environment. In particular, the PageRank of a web page can be combined with other, possibly content-based, measures to indicate the overall relevance of the web page with respect to a given query. For example, suppose for a given query, the similarity of a page p with the query is $sim(p)$. From this similarity and the PageRank of p, $R(p)$, a new score of the page with respect to the query can be computed by $w*sim(p) + (1–w)*R(p)$, where $0<w<1$. This formula emphasizes not only how similar the page is to the query but also how important the page is. As a result, a search engine that employs such a formula would favor more important pages that have similar similarity scores. Utilizing PageRank in retrieval has the effect of countering content-based spamming. Some web page authors insert many popular but irrelevant words in a page to boost its chance of being retrieved by search engines. When PageRank is used, the search engine no longer relies on only words appearing in a page to determine its potential relevance. However, sophisticated link-spamming schemes have been used to influence the PageRanks of web pages.

Google is the first search engine to incorporate the PageRanks of web pages into its document ranking algorithm.

Use of User Profiles

Successful information finding on the Web is sensitive to the background interests of individual users. For example, for the query “apple,” a person with a history of retrieving documents in the computer area is more likely to be interested in information related to “Apple computer” while a person interested in food probably would like to see pages that consider apple as a fruit. Personalized search is an important method to improve the retrieval effectiveness of a search engine. The background information of each user (i.e., user profile) can be collected in a number of ways such as from query logs, bookmarks, and cookies.

Parts of user profiles can also be generated from implicit user feedback (Liu, Yu, and Meng 2004; Speretta and Gauch 2005). When a user submits a query to a search engine, the user may have some of the following behaviors or reactions regarding the returned web pages:

- Click some pages in certain order while ignoring others.
- Read some clicked pages for a longer time than some other clicked pages.

If a user spends a significant amount time on a page (before clicking another page), then the page could be considered as useful to the user. Information, such as significant terms, extracted from the queries submitted by a user and from the pages that are considered to be useful to the user can be used, together with possibly other information about the user such as bookmarks and cookies, to form a profile for the user. In general, a user may have multiple profiles corresponding to the different interests of the user, and these profiles may evolve with time.

User profiles may be utilized in a number of ways to improve the retrieval effectiveness of a search engine as discussed below.

User profiles can be used to help determine the meaning of a query term (Liu, Yu, and Meng 2004). If a user submits a query with a single term "bank" and the user has a profile on environment but no profile on finance, then it is likely that the current usage of this term is as in "river bank" rather than in "investment bank." Web users often submit very short queries to search engines (see the subsection on Web Search Queries) and for such short queries, correctly determining the meanings of query terms can help retrieve relevant web pages.

User profiles can be utilized to perform *query expansion* (Pitkow et al. 2002). When an appropriate profile can be identified to be closely related to a query, then terms in the profile may be added to the query such that a longer query can be processed. In text retrieval, it is known that longer queries tend to return better-matched documents because they are often more precise in describing users' information needs than short queries.

User profiles can be used to filter and/or re-rank initial results (Pitkow et al. 2002). When a user query is received by a search engine, a list of results based on the query can be obtained first. These results can then be compared with the profiles of the user to filter out web pages that are unlikely to be useful to this particular user and re-rank the remaining results based on their similarities with the profiles of the user.

User profiles of one user can be used to help find useful pages for another user. Part of a user profile may include what queries have been submitted by a user and what pages have been considered as useful for each query. When a user u_1 submits a new query q, it is possible to find another user u_2 such that the profiles of the two users are similar and user u_2 has submitted query q before. In this case, the search engine may rank highly the pages that were identified to be useful by u_2 for query q in the result for u_1. Furthermore, from the profiles of all users, it is possible to know how many users have considered a particular page to be useful (regardless of for what queries). Such information can be used to create a *recommender system* in the search engine environment. Essentially, if a page has been considered to be useful by many users for queries similar to a newly received query, then the page is likely to be useful to the new query and should be ranked high in the result. The DirectHit search engine (which was acquired by Ask.com in 1999) was built based on the principles of recommender systems.

Result Organization

Most search engines organize retrieved results in descending order of their estimated desirability with respect to a given query. The *desirability* of a page to a query could be approximated in many different ways such as the similarity of the page with the query, a combined measure including similarity and PageRank of the page, or the authority score of the page. A related issue is how to represent a *search result record*, which corresponds to a retrieved web page, in a result page. The search result records generated by most current search engines consist of primarily three pieces of information: the title of the web page, the URL of the web page, and a short text known as a *snippet*. Other information such as the publication time and the size of the web page may also be included in the search result records by some search engines. Sometimes, the URL is hidden behind the title, that is, the title is a clickable text with the URL encoded. The snippet is usually a short descriptive text extracted from the web page with the purpose of providing more hints about the content of the page so as to permit the user to make a more informed decision on whether to request the full web page. Often a text segment of a certain length that starts a sentence and contains the query term(s) is extracted from the web page as the snippet of the web page.

Some search engines organize their results into groups such that pages that have certain common features are placed into the same group. Such an organization of the results, when meaningful labels (annotations) are assigned to each group, can facilitate users for identifying useful pages from the returned results. This is especially useful when the number of pages returned for a query is large. The Vivisimo search engine (`www.vivisimo.com`) organizes returned results for each query into a hierarchy of groups. A study indicates that clustering/categorizing search results is effective in helping a user identify relevant results (Hearst and Pedersen 1996), especially when user queries are short. Short queries often result in diverse results because short queries can have different interpretations. When results are organized into multiple clusters, results corresponding to the same interpretation tend to fall in the same cluster. In this case, when clusters are appropriately annotated, finding relevant results becomes much easier.

There are also search engines that present the search results graphically so that the relationships between different results can be visualized. For example, the results of the Liveplasma music and movie search engine (`www.liveplasma.com`) are displayed as annotated icons that are grouped by some common features (say movies with the same lead actor) and are connected by different links (e.g., a link to the director of a movie).

Sponsored Search

Sponsored search is an online advertising mechanism that places advertisements (ads) on returned result pages. Sponsored search is a critical source of revenue for commercial search engines such as Google and Yahoo. To be effective, it is important for the ads placed on a result page to be relevant to the user query that retrieved the results because this would increase the chance that the user will be interested in the ads. There are three basic pricing models for this type of ads (Broder et al. 2007): they are pay-per-click or cost-per-click (the advertisers pay a certain amount of money for every click on the ad), pay-per-impression (the advertisers' pay is based on the number of times the ad is shown to users), and pay-per-action (the advertisers pay only when the ad leads to an action, e.g., a sale). It is possible to convert from one pricing model to another based on certain conversion rates (Fain and Pedersen 2006).

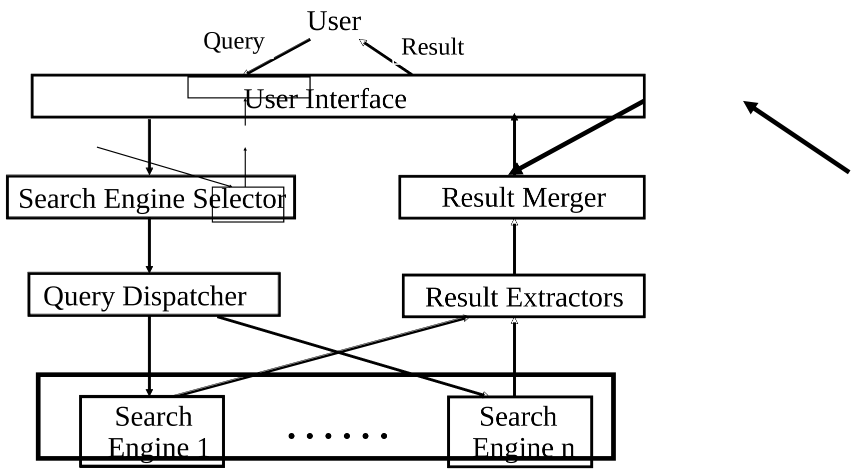
The most widely used ad placement method can be summarized as follows. First, the search engine operators (more generically called publishers) auction certain bid-phrases, and the advertisers bid for them based on a certain pricing model. Next, when a search engine receives a user query, it tries to rank nonsponsored (normal) results and sponsored results (i.e., the ads) simultaneously. To rank the ads, the search engine extracts the bid-phrases from the user query and computes a similarity score between the query and the description of each ad based on the extracted bid-phrases. Among the ads that have similar scores, they are ranked in descending order of the bids, that is, advertisers that are willing to pay higher prices for the bid-phrases in the query will have their ads ranked higher in the sponsored result section of the result page produced by the search engine.

In general, ads can be placed on generic web pages, not necessarily search result pages. This is known as *contextual advertising* (Broder et al. Riedel 2007).

METASEARCH ENGINE TECHNOLOGY

A metasearch engine provides a unified way to access multiple existing search engines. There are two types of metasearch engines, general-purpose metasearch engines and special-purpose metasearch engines. The former aims to search the entire (Surface) Web, while the latter focuses on searching information in a particular domain (e.g., news). Most of the current popular metasearch engines, such as Dogpile (www.dogpile.com), Mamma (www.mamma.com), Vivisimo (vivisimo.com) and MetaCrawler (www.metacrawler.com), are primarily general-purpose metasearch engines, even though some of them also support the search of information from a particular domain. While most general-purpose metasearch engines are created on top of a small number of (usually less than 10) general-purpose search engines, some special-purpose metasearch engines employ a large number of specialized search engines. For example, AllInOneNews (www.allinonenews.com) provides unified access to 1,800 news search engines.

One of the most significant benefits of metasearch engines is the ability to combine the coverage of multiple search engines. This can be viewed from two aspects. First, different search engines index different sets of web pages; this is true even for popular general-purpose search engines. A metasearch engine enables the search of the union of the web page sets of the component search engines of the metasearch engine. Second, different search engines often employ different document indexing and result ranking algorithms and as a result, they often return different sets of top results for the



Software Component Architecture

A reference software component architecture of a metasearch engine is illustrated in Figure 31.4. More details regarding each software component are:

Database selector: If the number of component search engines in a metasearch engine is very small, say less than five, it might be reasonable to send each user query submitted to the metasearch engine to all the component search engines. However, if the number is large, say in the tens or even hundreds, then sending each query to all component search engines will be an inefficient strategy because most component search engines will be useless with respect to any given query. Sending a query to useless search engines may cause serious inefficiencies. For example, transmitting the query to useless search engines from the metasearch engine and transmitting useless retrieved pages from these search engines to the metasearch engine would cause wasteful network traffic. Furthermore, the metasearch engine may be overwhelmed by the documents returned by these search engines. Therefore, it is important to send each user query to only potentially useful search engines for processing. The problem of identifying potentially useful search engines to invoke for a given query is known as the *database selection problem*. Several database selection techniques will be discussed in the subsection Database Selection.

Query dispatcher: After a component search engine has been selected to participate in the processing of a user query, the *query dispatcher* establishes a connection with the server of the search engine and passes the query to it. HTTP (HyperText Transfer Protocol) is used for the connection and data transfer (sending queries and receiving results). In general, different search engines may have different requirements on the HTTP request method such as the GET method or the POST method, and the query format such as the specific query box name. Therefore, the query dispatcher consists of many *connection programs*, one for each component search engine. In addition, the query sent to a particular search engine may or may not be the same as the one received by the metasearch engine. In other words, the original user query may be modified to a new query before being sent to a search engine. For vector space queries, query modification is usually not needed but two types of modifications are possible. First, some highly related terms may be added to a query. For example, in the AllInOneNews metasearch engine, synonyms and hyponyms whose meanings are close to the query terms are added to the query. Second, the relative weights of query terms in the original user query may be changed before the query is sent to a component search engine. The change could be accomplished by repeating some query terms an appropriate number of times as the weight of a term is usually an increasing function of its frequency. Such a modification of query term weights could be useful to influence the ranking of the retrieved web pages from the component search engine in a way as desired by the metasearch engine (Meng, Yu, and Liu 2002). Query dispatch will not be discussed further in this chapter.

Result extractors: After a user query is processed by a search engine, one or more result pages are returned by the search engine. A typical result page contains multiple (usually 10) search result records, each of which corresponds to a retrieved web page, and it typically contains the URL and title of the page plus a snippet of the page. Result pages are dynamically generated HTML documents and they often also contain content unrelated to the user query such as advertisements and information about the host web site. A program (a result extractor) is needed to extract the correct search result records from the result pages. This program is sometimes called an *extraction wrapper*. Since different search engines format their results differently, a separate *result extractor* or wrapper needs to be created for each component search engine.

Result extraction techniques will be discussed in the subsection on Result Record Extraction.

Result merger: After the results from selected component search engines are returned to the metasearch engine, the *result merger* combines the results into a single ranked list. The top *m* pages in the list are then returned to the user through the user interface, where *m* is the number of pages desired by the user. A good result merger should rank all returned pages in descending order of their desirability. Different methods for performing result merging will be discussed in the subsection Result Merging.

Database Selection

As we explained previously, when a metasearch engine receives a query from a user, the database selector is invoked to select component search engines that are likely to contain useful web pages for the query. To enable database selection, some characteristic information representing the contents of the document database of each search engine needs to be collected and made available to the database selector. The characteristic information about a database will be called the *representative* of the database in this article. Many database selection techniques have been proposed, and these techniques can be classified into the following three categories (Meng, Yu, and Liu 2002).

1. **Rough representative approaches:** In these approaches, the representative of a database contains only a few selected key words or paragraphs. Clearly, rough representatives can only provide a very general description about the contents of the databases. Consequently, database selection techniques based on rough representatives are not very accurate in estimating the true usefulness of each database with respect to a given query. Rough representatives are often manually generated.
2. **Statistical representative approaches:** Database representatives using these approaches have detailed statistical information about the document databases. Typically, statistics for each term in a database such as the *document frequency* of the term and the average weight of the term in all documents containing the term are collected. While detailed statistics allow more accurate estimation of database usefulness with respect to any user query, more effort is needed to collect them and more storage space is needed to store them.
3. **Learning-based approaches:** Learning-based approaches acquire knowledge regarding which databases are likely to return useful pages to what types of queries from past retrieval experiences. Such knowledge is then used to determine the usefulness of the databases for each new query. For these approaches, the representative of a database is simply the knowledge indicating the past performance of the database with respect to different queries.

Due to limited space of this chapter, for the rest of this section, we concentrate on the statistical representative approaches. A statistical representative of a database typically takes every distinct term in every page in the database into consideration. Usually, one or more pieces of statistical information for each term are kept in the representative. A large number of approaches based on statistical representatives have been proposed (Meng, Yu, and Liu 2002). In this chapter, we describe two of these approaches.

CORI Net Approach

CORI Net (Collection Retrieval Inference Network (Callan, Lu, and Croft 1995)) is a research system for retrieving documents from multiple document collections. Each collection corresponds to a database in a metasearch engine environment. Let $t_1, ..., t_n$ be all the distinct terms in all collections in the system. The representative of each collection *C* conceptually consists of a set of triplets (t_i, df_i, cf_i), $i=1, ..., n$, where cf_i is the *collection frequency* of term t_i (i.e., the number of collections that contain t_i) and df_i is the *document frequency* of t_i in *C*. If a particular term, say t_j, does not appear in *C*, then $df_j = 0$ for *C* and the triplet (t_j, df_j, cf_j) need not be kept. Note that the collection frequency of a term is a system-wide statistic, and only one *cf* needs to be kept for each term in the system for collection selection.

For a given query *q*, CORI Net ranks collections using a technique that was originally proposed to rank documents in the INQUERY document retrieval system (Callan, Croft, and Harding 1992). This technique is known as the *inference network* approach. When ranking collections, CORI Net conceptually treats each collection representative as a (super) document. Thus, the set of all representatives forms a collection of super-documents. Let *D* denote this collection of super-documents. Consider collection *C* of regular documents and a term *t* in *C*. The df_i of *t* can be treated as the *term frequency* of *t* in the super-document of *C*. Similarly, the cf_i of *t* can be treated as the *document frequency* of *t* in *D*. This means that we have the term frequency and document frequency of each term in each super-document. With the representative of each collection being treated like a super-document, the problem of ranking collections has been reduced to the problem of ranking (super) documents.

CORI Net employs an inference network-based probabilistic approach to rank super-documents for a given query. In this approach, the ranking score of a collection with respect to query *q* is an estimated belief that the collection contains documents relevant to the query. The belief is the combined probability that the collection contains useful documents due to each query term.

More information about the CORI Net approach can be found in Callan (2000).

Most Similar Document Approach

One useful measure for ranking databases is the global similarity of the most similar document in a database for a given query. The global similarity of a document is computed by the metasearch engine according to a global similarity function. It may incorporate other usefulness measures of a page such as the PageRank in addition to the usual similarity between a query and the page. Theoretically, documents should be retrieved in descending order of some global measure to achieve optimal retrieval, if such a measure truly indicates the degree of relevance. The global similarities of the most similar documents in all databases can be used to rank databases optimally for a given query for retrieving the *m* most similar documents across all databases. Suppose there are *M* local databases $D_1, D_2, \ldots, D_M$ and the *m* most similar documents from these databases are desired. Intuitively, it is desirable to order these databases in such a way that the first *k* databases collectively contain the *m* most similar documents and each of these *k* databases contains at least one of these documents for some integer *k*. The concept of an optimal order of databases for a given query was introduced in Yu, Liu, Meng, Wu and Rishe (2002): For a given query *q*, databases $D_1, D_2, \ldots, D_M$ are said to be optimally ranked in the order $[D_1, D_2, ..., D_M]$ if there exists an integer *k* such that $D_1, D_2, ..., D_k$ contain all of the *m* most similar documents and each D_i, $1 \leq i \leq k$, contains at least one of the *m* most similar documents.

Clearly, if the databases are optimally ranked for a query, then it is sufficient to search the first *k* databases only. Let *msim*(*q, D*) be the global similarity of the most similar document in database *D* with the query *q*. It can be shown (Yu et al. 2002) that if the databases are ranked in descending order of their *msim*(*q, D*)'s, then these databases are optimally ranked with respect to *q*. Thus, for each query, each database is characterized by precisely one quantity—the global similarity of the most similar document in that database, and these quantities should be used to rank databases. This theory cannot be applied as is because it is not practical to search all databases, find the global similarities of the most similar documents in these databases, and then rank them for each query. The solution to this problem is to estimate *msim*(*q, D*) for any database *D* using the representative of *D* stored in the metasearch engine. Readers are referred to Yu and colleagues (2002) for several methods for estimating *msim*(*q, D*).

Constructing Database Representatives

Statistical database selection methods depend on the availability of detailed statistical information about the terms in the document collection of each search engine. Cooperative search engines may provide desired statistical information about their document collection to the metasearch engine. For search engines that follow a certain standard, say the proposed STARTS standard (Gravano et al. 1997), the needed statistics may be obtained from the information that can be provided by these search engines. For uncooperative search engines that do not follow any standard, their representatives may have to be extracted from sampled documents (e.g., Callan, 2000; Ipeirotis and Gravano, 2002).

Result Record Extraction

As mentioned previously, a result page returned by a search engine usually also contains some unwanted information such as advertisements. It is important to correctly extract the search result records on each result page. Some search engines, especially the large ones like Google and Yahoo, provide APIs (application programming interface) or Web services interface, to allow applications to interact with their search engines. These interfaces make the extraction of the search result records quite easy. Unfortunately, most search engines do not provide such interfaces. Consequently, techniques that can extract the search result records from the returned search result pages directly are needed. Since different search engines produce result pages in different formats, a separate result extractor (extraction wrapper) needs to be generated for each search engine. A large number of data extraction techniques have been proposed, and a survey of some earlier extraction techniques can be found in Laender and colleagues (2002).

Wrappers may be generated manually, semiautomatically, and automatically. Note that essentially all result pages are HTML documents. Manual techniques require an experienced developer to manually analyze the HTML source files of several result pages from the same search engine and identify the patterns or rules for extracting search result records. Semiautomatic methods often require a human to manually identify the correct result records on multiple result pages; then the wrapper generation system induces the extraction rules based on the human input. Both manual and semiautomatic techniques involve significant manual effort. Search engines change their result display format from time to time, which may cause existing wrappers to become useless. Creating and maintaining wrappers manually and semiautomatically are inconvenient and expensive. Therefore, in recent years, automatic wrapper generation received a lot of attention. Below we introduce ViNTs (Zhao et al. 2005), an automatic wrapper generator that utilizes the **v**isual **i**nformation on result pages a**n**d the **t**ag **s**tructures of the HTML source documents to generate wrappers.

ViNTs take one or more sample result pages from a search engine as input and generate a wrapper for extracting the result records from new result pages from the same search engine as output. The sample result pages can be automatically generated by the system through submitting automatically generated sample queries to the search engine. For each input sample result page, its tag tree is built to analyze its tag structures, and it is rendered on a browser to extract its visual information. *Content-line* is the basic building block of the ViNTs approach. It is a group of characters that visually form a horizontal line on the rendered page. In ViNTs, eight types of content lines (such as link line (the line is a hyperlink), text line, blank line, and so on) are differentiated. Next, the content lines are grouped into *blocks* based on candidate *separators* that are content lines appearing multiple times. Each candidate separator that leads to similar blocks is kept, and these blocks are used to generate a candidate wrapper (a regular expression of tags that is capable of extracting these blocks correctly). The similarity of two blocks is computed based on certain visual and structural information of the blocks.

Note that a result page may have multiple sections, and more than one section may contain neatly arranged records. For example, one section may contain sponsored links, and another section may contain regular search results. If more than one section has neatly arranged records, then more than one candidate wrapper may be generated. Next, ViNTs selects one of the candidate wrappers as the wrapper for this web page using several heuristics, including the size of the section as determined by its records (the main section usually has the largest area compared to other sections), the location of the section (the main section is usually centrally located on the result page), and so on. Many search engines do not always display their search results in the same format. To increase the likelihood of capturing all varieties, multiple sample result pages from the same search engine should be used to generate the wrapper. For each sample page, the process described above is followed to generate a wrapper for the page. Then the wrappers for different sample pages are integrated into a single and more robust wrapper for the search engine.

Result Merging

Ideally, a metasearch engine should provide local system transparency to its users. From a user's point of view, such a transparency means that a metasearch search engine should behave like a regular search engine. That is, when a user submits a query, the user does not need to be aware that multiple search engines may be used to process this query, and when the user receives the search results from the metasearch engine, he should be hidden from the fact that the results are retrieved from multiple search engines. Result merging is a necessary task in providing the above transparency. When merging the results returned from multiple search engines into a single result list, pages in the merged result should be ranked in descending order of global similarities. However, the heterogeneities that exist among the component search engines and between the metasearch engine and the component search engine make result merging a challenging problem. Usually, pages returned from a component search engine are ranked based on these pages' local similarities. However, most component search engines do not make the local similarities of returned pages available to the user.

The challenge here is how to merge the pages returned from multiple component search engines into a single ranked list in a reasonable manner in the absence of local similarities. An additional complication is that some pages may be returned by different numbers of component search engines. For example, one page could be returned by one of the selected component search engines and another may be returned by all of them. The question is whether and how this should affect the ranking of these pages.

Note that when we say that a page is returned by a search engine, we really mean that the URL of the page is returned. One simple approach that can solve all of the above problems is to actually fetch/download all retrieved pages from their local servers and compute their global similarities in the metasearch engine. One metasearch engine that employs this approach for result merging is the Inquirus system (Lawrence and Lee Giles 1998). Inquirus ranks pages returned from component search engines based on analyzing the contents of downloaded pages, and it employs a ranking formula that combines similarity and proximity matches. The biggest drawback of this approach is its slow response time as fetching pages and analyzing them on the fly can be time-consuming.

Most result-merging methods perform merging without downloading the retrieved web pages, and they all attempt to utilize the information that is already available from the result pages returned from component search engines. In general, for a page *P* retrieved from a search engine *S*, the following information about *P* is often available on the result page returned from *S*: (1) the local rank of *P*, (2) the content information contained in the search result record for *P*, i.e., the title and snippet of *P*, and (3) the publication date/time of *P*. Most present-day search engines return a snippet for each retrieved page in addition to the title. Search engines that retrieve time-sensitive information, such as those for searching news articles, frequently provide the publication date/time of the returned results. In addition, if a metasearch engine performs database selection, then it also already knows the ranking score of each selected search engine with respect to the user query. Below we briefly introduce techniques that utilize the above information to perform result merging.

Use Local Ranks

Borda Count (Aslam and Montague 2001) is a voting-based result merging method that uses only the rank information to perform merging. In this method, the retrieved results are considered as the candidates and each component search engine as a voter. Suppose there are *n* candidates. For each voter, the top-ranked candidate is given *n* points, the second-top-ranked candidate is given *n*–1 points, and so on. For candidates that are not ranked by a voter (i.e., they are not retrieved by the corresponding search engine), the remaining points of the voter (each voter has a fixed number of points) will be divided evenly among them. The candidates are then ranked in descending order of the total points they receive. The basic Borda Count method can be improved by taking into consideration search engines' ranking scores. For example, the final score of a result can be obtained by multiplying the points it receives with its source search engine's ranking score.

Condorcet approach is another type of voting-based merging technique. It is based on pair-wise comparisons of the candidates. More specifically, if page A is ranked higher than page B by more search engines, then A should be ranked higher than B in the merged list. This type of merging requires much more computation than other methods. Some techniques on how to improve the efficiency are reported in Dwork et al. (2001) and Montague and Aslam (2002).

Voting-based methods were designed for the situation where the same set of candidates is to be ranked by the voters. However, in the metasearch engine context, different search engines usually have different sets of web pages, which make these methods less effective.

Use Title and Snippet

The search result records returned by most search engines contain the titles and snippets of the retrieved results. Often the snippets are generated specifically for the submitted user query, and they are intended to help the user determine which results are more likely to be relevant. As a consequence, the title and snippet of a result can provide good clues about whether the corresponding page is relevant. Several new result merging algorithms have been recently proposed to perform merging based on the titles and snippets of the retrieved results and we introduce some of them below.

A simple strategy is as follows (Rasolofo, Hawking, and Savoy, 2003; Lu et al. 2005). First, compute the similarity between the user query and the title and also compute the similarity between the query and the snippet of each result using some similarity function(s). Next, combine the two similarities, say by weighted sum, and use it as the global similarity between the result and the query. Finally, rank the results from all selected search engines in descending order of these global similarities. It is also possible to further adjust the global similarities by the search engines ranking scores. For example, in (Rasolofo, Hawking, and Savoy 2003), the global similarity of each result is adjusted by multiplying it with the relative deviation of its source search engine's ranking score to the mean of the ranking scores of all selected search engines.

Another strategy introduced in (Lu et al. 2005) tries to explore more features with respect to the occurrences of the query terms in the title and snippet of each result. This method works as follows. First, the returned result records are grouped based on the number of distinct query terms in their titles and snippets; the groups containing more distinct terms are ranked higher. Second, within each group, the result records are further divided into three subgroups based on the locations of the occurred distinct query terms; the subgroup with these terms in the title ranks highest, and then the subgroup with the distinct terms in the snippet, and finally the subgroup with the terms scattered in both title and snippet. Finally, within each subgroup, the result records that have higher total numbers of occurrences of the query terms are ranked higher. If two records have the same number of occurrences of the query terms, first the one with distinct query terms appearing in the same order and adjacently as they are in the query is ranked higher, and then, the one with smaller window size is ranked higher. If all the occurred distinct query terms are located in the title or the snippet, the window size is the smallest number of consecutive words in the title or the snippet that contains at least one occurrence of each occurred distinct query term; otherwise, the window size is infinite.

Other Considerations

For time-sensitive information such as news, the result merging algorithm should take the freshness of a result into consideration. In Rasolofo, Hawking, and Savoy (2003), among results that have the same similarity, the publication date is used to break ties, that is, more recent news articles are ranked higher. AllInOneNews also takes publication date/time of the results into the merging process.

It is possible that the same page is retrieved by multiple component search engines. Result merging in this situation is usually carried out in two steps. In the first step, similarities are computed for all results, regardless of whether they are returned by one or more search engines, using techniques discussed above. In the second step, for each page *p* that is returned by multiple search engines, the similarities of *p* due to multiple search engines are combined in a certain way to generate a final similarity for *p*. Many combination functions have been proposed and studied (Croft 2000), and some of these functions have been used in metasearch engines. For example, the *max* function was used in ProFusion (Gauch, Wang, and Gomez 1996) and the *sum* function was used in MetaCrawler (Selberg and Etzioni 1997). For result merging techniques that do not compute similarities for results, the number of search engines that retrieve a result can be used to break ties, that is, results that are retrieved from more search engines are preferred in a tie situation.

Conclusion

In the last decade, we have all witnessed the explosion of the Web. The Web has become the largest digital library used by hundreds of millions of people. Search engines and metasearch engines have become indispensable tools for Web users to find desired information.

While most Web users probably have used search engines and metasearch engines, few know the technologies behind these wonderful tools. This chapter provides an overview of these technologies. As can be seen from this chapter, Web-based search technology has its roots in text retrieval techniques, but it also has many unique features. Some efforts to compare the quality of different search engines have been reported (for example, see Hawking et al. 2001)). An interesting issue is how to evaluate and compare the effectiveness of different techniques. Since most search engines employ multiple techniques, it is difficult to isolate the effect of a particular technique on effectiveness even when the effectiveness of search engines can be obtained.

Web-based search is still a pretty young discipline, and it still has a lot of room to grow. A large number of researchers and developers are actively working on adding new capabilities to future search engines and metasearch engines to make them more effective and easier to use. Some of these new capabilities are:

Personalized and contextualized search: Current search systems return the same results for the same query, and they do not take the searcher's personal situation into consideration. Future search systems should incorporate each searcher's search context into the search process. A searcher's context may include the searcher's past search interests and educational background, the queries that immediately precede the present query, the location of the searcher, the time when the query is submitted, and so on.

Natural language queries and specific answers: Current search systems are designed to process keyword queries, and they return a long list of pages as the result. Future search systems should be able to understand natural language queries and provide more specific results whenever possible. As an example, a natural language query could be "Who is the lead actor in the movie *A Beautiful Mind*?" and a specific answer could be "Russell Crowe."

Advanced search: Current search systems can only return basic results (e.g., basic information from individual pages). Future search systems should support advanced search needs such as find the reputation of a car dealer, find the average number of publications for the faculty in a department, and summarize the reports about a major event. These queries require the analysis of the content of multiple pages and/or the identification of named entities as well as the analysis of the opinions and comments of different users/bloggers.

There are many other challenges in Web search technology. Henzinger, Motwani, and Silverstein (2002) described several of these challenges including the development of more effective techniques to defend against spam and new techniques to identify web pages with high quality content. It is safe to say that these challenges are still relevant today.

Acknowledgement: This work is supported in part by the following NSF grants: IIS-0414981, IIS-0414939 and CNS-0454298. We would like to thank anonymous reviewers for their very valuable suggestions that have helped improve this article significantly.

GLOSSARY

database selection

The process of selecting potentially useful data sources (databases, search engines, etc.) for each user query.

deep web

The set of web pages that are not publicly accessible or are dynamically generated by search engines from searching databases.

hub and authority

In a group of pages related to the same topic, a hub is a page that has links to important (authority) pages in the group, and an authority is a page that is linked from hub pages in the group.

metasearch engine

A web-based search tool that utilizes other search engines to retrieve information for its user.

PageRank

A measure of web page importance based on how web pages are linked to each other on the web.

result merging

The process of merging documents retrieved from multiple sources into a single ranked list.

search engine

A web-based tool that retrieves potentially useful results (web pages, products, etc.) for each user query.

Surface Web

The collection of web pages that is publicly and directly accessible without the need to go through a registration, login, or a search engine.

text retrieval

A discipline that studies techniques to retrieve relevant text documents from a document collection for each query.

vector space model

A popular text representation model where both documents and user queries are represented as vectors of terms with weights.

Web (World Wide Web)

Hyperlinked documents residing on networked computers, allowing users to navigate from one document to any linked document.

Web crawler

A computer program for recursively fetching web pages from the Web. The program is also referred to as Web spider and Web robot.

REFERENCES

Aslam, J., and M. Montague 2001. Models for metasearch. ACM SIGIR International Conference on Research and Development in Information Retrieval, New Orleans:276–284.

Baeza-Yates, R., and B. Ribeiro-Neto. 1999. *Modern information retrieval.* ACM Press/Addison-Wesley, New York.

Bergman, M. 2001. The deep web: Surfacing the hidden value. BrightPlanet. www.brightplanet.com/images/stories/pdf/deepwebwhitepaper.pdf (date of access: June 2008.

Broder, A. 2002. A taxonomy of web search. *ACM SIGIR Forum* 36, no. 2:3–10.

Broder, A., M. Fontoura, V. Josifovski, and L. Riedel. 2007. A semantic approach to contextual advertising. ACM SIGIR International Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands:559–566.

Callan, J. 2000. Distributed information retrieval. In *Advances in information retrieval: Recent research from the center for intelligent information retrieval*, ed. W. Bruce Croft:127–150. Kluwer Academic Publishers, Boston.

Callan, J., W. Croft, and S. Harding. 1992. The INQUERY retrieval system. Third DEXA Conference, Valencia, Spain:78–83.

Callan, J., Z. Lu, and W. Croft. 1995. Searching distributed collections with inference networks. ACM SIGIR International Conference on Research and Development in Information Retrieval, Seattle:21–28.

Chakrabarti, S., M. van den Bergand, and B. Dom. 1999. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks* 31:1623–1640.

ComScore. 2008. www.comscore.com/press/release.asp?press=2405.

Croft, W. 2000. Combining approaches to information retrieval. In *Advances in information retrieval: Recent research from the center for intelligent information retrieval*, ed. W. Bruce Croft:1–36. Kluwer Academic Publishers, Boston.

Cutler, M., H. Deng, S. Manicaan, and W. Meng. 1999. A new study on using HTML structures to improve retrieval. Eleventh IEEE Conference on Tools with Artificial Intelligence, Chicago:406–409.

Davison, B. D. 2000. Topical locality in the web. ACM SIGIR International Conference on Research and Development in Information Retrieval. Athens, Greece:272–279.

Dogpile. 2007. Different engines, Different results. A research study by Dogpile.com, April 2007. www.infospaceinc.com/onlineprod/Overlap-DifferentEnginesDifferentResults.pdf.

Dwork, C., R. Kumar, M. Naor, and D. Sivakumar. 2001. Rank aggregation methods for the web. World Wide Web Conference. Hong Kong:613–622.

Fain, D. C., and J. O. Pedersen. 2006. Sponsored search: A brief history. The Second Workshop on Sponsored Search Auctions, Ann Arbor, Michigan.

Frakes, W., and R. Baeza-Yates, 1992. *Information retrieval: Data structures and algorithms.* Prentice-Hall, Eaglewood Cliffs, New Jersey.

Gauch, S., G. Wang, and M. Gomez. 1996. ProFusion: Intelligent fusion from multiple, distributed search engines. *Journal of Universal Computer Science* 2, no. 9:637–649.

Gravano, L., C. Chang, H. Garcia-Molina, and A. Paepcke. 1997. Starts: Stanford proposal for internet meta-searching. ACM SIGMOD International Conference on Management of Data, Tucson, Arizona:207–218.

Hawking, D., N. Craswell, P. Bailey, and K. Griffiths. 2001. Measuring search engine quality. *Journal of Information Retrieval* 4, 1:33–59.

Hearst, M., and J. Pedersen. 1996. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. ACM SIGIR International Conference on Research and Development in Information Retrieval. Zurich, Switzerland:76–84.

Henzinger, M. R., R. Motwani, and C. Silverstein. 2002. Challenges in web search engines. *ACM SIGIR Forum* 36, no. 2:11–22.

Ipeirotis, P. G. and L. Gravano. 2002. Distributed search over the hidden web: Hierarchical database sampling and selection. International Conference on Very Large Data Bases, Hong Kong, 394–405.

Laender, A., B. Ribeiro-Neto, A. da Silva, and J. Teixeira. 2002. A brief survey of web data extraction tools. *ACM SIGMOD Record* 31, no. 2:84–93.

Lawrence, S., and C. Lee Giles. 1998. Inquirus, the NECi meta search engine. Seventh International World Wide Web conference. Brisbane, Australia:95–105.

Liu, F., C. Yu, and W. Meng. 2004. Personalized web search for improving retrieval effectiveness. *IEEE Transactions on Knowledge and Data Engineering* 16, no. 1:28–40.

Lu, Y., W. Meng, L. Shu, C. Yu, and K. Liu. 2005. Evaluation of result merging strategies for metasearch engines. International Conference on Web Information Systems Engineering. New York:53–66.

Meng, W., C. Yu, and K. Liu. 2002. Building efficient and effective metasearch engines. *ACM Computing Surveys* 34, no. 1:48–84.

Montague, M., and J. A. Aslam. 2002. Condorcet fusion for improved retrieval. ACM Conference on Information and Knowledge Management. McLean, VA:538–548.

Page, L., S. Brin, R. Motwani, and T. Winograd. 1998. The PageRank citation ranking: Bring order to the Web. Technical Report, Stanford University.

Pitkow, J., H. Schutze, T. Cass, R. Cooley, D. Turnbull, A. Edmonds, E. Adar, and T. Breuel. 2002. *Personalized Search. Communications of the ACM* 45, no. 9:50–55.

Ponte, J. M., and W. B. Croft. 1998. A language modeling approach to information retrieval. ACM SIGIR International Conference on Research and Development in Information Retrieval, Melbourne, Australia:275–281.

Rasolofo, Y., D. Hawking, and J. Savoy. 2003. Result merging strategies for a current news metasearcher. *Information Processing and Management* 39, no. 4:581–609.

Robertson, S. E., and S. Walker. 1999. Okapi/Keenbow at TREC-8. The Eighth Text Retrieval Conference, Gaithersburg, Maryland:151–161.

Salton, G. and McGill, M. J. 1983. Introduction to Modern Information Retrieval. McGraw-Hill.

Selberg, E., and O. Etzioni. 1997. The MetaCrawler architecture for resource aggregation on the Web. *IEEE Expert* 12, no. 1:8–14.

Silverstein, C., M. Henzinger, H. Marais, and M. Moriciz. 1999. Analysis of a very large web search engine query log. *ACM SIGIR Forum* 33, no. 1:6–12.

Speretta, M., and S. Gauch. 2005. Personalizing search based on user search histories. IEEE/WIC/ACM International Conference on Web Intelligence: 622–628.

Turtle, H., and J. Flood. 1995. Query evaluation: Strategies and optimizations. *Information Processing & Management* 31, no. 6:831–850.

Voorhees, E. M., and D. K. Harman. 2005. *TREC: Experiment and evaluation in information retrieval.* Cambridge: MIT Press.

Yu, C., K. Liu, W. Meng, Z. Wu, and N. Rishe. 2002. A methodology to retrieve text documents from multiple databases. *IEEE Transactions on Knowledge and Data Engineering* 14, no. 6:1347–1361.

Zhai, C., and J. Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems* 22, no. 2:179–214.

Zhao, H., W. Meng, Z. Wu, V. Raghavan, and C. Yu. 2005. Fully automatic wrapper generation for search engines. World Wide Web Conference, Chiba, Japan:66–75.

Zillman, P. M. 2009. Deep web research 2009.http://www.llrx.com/features/deepweb2009.htm