Web Search Technology

Clement Yu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607

Weiyi Meng
Department of Computer Science
SUNY at Binghamton
Binghamton, NY 13902

Outline

Glossary: Search engine, Web, text retrieval, PageRank, hub and authority, metasearch engine, database selection, collection fusion, result merging

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. 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. When a metasearch engine receives a query from a user, it invokes the underlying search engines to retrieve useful information for the user and combines the returned results from multiple search engines into a single ranked list before displaying it to the user. In this article, we describe the inner workings of search engines and metasearch engines. Specifically, we describe techniques that are used to build search engines and metasearch engines.

1. 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. 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 indexable. Each such page has a logical address called Uniform Resource Locator or URL. It was estimated that the Surface Web contained about 2 billion Web pages in 2000 (Bergman, 2000). The Hidden Web contains Web pages that are not publicly indexable. 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 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 Hidden Web. A recent study estimated the size of the Hidden Web to be about 500 billion pages (Bergman, 2000).

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 a database system through a Web interface. Database-driven search engines

are mostly employed for e-commerce applications such as buying cars or books. This article concentrates on document-driven search engines only. When a user submits a query, which usually 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) from the set of Web pages covered by the search engine. Usually, 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. Google, AltaVista, Lycos and HotBot are some of the most popular document-driven search engines on the Web. The Deep Web is usually accessed through Deep Web search engines (like the publisher's search engine we mentioned earlier). 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 called the index database of the search engine. So the second issue is how to construct such an index database. Another issue is how to use the index database to determine if a Web page matches well with a query. These issues will be discussed in Sections 2 and 3 of this article. More specifically, in Section 2, we will provide an overview of some basic concepts on text retrieval, including how to build an index database for a given document collection and how to measure the closeness of a document to a query; in Section 3, 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 and what are the techniques to organize the results of a query.

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 2 billion Web pages. But the entire Web is believed to have more than 500 billion pages. 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*. A metasearch engine can be considered as a system that supports unified access to multiple existing search engines. Combining many deep Web search engines can be an effective way to enable the search of a large portion of the deep Web. A metasearch engine does not maintain its own collection of Web pages but it may maintain information about its underlying 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 underlying search engines; when the metasearch engine receives the Web pages returned from its underlying search engines, it merges these pages into a single ranked list and display them to the user. Techniques that can be used to build efficient and effective metasearch engines are reviewed in Section 4 of this article. In particular, we will review techniques for selecting the right search engines to invoke for a given query and techniques for merging results returned from multiple search engines.

This article concentrates on the techniques that are used to build search engines and metasearch engines. Another article in this Encyclopedia covers Web searching related issues from the users' and Web page authors' points of views (Wisman, 2003).

2. 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. Documents are typically preprocessed and represented in a format that facilitates efficient and accurate retrieval. In this section, we provide a brief overview of some basic concepts in classical text retrieval.

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 semantic information. These words are called *stop words* and are usually not used for document representation. The remaining words are content words and can be used to represent the document. Variations of the same word may be mapped to the same term. For example, the words "beauty", "beautiful" and "beautify" can be denoted by the term "beaut". This can be achieved by a *stemming program*, which removes suffixes or replaces them by other characters. After removing stop words and stemming, each document can be logically represented by a vector of n terms (Salton and McGill, 1983), where n is the total number of distinct terms in the set of all documents in a document collection.

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 of the entries in the vector will be zero because most terms do not appear in any given 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 having 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 documents having the term from documents not having it. 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, i.e., *tfw * idfw*.

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

After all documents and a query have been represented as vectors, the query vector can be matched against document vectors so that well matched documents can be identified and retrieved. Usually, a *similarity function* is used to measure the degree of closeness between two vectors. 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. Longer documents have more terms than shorter documents, and as a result, 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, 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 (Salton and McGill, 1983) between two vectors is really 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°).

Computing the similarity between a query and every document directly is inefficient because many documents do not have any term in common with a given query and computing the similarity or these documents is a waste of effort. To improve the efficiency, an *inverted file index* is created in advance. For each term t_i, an inverted list of the format $[(D_{i1}, w_{i1}), ..., (D_{ik}, w_{ik})]$ is generated and stored, where D_{ij} is the identifier of a document containing t_i and w_{ij} is the weight of t_i in D_{ij}, $1 \le j \le k$. In addition, a *hash table* is created to locate for each given term the storage

location of the inverted file 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. 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.

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:

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

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

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.

An ideal text retrieval system should retrieve exactly the set of relevant documents for each query. In other words, a perfect system has recall = 1 and precision = 1 at the same time. In practice, perfect performance is not achievable due to many reasons. For example, a user's needs may be incorrectly or imprecisely specified by the query used and the representation of documents and queries as vectors does not capture their contents completely.

3. SEARCH ENGINE TECHNOLOGY

A Web search engine is essentially a Web-based text retrieval system for Web pages. However, the Web environment has some special characteristics that make building 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.

1. Web pages are stored at numerous autonomous Web servers. A program known as *Web robot* or *Web spider* is needed to gather them so that they can be processed for later search. Web robots are described in Section 3.1.
2. Web pages are highly tagged documents, that is, tags that control the presentations 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 but XML (eXtensible Markup Language) documents are making their ways into the Web. 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. In Section 3.2, we discuss some techniques that utilize these tags to improve retrieval effectiveness.
3. 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. Second, the author of page A considers page B to be of some value. Third, the set of words associated with the link, called *anchor terms* of the link, usually provides a short description of page B. In Section 3.3, several methods for utilizing link Information among Web pages to improve the search engine retrieval performance will be described.

Another special characteristic of the Web is its size and popularity. A large search engine can index billions of pages and needs to process millions of queries a day. For example, the Google search engine has indexed over 2 billion pages and processes over 27 million queries daily. To accommodate the high computation demand, a large search engine often uses numerous computers with large memory and cache devices, in addition to employing efficient query processing techniques. For example, the inverted file lists for different terms can be stored at different computers. When a multi-term query is received, all computers containing the inverted file list of at least one query term can participate in the evaluation of the query in parallel.

3.1 Web Robot

A Web robot (also known as *spider* and *crawler*) is a program for fetching Web pages from remote Web servers. Web robots are widely used to build the Web page collection for a search engine.

The main idea behind the robot 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 queue. 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 queue 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 queue.

One of the more challenging problems when implementing a Web robot is to identify all (new) URLs from a Web page. This requires the identification of all possible HTML tags and tag attributes that may hold URLs. While most URLs appear in the anchor tag (e.g., ...), URLs may also appear in other tags. For example, a URL may appear in the option tag as in <option value="URL" ...> ... </option>, in the area tag (map) as in <area href="URL" ...> ... </area>, or in the frame tag as in <frame src="URL" ...> ... </frame>. Frequently a URL appearing in a Web page P does not contain the full path needed to locate the corresponding Web page from a browser. Instead, a partial or relative path is used and a full path must be constructed using the relative path and a base path associated with P. When building a Web robot, it is important to be considerate to remote servers from which Web pages are fetched. *Rapid fires* (fetching a large number of Web pages from the same server in a short period of time) may overwhelm a server. A well-designed Web robot should control the pace of fetching multiple pages from the same server.

3.2 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 Section 2, 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 such as AltaVista and HotBot 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, Deng, Manicaan and Meng, 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 be formed for each page P. The first contains term occurrences in plain text (i.e., with no tags) and the second contains anchor terms 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 experimentally based on a test bed (Cutler et al, 1999). The test bed contains a Web page collection and a set of queries; for each query, the set of relevant pages is identified. Then different civ's can be tested based on the test bed and the civ that yields the highest retrieval effectiveness can be considered as an optimal civ. In practice, the number of different civ's may be too large to permit all civ's to be tested. Instead, heuristic algorithms such as genetic algorithm may be employed to find an optimal or near optimal civ efficiently.

3.3 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 terms* associated with 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 of high quality terms for indicating the contents of B. Many search engines (e.g., Google) use anchor terms 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. One method (the PageRank method) tries to find the overall importance of each Web page regardless of the contents of the page and another method (the Hub-Authority method) tries to identify pages that provide good content pages (authoritative pages) with respect to a given topic as well as good reference pages (hub pages) to these good content pages. These two methods are described below.

3.3.1 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 (forward links) and a number of incoming edges (backlinks). As mentioned at the beginning of Section 3, when an author places a link in page A to point to page B, the author shows that he/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, Brin, Motwani and Winograd, 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. For example, the homepage of Yahoo! is probably one of the most linked pages on the Web and it is certainly one of the most important pages on the web.
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 the 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. As an example, if a page is pointed to by the homepage of Yahoo!, then the page is likely to be important. 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 the PageRanks of Web pages need to be computed 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:

$$R(u) = \sum_{v \in B_u} \frac{R(v)}{|F_v|} \quad (1)$$

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 (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.

The PageRanks of Web pages can be used to help retrieve better Web pages in a search engine environment. In particular, the PageRanks of Web pages can be combined with other, say content-based, measures to indicate the overall relevance of Web pages 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 to rank Web pages tends to favor more important pages that have similar similarities. Utilizing

PageRank in retrieval has the anti-spamming effect. 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.

One of the most distinctive features of the Google search engine from other Web search engines is its incorporation of the PageRanks of Web pages, together with other factors such as similarities and proximity information, into the Web page selection process when processing a user query.

3.3.2 Hub-and-Authority Method. PageRank measures the global relative importance of Web pages but the importance is topic independent. Although the global importance is very useful, we often also need topic specific importance. For example, a movie lover is more likely to be interested in important movie pages. A topic specific important page can be considered to be an authoritative page with respect to a given topic. The question here is how to measure and find authoritative pages for a given topic. Kleinberg (1998) proposed to use authority scores to measure the importance of a Web page with respect to a given topic. He also suggested a procedure for finding topic specific important pages. The main ideas of this approach are reviewed here.

According to Kleinberg (1998), Web pages may be conceptually classified into two types: *Authoritative Pages* and *Hub pages*. The former contain informative contents on some topic and the latter has links to authoritative pages. It is observed that for a given topic, a good authoritative page is often pointed to by many good hub pages and a good hub page often has links to good authoritative pages. Good authoritative pages and good hub pages related to the same topic often reinforce each other. It is possible for the same page to be both a good authoritative page and a good hub page on the same topic.

In the search engine context, a topic can be defined by a user query submitted to a search engine. The procedure proposed in (Kleinberg, 1998) for finding good authoritative and hub pages can be summarized into the following three steps. Let *q* be a user submitted query.

1. Process the query by submitting it to a regular (similarity-based) search engine. Let *S* denote the set of documents returned by the search engine. The set *S* is called the *root set*. In order to find good authoritative pages with respect to a query, the size of *S* should be reasonably large, say in the hundreds.
2. Expand the root set *S* into a larger set *T* called the *base set*. *T* is expanded from *S* by including any page that points to a page in *S* or is pointed to by a page in *S*. In other words, *T* contains all pages in *S* as well as all the parent pages and child pages of the pages in *S*. In order to compute *T* from *S* quickly, the search engine can save

the link structure of the Web in advance. While a Web page typically has a small number of child pages, it may have an extremely large number of parent pages. For example, the homepage of Yahoo! may be linked by millions of pages. In order to limit the size of the base set, a threshold, say 20, can be used such that at most 20 parent pages of each page in *S* are included in *T*. Similar restriction may also be applied to the inclusion of child pages. Some heuristics may be applied to choose parent pages for inclusion. One heuristic is to select those parent pages whose anchor terms associated with the links contain terms in the query. The base set for a given query typically contains thousands of pages related to the query. The last step is to identify the most authoritative pages with respect to the query from the base set.

3. Compute the authority score and hub score of each page in the base set *T*. For a given page *p*, let $a(p)$ and $h(p)$ be the authority and hub scores of *p*, respectively. Initially, $a(p) = h(p) = 1$ for each page *p*. For pages *p* and *q*, let (p, q) denote a link from *p* to *q*. The computation is carried out in a number of iterations. In each iteration, two basic operations and two normalizations are executed for each page. The two operations are called Operation I and Operation O.

    - Operation I: Update each $a(p)$ to be the sum of the current hub scores of Web pages in the base set that point to *p*. More precisely, $a(p) = \sum_{q:(q,p)\in E} h(q)$, where *E* is the set of links between pages in the base set *T*.
    - Operation O: Update each $h(p)$ to be the sum of the current authority scores of Web pages in the base set that are linked from *p*. More precisely, $h(p) = \sum_{q:(p,q)\in E} a(q)$.
    - After all authority and hub scores have been updated in the current iteration, normalize each authority score and each hub score as follows:

$$a(p) = \frac{a(p)}{\sqrt{\sum_{q\in T}[a(q)]^2}}, \quad h(p) = \frac{h(p)}{\sqrt{\sum_{q\in T}[h(q)]^2}}$$

The above computation process is repeated until the scores converge.

After all scores are computed, the Web pages in the base set are sorted in descending authority score and the pages with top authority scores are displayed to the user.

The above method for calculating authority and hub scores treats all links the same. In reality, some links may be more useful than other links in identifying authoritative pages with respect to a given topic. Two cases are considered below.

1. Two types of links can be distinguished based on whether or not the two pages related to a link are from the same domain, where the domain name of a page is the first level string of its URL (i.e., the part before the first "/"). A link between pages with different domain names is called a *transverse link* and a link between pages with the same domain name is called an *intrinsic link*. It can be argued that transverse links are more significant than intrinsic links for two reasons. First, intrinsic links are sometimes used for presentation purposes, i.e., breaking a large document into smaller linked pieces to facilitate browsing. Second, intrinsic links can be considered to be self-referencing whose significance should be lower than references made by others. One way to handle intrinsic links is to simply discard them (Kleinberg, 1998). Another method is to give a lower weight to intrinsic links (Chakrabarti et al, 1999).
2. Recall that a link often has a set of associated anchor terms. It can be argued that if the anchor terms of a link contain terms in the query (topic), then the likelihood that the page pointed to by the link is an authoritative page with respect to the topic is increased. In general, a vicinity of a link can be defined to include terms within certain distance (say 50 characters) on both sides on the link. Then the weight associated with a link can be defined as an increasing function of the number of terms in the vicinity of the link that appear in the query and this weight can be incorporated into the computation of authority and hub scores (Chakrabarti et al, 1998).

3.4 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 through the use of bookmarks and cookies.

Parts of user profiles can also be generated from implicit user feedback. 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:

1. Click some pages in certain order while ignore others.

2. Read some clicked pages for a longer time than some other clicked pages.
3. Save/print certain pages.

If a user saves/prints a page or spends a significant amount time on a page (before clicking another page), then the page can 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.

1. User profiles can be used to help determine the meaning of a query term. 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 like in “river bank” rather than in “investment bank”. Web users often submit short queries to search engines. A typical query has about 2.3 terms and about 30% of all queries have just one term. For these short queries, correctly determining the meanings of query terms can help retrieve relevant Web pages.
2. User profiles can be utilized to perform *query expansion*. 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.
3. User profiles can be used to filter initial results. When a user query is received by a search engine, a list of results based on only the query but not any profile of the user can be obtained first. These results can then be compared with the profiles of the user to help identify Web pages that are more likely to be useful to this particular user.
4. 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 (www.directhit.com) has incorporated the principles of recommender systems.

3.5 Result Organization

Most search engines organize retrieved results (including URLs and some short descriptions known as *snippets*) in descending order of their estimated desirabilities 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 rank of the page, or the authority score of the page. Some search engines, such as FirstGov (www.firstgov.gov) and Northern Light (www.northernlight.com) also provide the estimated desirabilities of returned pages while some, such as AltaVista and Google, do not provide such information.

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) and the DynaCat system (Pratt, Hearst and Fagan, 1999) organize retuned results for each query into a hierarchy of groups. In general, the issues that need to be addressed when implementing an online result-clustering algorithm include: (1) What information (titles, URLs, snippets versus the full documents) should be used to perform the clustering? While more information may improve the quality of the clusters, using too much information may cause long delays for users. (2) How to cluster? A large number of text clustering algorithms exist. (3) How to come up with labels that are meaningful descriptions of each group? (4) How to organize the groups? They could be linearly ordered or hierarchically ordered. In the former case, what should be the linear order? In the latter case, how to generate the hierarchy? (5) How to order pages in each cluster? Many of the issues are still being actively researched.

Study indicates that clustering/categorizing search results is effective in helping 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.

4. METASEARCH ENGINE TECHNOLOGY

A metasearch engine provides a way to access multiple existing search engines with ease. One of the most significant benefits of metasearch engines is its ability to combine the coverage of many search engines. In particular, by employing many search engines for the deep Web, a metasearch engine can be an effective tool to quickly reach a large portion of the deep Web. In this section, we provide an overview of the metasearch engine technology. In Section 4.1, a reference software component architecture of a metasearch engine is introduced. In Section 4.2, techniques that identify what search engines are likely to contain useful results for a given query are discussed. The set of Web pages that can be searched by a search engine is the Web page database of the search engine. Therefore, the search engine selection problem is also known as the database selection problem. In Section 4.3, methods that determine what pages from selected search engines should be retrieved and how the results from different search engines should be merged are reviewed.

Figure 1 about here

4.1 Software Component Architecture

A reference software component architecture of a metasearch engine (Meng, Yu and Liu, 2002) is illustrated in Figure 1. The numbers associated with the arrows indicate the sequence of steps for processing a query. More details regarding each software component are described below.

1. **Database selector**: If the number of local search engines in a metasearch engine is small, then it would be reasonable to send each user query submitted to the metasearch engine to all the local search engines. However, if the number is large, say in the hundreds, then sending each query to all local search engines will be an inefficient strategy because most local search engines will be useless with respect to any given query. As an example, suppose only the 10 best matched pages are needed for a query. Clearly, the 10 desired pages are contained in no more than 10 search engines. This means that if there are 500 local search engines, then 490 of them are useless with respect to this 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. As another example, when a query is evaluated at useless search engines, system resources at these local systems would be wasted. 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*. The software component *database selector* is responsible for performing database selection. Selected database selection techniques will be discussed in Section 4.2.

2. **Collection fusion**: When searching from multiple document databases, *collection fusion* is a method for providing transparency to multiple databases. In the metasearch engine context, a collection fusion method determines what Web pages should be retrieved from each selected search engine and how the retrieved Web pages from multiple search engines should be merged into a single result list. In other words, collection fusion consists of a document selection module (*document selector*) and a result merge module (*result merger*). More details about these two modules are provided below.

   Document selector: For each search engine selected by the *database selector*, *document selector* determines what pages to retrieve from the document database of the search engine. The objective is to retrieve, from each selected local search engine, as many potentially useful pages as possible, and as few useless pages as possible. When more useless pages are returned from a search engine, greater effort would be needed by the metasearch engine to identify potentially useful ones.

   Result merger: After the results from selected local 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 desirabilities.

   Result extractor: One technical issue related to result merging is result extraction. When search results are returned by a search engine, they are grouped into one or more result pages, which contain the URLs and possibly some snippets of retrieved Web pages. Each result page is a dynamically generated HTML file. Usually, in addition to the URLs of retrieved pages, a result page also contains URLs unrelated to the user query. These unrelated URLs include URLs for advertisement pages and service pages. Therefore, the URLs of retrieved pages need to be correctly extracted from the HTML file of each result page. Since different search

engines use different ways to organize their result, a separate *result extractor* needs to be created for each local search engine. Result extraction will not be discussed further in this article.

Different methods for performing document selection and result merging will be discussed in Section 4.3.

3. **Query dispatcher**. After a local 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 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 (wrappers), one for each local 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. Two possible types of modifications are as follows. First, the relative weights of query terms in the original user query may be changed before the query is sent to a local 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 on query term weights could be useful to influence the ranking of the retrieved Web pages from the local search engine in a way as desired by the metasearch engine (Meng et al, 2002). Second, the number of pages to be retrieved from a local search engine may be different from that desired by the user. For example, suppose as part of a query, a user of the metasearch engine indicates that m Web pages are desired. The document selector may decide that k pages should be retrieved from a particular local search engine. In this case, the number k, usually different from m, should be part of the modified query to be sent to the local search engine. Note that not all search engines on the Web supports the specification of the desired number of pages by a user. For these search engines, the second type of query modification is not possible. Query dispatch will not be discussed further in this article.

4.2 Database Selection

As we explained in Section 4.1, when a metasearch engine receives a query from a user, the database selector is invoked to select local 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 et al, 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 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 having 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*: As the databases of different local search engines are different, they are not equally useful for a given query. Learning-based approaches learn the 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 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.

The main appeal of rough representative approaches is that the representatives can be obtained relatively easily, and they require little storage space. If all local search engines in a metasearch engine are highly specialized with diversified topics such that the contents of their databases can be easily summarized and differentiated, then these approaches may work reasonably well. On the other hand, it is unlikely that the short representative of a database can summarize the contents of the database sufficiently comprehensively, especially when the database contains Web pages of diverse topics. Therefore, these approaches can easily miss potentially useful databases for a query when performing database selection. A widely used method to alleviate this problem is to involve users in the database selection process. For example, in ALIWEB (Koster, 1994) and WAIS (Kahle and Medlar, 1991), users will make the final decision on which databases to use based on the preliminary selections made by the database

selector. In another system that employs rough database representatives, Search Broker (Manber and Bigot, 1997), user queries are required to contain the subject areas for the queries. Users often do not know all local search engines well. As a result, their contribution in database selection is limited. In general, rough representative approaches are inadequate for large-scale metasearch engines. Rough representative approaches will not be discussed further in this article. For the rest of this section, we concentrate on the other two types of approaches.

4.2.1 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. As a result, database selection techniques using statistical representatives are more likely to be able to detect the existence of individual potentially useful pages in a database for any given query. A large number of approaches based on statistical representatives have been proposed (Meng et al, 2002). In this article, 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) needs not to be kept. Note that the collection frequency of a term is a system wide statistics 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. At this point, any term frequency and document frequency based term weighting scheme may be utilized to compute the weight of each term in each super document. As a result, all super documents can be represented as vectors of terms with weights. Furthermore, any vector based similarity function such as those discussed in Section 2 may be used to compute the similarities between a given query and any super document, and these similarities can be used as the ranking scores of collections for the query.

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 to the query. The belief is the combined probability that the collection contains useful documents due to each query term. Suppose query q contains k terms t_1, ..., t_k. Let N be the number of collection under consideration. Let df_{ij} be the document frequency of the j-th term in the i-th collection C_i and cf_j be the collection frequency of the j-th term. The belief that C_i contains useful documents due to the j-th query term is estimated by:

$$p(t_j \mid C_i) = a_1 + (1 - a_1) \cdot T_{ij} \cdot I_j, \text{ where } T_{ij} = a_2 + (1 - a_2) \cdot \frac{df_{ij}}{df_{ij} + K}, \quad I_j = \log\left(\frac{N + 0.5}{cf_j}\right) \Big/ \log(N + 1.0)$$

In the above, T_{ij} is a formula employed by CORI Net for computing the term frequency weight of the j-th term in the super document of C_i, I_j is for computing the inverse document frequency weight of the j-th term, a_1 and a_2 are constants and K is a parameter related to the size of collection C_i. Appropriate values for a_1, a_2 and K can be determined empirically by experiments (Callan et al, 1995). Let $p(q|t_j)$ denote the belief that term t_j represents query q. One way to estimate $p(q|t_j)$ is to use the weight of t_j in q. Finally, the belief that collection C_i contains documents useful to query q can be estimated to be:

$$r_i = p(q \mid C_i) = \sum_{j=1}^{k} p(q \mid t_j) \cdot p(t_j \mid C_i)$$

A nice feature of the CORI Net approach is that the same method can be used to rank documents for a query as well as to rank collections for a query. 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 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 following definition introduces the concept of an optimal database order for a given query (Yu, Meng, Liu, Wu and Rishe, 1999).

Definition. 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 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, 1999) that if databases are ranked in descending order of their $msim(q, D)$'s, then these databases are optimally ranked with respect to q. 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.

One method to estimate $msim(q, D)$ is as follows (Yu et al, 1999). Two types of representatives are used. In the global representative for all local databases, the global inverse document frequency weight ($gidf_i$) is kept for each distinct term t_i. There is a local representative for each local database D. For each distinct term t_i in D, two quantities, mnw_i and anw_i, are kept, where mnw_i and anw_i are the *maximum normalized weight* and the *average normalized weight* of term t_i, respectively. If d_i is the weight of t_i in a document d, then $d_i / |d|$ is the normalized weight of t_i in d, where $|d|$ is the length of d. For term t_i in database D, mnw_i and anw_i are defined to be the maximum and the average of the normalized weights of t_i in all documents in D, respectively. Let $(q_1, ..., q_k)$ be the vector of query q. Then $msim(q, D)$ can be estimated by the following formula.

$$msim(q, D) = \max_{1 \le i \le k} \left\{ q_i * gidf_i * mnw_i + \sum_{j=1, j \ne i}^{k} q_j * gidf_j * anw_j \right\} / |q| \quad (2)$$

Intuitively, if document *d* is the most similar document in a database *D*, then one of the terms in *d*, say t_i, is likely to have the maximum normalized weight of t_i in all documents in *D*, while each of the other query terms is expected to take the average normalized weight of the corresponding term. Since any one of the *k* query terms in *d* could have the maximum normalized weight, the maximum is taken over all terms t_i. Finally, normalization by the query length, $|q|$, yields a value less than or equal to 1. As $|q|$ is common to all *msim*(*q, D*)'s, it can be dropped from the above formula without affecting the ranking of databases.

It was observed in (Wu, Meng, Yu and Li, 2001) that for a given term, its maximum normalized weight is typically two or more orders of magnitude larger than its average normalized weight. This is mainly because the average normalized weight of a term is computed over all documents, including those that do not contain the term. Meanwhile, it can be observed that for most user queries, all query terms have the same *tf* weight, that is, $q_i = q_j$, $i \neq j$. This is because in a typical query, each term appears the same number of times, namely once. From these two observations, we can see that in Formula (2), for a typical query, $gidf_i * mnw_i$ is likely to dominate $\sum_{j=1, j \ne i}^{k} gidf_j * anw_j$, especially when the number of terms, *k*, in the query is small (according to Kirsch (1998), the number of terms in a search engine query is 2.3 on the average). This means that the rank of database *D* with respect to a given query *q* is largely determined by $\max_{1 \le i \le k} \{q_i * gidf_i * mnw_i\}$. This leads to the following more scalable formula for ranking databases (Wu et al, 2001): $\max_{1 \le i \le k} \{q_i * am_i\}$, where $am_i = gidf_i * mnw_i$ is the *adjusted maximum normalized weight* of term t_i in *D*. This formula requires just one quantity, namely am_i, to be kept in the database representative for each distinct term in the database.

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 uncooperative search engines that follow a certain standard, say the proposed STARTS standard (Gravano, Chang, Garcia-Molina and Paepcke, 1997), the needed statistics may be obtained from the information that can be provided by these search engines such as the document frequency and the average document term weight of any query term. For uncooperative search engines

that do not follow any standard, their representatives may have to be extracted from sampled documents (e.g., Callan (2000), Callan, Connell and Du (1999)).

4.2.2 Learning-based Approaches. The usefulness of a database for different kinds of queries can be learned by examining the returned documents for past queries. The learned knowledge can then be used to select potentially useful databases to search for new queries. This is the idea behind all learning-based database selection techniques. The learning may be conducted in a number of ways. For example, carefully chosen training queries can be used. Ideally, training queries should reflect real user queries and cover the diversified contents of all local databases. Learning based on training queries is usually conducted offline. The knowledge obtained from training queries may become outdated due to the changes of database contents and query patterns. Another way to carry out learning is to use real user queries. In this case, the metasearch engine keeps track of user queries, and for each query, the documents that are implied by the user to be useful (e.g., the user spends much time with a page), and finally, from which database each implied useful document is retrieved. When real user queries are used for learning, the knowledge base can be updated continuously. Clearly, it may take a while for the metasearch engine to accumulate sufficient knowledge to enable meaningful database selection. It is possible for a metasearch engine to utilize both training queries and real user queries for learning. In this case, training queries are used to obtain the initial knowledge base while real user queries are used to update the knowledge base. Two learning based database selection methods are reviewed below. The first method uses only real user queries while the second method uses both training queries and real user queries.

SavvySearch approach

SavvySearch (www.search.com) conducts learning based on real user queries and the reactions of real users to retrieved pages. In SavvySearch (Dreilinger and Howe, 1997) the ranking score of a local search engine for a given query is computed based on the past retrieval performance related to queries that contain terms in the new query. More specifically, a weight vector (w_1, ..., w_m) is maintained for each local search engine by the metasearch engine, where w_i corresponds to the i-th term in the database of the search engine. All weights are zero initially. Consider a user query with k terms. Suppose t_i is one of the query terms and for this query, database D is selected to retrieve documents. Then the weight w_i of t_i for D is adjusted according to the retrieval result. If no document is returned from D, then w_i is reduced by $1/k$, indicating that each of the k query terms contributed equally to the poor result. On the other hand, if at least one returned document is clicked by the user, showing the interest of the user to

the document, then w_i is increased by $1/k$, indicating that each of the k query terms is equally responsible for the good result. In other cases, w_i is left unchanged. Over time, if a database has a large positive weight for term t_i, then the database is considered to have responded well to term t_i in the past. Conversely, if a large negative weight is recorded for t_i for a database, then the database has responded poorly to t_i in the past.

An interesting feature of SavvySearch is that its database selection algorithm combines both content-based selection and performance-based selection. Most database selection methods employ content-based selection only. In contrast, performance-based selection method takes into consideration information such as the speed and the connectability of each local search engine when performing database selection. SavvySearch keeps track of two types of performance related information for each search engine (Dreilinger and Howe, 1997). The first is h, the average number of documents returned for the most recent five queries sent to the search engine, and the other is r, the average response time for the most recent five queries. If h is below a threshold T_h (the default value is 1), then a penalty $p_h = (T_h - h)^2/T_h^2$ for the search engine is applied. Similarly, if the average response time r is greater than a threshold T_r (the default is 15 seconds), then a penalty $p_r = (r - T_r)^2/(r_o - T_r)^2$ is computed, where $r_o = 45$ (seconds) is the maximum allowed response time before a timeout.

For a new query q with terms $t_1, ..., t_k$, SavvySearch uses the following formula to compute the ranking score of database D.

$$r(q, D) = \frac{\sum_{i=1}^{k} w_i \cdot \log(N / cf_i)}{\sqrt{\sum_{i=1}^{k} |w_i|}} - (p_h + p_r),$$

where $log(N/cf_i)$ is the *inverse collection frequency weight* of term t_i, N is the number of databases and cf_i is the number of databases having a positive weight value for term t_i.

ProFusion approach

ProFusion (www.profusion.com) employs both training queries and real user queries for learning. In addition, ProFusion incorporates 13 pre-selected topic categories into the learning process (Fan and Gauch, 1999; Gauch, Wang and Gomez, 1996). The 13 categories are “Science and Engineering”, “Computer Science”, “Travel”, “Medical and Biotechnology”, “Business and Finance”, “Social and Religion”, “Society, Law and Government”, “Animals and Environment”, “History”, “Recreation and Entertainment”, “Art”, “Music” and “Food”. For each category, a set of terms is selected to indicate the topic of the category. During the training phase, a set of training queries is identified for each category. For a given category C and a given local database D, each training query

selected for *C* is submitted to *D*. From the top 10 retrieved documents, useful ones are identified by the user conducting the training. Then a score reflecting the effectiveness of *D* with respect to the query and the category is computed by $c * \frac{\sum_{i=1}^{10} N_i}{10} * \frac{R}{10}$, where *c* is a constant; N_i is $1/i$ if the *i*-th ranked document is useful, and 0 otherwise; *R* is the number of useful documents in the 10 retrieved documents. This formula captures both the *rank order* of each useful document and the *precision* of the top 10 retrieved documents. Finally, the scores of database *D* using all training queries selected for category *C* is averaged to yield the *confidence factor* of *D* with respect to category *C*. At the end of the training phase, a confidence factor for each database with respect to each of the 13 categories is obtained. By using the categories and dedicated training queries, how well each local database responds to queries in different categories can be learned.

After the training is completed, the metasearch engine is ready to accept user queries. ProFusion performs database selection as follows. First, each user query *q* is mapped to one or more categories. Query *q* is mapped to category *C* if at least one term in the set of terms associated with *C* appears in *q*. Next, the *ranking score* of each database is computed and the databases are ranked in descending order of their ranking scores. The ranking score of a database for *q* is the sum of the confidence factors of the database with respect to the mapped categories. In ProFusion, only the three databases with the largest ranking scores are selected to search for each query.

ProFusion ranks retrieved documents in descending order of the product of each document's local similarity and the ranking score of the database from which the document is retrieved. Among the documents that are returned to the user, let *d* from database *D* be the one that is clicked by the user first. If the ranking algorithm were perfect, then *d* would be ranked at the top among all returned documents. Therefore, if *d* is not ranked at the top, then some adjustment should be made to fine-tune the ranking system. In ProFusion, when the first clicked document *d* is not ranked at the top, the ranking score of *D* is increased while the ranking scores of those databases whose documents are ranked higher than *d* are reduced. This is carried out by proportionally adjusting the confidence factors of *D* in mapped categories. Clearly, with this ranking score adjustment policy, document *d* is likely to be ranked higher if the same query is processed in the future.

4.3 Collection Fusion

After the database selector has chosen the local search engines for a given query, the next task is to determine what pages to retrieve from each selected search engine and how to merge them into a single ranked list. Different

techniques to implement the document selector will be presented in Section 4.3.1. The merging of results from multiple search engines will be covered in Section 4.3.2.

4.3.1 Document Selection. A search engine typically retrieves pages in descending order of the locally computed desirabilities of the pages. Therefore, the problem of selecting what pages to retrieve from a local database can be translated into the problem of how many pages to retrieve from the database. If *k* pages are to be retrieved from a local database, then the *k* highest ranked pages will be retrieved.

A simple document selector can request that each selected search engine returns all the pages that are retrieved from the search engine. This approach may cause too many pages to be returned from each local system, leading to higher communication cost and more result merging effort. Another simple method to implement a document selector is to utilize the fact that most search engines return retrieved results in groups. Usually, only the top 10 to 20 results are returned in the first result page but the user can make additional requests for more result pages and more results. Hence, a document selector may ask each search engine to return the first few result pages. This method tends to return the same number of pages from each selected search engine. Since different search engines may contain different numbers of useful pages for a given query, retrieving the same number of pages from each search engine is likely to cause over-retrieval from less useful databases and under-retrieval from highly useful databases.

More elaborate document selection methods try to tie the number of pages to retrieve from a search engine to the ranking score (or the rank) of the search engine relative to the ranking scores (or ranks) of other search engines. This can lead to proportionally more pages to be retrieved from search engines that are ranked higher or have higher ranking scores. This type of approaches is referred to as *weighted allocation* approaches in (Meng et al, 2002).

For each user query, the database selector of the metasearch engine computes a rank (i.e., 1st, 2nd, ...) and a ranking score for each local search engine. Both the rank information and the ranking score information can be used to determine the number of pages to retrieve from different local search engines. For example, in the D-WISE system (Yuwono and Lee, 1997), the ranking score information is used. Suppose for a given query *q*, r_i denotes the ranking score of the local database D_i, $i=1,...,k$, where *k* is the number of selected local databases for the query, and $\alpha = \sum_{j=1}^{k} r_j$ denotes the total ranking score for all selected local databases. D-WISE uses the ratio r_i / α to determine how many pages should be retrieved from D_i. More precisely, if *m* pages across these *k* databases are to be retrieved, then D-WISE retrieves $m * r_i / \alpha$ pages from database D_i. An example system that uses the rank information to select documents is CORI Net (Callan et al, 1995). Specifically, if *m* is the total number of pages to be retrieved from *k*

selected local search engines, then m * $\frac{2(1+k-i)}{k(k+1)}$ pages are retrieved from the *i*-th ranked local database, *i=1,...,k*. Since $\frac{2(1+k-u)}{k(k+1)} > \frac{2(1+k-v)}{k(k+1)}$ for $u < v$, more pages will be retrieved from the *u*-th ranked database than from the *v*-th ranked database. Because, $\sum_{i=1}^{k} \frac{2(1+k-i)}{k(k+1)} = 1$, exactly *m* pages will be retrieved from the *k* top-ranked databases. In practice, it may be wise to retrieve slightly more than *m* pages from local databases in order to reduce the likelihood of missing useful pages.

It is possible to combine document selection and database selection in a single integrated process. In Section 4.2, we described a method for ranking databases in descending order of the estimated similarity of the most similar document in each database for a given query. A combined database selection and document selection method for finding the *m* most similar pages based on these ranked databases was proposed in (Yu et al, 1999). This method is sketched below. First, for some small positive integer *s* (e.g., *s* can be 2), each of the *s* top ranked databases are searched to obtain the actual global similarity of its most similar page. This may require some locally top-ranked pages to be retrieved from each of these databases. Let *min_sim* be the minimum of these *s* similarities. Next, from these *s* databases, retrieve all pages whose actual global similarities are greater than or equal to *min_sim*. If *m* or more pages have been retrieved, then sort them in descending order of similarities, return the top *m* pages to the user and terminate this process. Otherwise, the next top ranked database (i.e., the (*s*+1)th ranked database) is considered and its most similar page is retrieved. The actual global similarity of this page is then compared with the current *min_sim* and the minimum of these two similarities will be used as the new *min_sim*. Then retrieve from these *s*+1 databases all pages whose actual global similarities are greater than or equal to the new *min_sim*. This process is repeated until *m* or more pages are retrieved and the *m* pages with the largest similarities are returned to the user. A seemingly problem with this combined method is that the same database may be searched multiple times. In practice, this problem can be avoided by retrieving and caching an appropriate number of pages when a database is searched for the first time. In this way, all subsequent "interactions" with the database would be carried out using the cached results. This method has the following property (Yu et al, 1999). If the databases containing the *m* desired pages are ranked higher than other databases and the similarity (or desirability) of the *m*-th most similar (desirable) page is distinct, then all of the *m* desired pages will be retrieved while searching at most one database that does not contain any of the *m* desired pages.

4.3.2 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 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 result from the metasearch engine, he/she 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, pages in the merged result should be ranked in descending order of global similarities (or global desirabilities). However, the heterogeneities that exist among local search engines and between the metasearch engine and local search engine make result merging a challenging problem. Usually, pages returned from a local search engine are ranked based on these pages' local similarities. Some local search engines make the local similarities of returned pages available to the user (as a result, the metasearch engine can also obtain the local similarities) while other search engines do not make them available. For example, Google and AltaVista do not provide local similarities while Northern Light and FirstGov do. To make things worse, local similarities returned from different local search engines, even when made available, may be incomparable due to the use of different similarity functions and term weighting schemes by different local search engines. Furthermore, the local similarities and the global similarity of the same page may be quite different still as the metasearch engine may use a similarity function that is different from those used in local systems. In fact, even when the same similarity function were used by all local systems and the metasearch engine, local and global similarities of the same page may still be very different. This is because some statistics used to compute term weights, for example the document frequency of a term, are likely to be different in different systems.

The challenge here is how to merge the pages returned from multiple local search engines into a single ranked list in a reasonable manner in the absence of local similarities and/or in the presence of incomparable similarities. An additional complication is that retrieved pages may be returned by different numbers of local search engines. For example, one page could be returned by one of the selected local 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 returned 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 (www.neci.nec.com/~lawrence/inquirus.html). Inquirus ranks pages returned from local search engines based on analyzing the contents of downloaded pages, and it employs a ranking formula that combines similarity and proximity matches (Lawrence and Lee Giles, 1998). In addition to be able to rank results based on desired global similarities, this approach also has some other advantages (Lawrence and Lee Giles, 1998). For example, when attempting to download pages, obsolete URLs can be discovered. This helps to remove pages with dead URLs from the final result list. In addition, downloading pages on the fly ensures that pages will be ranked based on their current contents. In contrast, similarities computed by local search engines may be based on obsolete versions of Web pages. The biggest drawback of this approach is its slow speed as fetching pages and analyzing them on the fly can be time consuming.

Most result merging methods utilize the local similarities or local ranks of returned pages to perform merging. The following cases can be identified:

1. *Selected databases for a given query do not share pages and all returned pages have local similarities attached*.

   In this case, each result page will be returned from just one search engine. Even though all returned pages have local similarities, these similarities may be normalized using different ranges by different local search engines. For example, one search engine may normalize its similarities between 0 and 1 and another between 0 and 1000. In this case, all local similarities should be re-normalized based on a common range, say [0, 1], to improve the comparability of these local similarities (Dreilinger and Howe, 1997; Selberg and Etzioni, 1997).

   Re-normalized similarities can be further adjusted based on the usefulness of different databases for the query. Recall that when database selection is performed for a given query, the usefulness of each database is estimated and is represented as a score. The database scores can be used to adjust re-normalized similarities. The idea is to give preference to pages retrieved from highly ranked databases. In CORI Net (Callan et al, 1995), the adjustment works as follows. Let s be the ranking score of local database D and $\bar{s}$ be the average of the scores of all searched databases for a given query. Then the following weight is assigned to D: $w = 1 + k * (s - \bar{s}) / \bar{s}$, where k is the number of databases searched for the given query. It is easy to see from this formula that databases with higher scores will have higher weights. Let x be the re-normalized similarity of page p retrieved from D. Then CORI Net computes the adjusted similarity of p by $w * x$. The result merger lists returned pages in descending order of adjusted similarities. A similar method is used in ProFusion (Gauch et al, 1996). For a

given query, the adjusted similarity of a page *p* from a database *D* is the product of the re-normalized similarity of *p* and the ranking score of *D*.

2. *Selected databases for a given query do not share pages but some returned pages do not have local similarities attached*. Again, each result page will be returned by one local search engine. In general, there are two types of approaches for tackling the result merging problem in this case. The first type uses the local rank information of returned pages directly to perform the merge. Note that in this case, local similarities that may be available for some returned pages would be ignored. The second type first converts local ranks to local similarities and then applies techniques described for Case 1 to perform the merge.

   One simple way to use rank information only for result merging is as follows (Meng et al, 2002). First, arrange the searched databases in descending order of usefulness scores. Next, a round-robin method based on the database order and the local page rank order is used to produce an overall rank for all returned pages. Specifically, in the first round, the top-ranked page from each searched database is taken and these pages are ordered based on the database order such that the page order and the database order are consistent; if not enough pages have been obtained, the second round starts, which takes the second highest-ranked page from each searched database, orders these pages again based on the database order, and places them behind those pages selected earlier. This process is repeated until the desired number of pages are obtained.

   In the D-WISE system (Yuwono and Lee, 1997), the following method for converting ranks into similarities is employed. For a given query, let r_i be the ranking score of database D_i, r_{min} be the smallest database ranking score, r be the local rank of a page from D_i, and g be the converted similarity of the page. The conversion function is $g = 1 - (r - 1) * F_i$, where $F_i = r_{min} /(m * r_i)$ and m is the number of documents desired across all searched databases. This conversion has the following properties. First, all locally top-ranked pages have the same converted similarity, i.e., 1. Second, F_i is the difference between the converted similarities of the j-th and the $(j+1)$th ranked pages from database D_i, for any $j=1,2,...$. Note that the distance is larger for databases with smaller ranking scores. Consequently, if the rank of a page p in a higher rank database is the same as the rank of a page p' in a lower rank database and neither p nor p' is top-ranked, then the converted similarity of p will be higher than that of p'. This property can lead to the selection of more pages from databases with higher scores into the merged result. As an example, consider two databases D_1 and D_2. Suppose $r_1 = 0.2$, $r_2 = 0.5$ and $m = 4$. Then, $r_{min} = 0.2$, $F_1 = 0.25$ and $F_2 = 0.1$. Thus, the three top ranked pages from D_1 will have converted

similarities 1, 0.75, and 0.5, respectively, and the three top ranked pages from D_2 will have converted similarities 1, 0.9, and 0.8, respectively. As a result, the merged list will contain three pages from D_2 and one page from D_1.

3. *Selected databases for a given query share pages*. In this case, the same page may be returned by multiple local search engines. Result merging in this situation is usually carried out in two steps. In the first step, all pages, regardless of whether they are returned by one or more search engines, techniques discussed in the first two cases can be applied to compute their similarities for merging. 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 is used in ProFusion (Gauch et al, 1996) and the *sum* function is used in MetaCrawler (Selberg and Etzioni, 1997).

5. Conclusions

In the last decade, we have all witnessed the explosion of the Web. As up to now, the Web has become the largest digital library used by 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 article provides an overview of these technologies, from basic ideas to more advanced algorithms. As can be seen from this article, Web-based search technology has its root from 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. The upcoming transition of the Web from mostly HTML pages to XML pages will probably have a significant impact on Web-based search technology.

Acknowledgement: This work is supported in part by the following NSF grants: IIS-9902872, IIS-9902792 and EIA-9911099.

References

Bergman, M. (2000). The Deep Web: Surfacing the hidden value. BrightPlanet, available at www.completeplanet.com/ Tutorials/DeepWeb/index.asp (Date of access: April 25, 2002).

Callan, J. (2000). Distributed information retrieval. In W. Bruce Croft (Ed.), Advances in information retrieval: Recent research from the Center for Intelligent Information Retrieval (pp.127-150). Kluwer Academic Publishers.

Callan, J., Connell, M., and Du, A. (1999). Automatic discovery of language models for text databases. ACM SIGMOD Conference, Philadelphia, PA, 479-490.

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

Callan, J., Lu, Z., and Croft, W. (1995). Searching distributed collections with inference networks. ACM SIGIR Conference, Seattle, 21-28.

Chakrabarti, S., et al. (1998). Automatic resource compilation by analyzing hyperlink structure and associated text. 7th International World Wide Web Conference, Brisbane, Australia, 65-74.

Chakrabarti, S., et al. (1999). Mining the Web's link structure. IEEE Computer, 32, 8, 60-67.

Croft, W. (2000). Combining approaches to information retrieval. In W. Bruce Croft (Ed.), Advances in information retrieval: Recent research from the Center for Intelligent Information Retrieval (pp.1-36). Kluwer Academic Publishers.

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

Dreilinger, D., and Howe, A. (1997). Experiences with selecting search engines using metasearch. ACM Transactions on Information Systems, 15, 3, 195-222.

Fan, Y., and Gauch, S. (1999). Adaptive agents for information gathering from multiple, distributed information sources. AAAI Symposium on Intelligent Agents in Cyberspace, Stanford University, 40-46.

Gauch, S., Wang, G., and Gomez, M. (1996). ProFusion: Intelligent fusion from multiple, distributed search engines. Journal of Universal Computer Science, 2, 9, 637-649.

Gravano, L., Chang, C., Garcia-Molina, H., and Paepcke, A. (1997). Starts: Stanford proposal for Internet meta-searching. ACM SIGMOD Conference, Tucson, Arizona, 207-218.

Hawking, D., Craswell, N., Bailey, P., and Griffiths, K. (2001). Measuring Search Engine Quality. Journal of Information Retrieval. 4, 1, 33-59.

Hearst, M., and Pedersen, J. (1996). Reexamining the cluster hypothesis: Scatter/gather on retrieval results. ACM SIGIR Conference, Zurich, Switzerland, 76-84.

Kahle, B., and Medlar, A. (1991). An information system for corporate users: Wide area information servers. Technical Report TMC199, Thinking Machine Corporation.

Kirsch, S. (1998). The future of Internet search: Infoseek's experiences searching the Internet. ACM SIGIR Forum, 32, 2, 3-7.

Kleinberg, J. (1998). Authoritative sources in a hyperlinked environment. Ninth ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California, 668-677.

Koster, M. (1994). ALIWEB: Archie-like indexing in the Web. Computer Networks and ISDN Systems, 27, 2, 175-182.

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

Manber, U., and Bigot, P. (1997). The search broker. USENIX Symposium on Internet Technologies and Systems, Monterey, California, 231-239.

Meng, W., Yu, C., and Liu, K. (2002). Building efficient and effective metasearch engines. ACM Computing Surveys, 34, 1, 48-84.

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

Pratt, W., Hearst, H., and Fagan, L. (1999). A knowledge-based approach to organizing retrieved documents. Sixteenth National Conference on Artificial Intelligence, Orlando, Florida, 80-85.

Salton, G., and McGill, M. (1983). Introduction to modern information retrieval. New York: McCraw-Hill.

Selberg, E., and Etzioni, O. (1997). The MetaCrawler architecture for resource aggregation on the Web. IEEE Expert, 12, 1, 8-14.

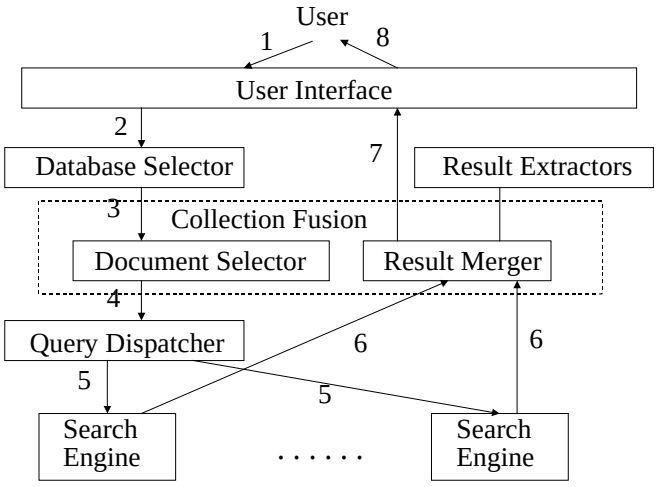
R. F. Wisman. (2003). Searching the Web. In The Internet Encyclopedia. Edited by H. Bidgoli. Wiley, 2003.

Wu, Z., Meng, W., Yu, C., and Li, Z. (2001). Towards a highly scalable and effective metasearch engine. Tenth World Wide Web Conference, Hong Kong, 386-395.

Yu, C., Meng, W., Liu, L., Wu, W., and Rishe, N. (1999). Efficient and effective metasearch for a large number of text databases. Eighth ACM International Conference on Information and Knowledge Management, Kansas City, 217-214.

Yuwono, B., and Lee, D. (1997). Server ranking for distributed text resource systems on the Internet. Fifth International Conference On Database Systems For Advanced Applications, Melbourne, Australia, 391-400.

Figure 1: Metasearch Software Component Architecture



Glossary terms:

Collection fusion: The technique that determines how to retrieve documents from multiple collections and merge them into a single ranked list.

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

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.

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

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

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

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