# Building Efficient and Effective Metasearch Engines

Weiyi Meng
State University of New York at Binghamton
Clement Yu
University of Illinois at Chicago
and
King-Lup Liu
DePaul University

Frequently a user's information needs are stored in the databases of multiple search engines. It is inconvenient and inefficient for an ordinary user to invoke multiple search engines and identify useful documents from the returned results. To support unified access to multiple search engines, a metasearch engine can be constructed. When a metasearch engine receives a query from a user, it invokes the underlying search engines to retrieve useful information for the user. Metasearch engines have other benefits as a search tool such as increasing the search coverage of the Web and improving the scalability of the search. In this article, we survey techniques that have been proposed to tackle several underlying challenges for building a good metasearch engine. Among the main challenges, the database selection problem is to identify search engines that are likely to return useful documents to a given query. The document selection problem is to determine what documents to retrieve from each identified search engine. The result merging problem is to combine the documents returned from multiple search engines. We will also point out some problems that need to be further researched.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems–distributed databases; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval–search process; selection process; H.3.4 [**Information Storage and Retrieval**]: Systems and Software–information networks

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Metasearch, Information Resource Discovery, Distributed Collection, Collection Fusion, Distributed Information Retrieval

Name: Weiyi Meng
Address: Department of Computer Science, State University of New York at Binghamton, Binghamton, NY 13902; email: meng@cs.binghamton.edu.
Name: Clement Yu
Address: Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607; email: yu@cs.uic.edu.
Name: King-Lup Liu
Address: School of Computer Science, Telecommunications and Information Systems, DePaul University, Chicago, IL 60604; kliu@cti.depaul.edu.

## 1. INTRODUCTION

The Web has become a vast information resource in recent years. Millions of people use the Web on a regular basis and the number is increasing rapidly. Most data on the Web are in the form of text or image. In this survey, we concentrate on the search of text data.

Finding desired data on the Web in a timely and cost effective way is a problem of wide interest. In the last several years, many search engines have been created to help Web users find desired information. Each search engine has a *text database* that is defined by the set of documents that can be searched by the search engine. When there is no confusion, the term *database* and the phrase *search engine* will be used interchangeably in this survey. Usually, an index for all documents in the database is created in advance. For each *term* which represents a content word or a combination of several (usually adjacent) content words, this index can identify the documents that contain the term quickly. Google, Altavista, Excite, Lycos and HotBot are all popular search engines on the Web.

Two types of search engines exist. General-purpose search engines aim at providing the capability to search all pages on the Web. The search engines we mentioned in the previous paragraph are a few of the well-known ones. Special-purpose search engines, on the other hand, focus on documents in confined domains such as documents in an organization or in a specific subject area. For example, the Cora search engine (cora.whizbang.com) focuses on computer science research papers and Medical World Search (www.mwsearch.com) is a search engine for medical information. Most organizations and business sites have installed search engines for their pages. It is believed that hundreds of thousands of special-purpose search engines currently exist on the Web [Bergman 2000].

The amount of data on the Web is huge. It is believed that by February of 1999, there were already more than 800 million publicly indexable web pages [Lawrence and Lee Giles 1999] and the number is well over 1 billion now (Google has indexed over 1.3 billion pages) and is increasing at a very high rate. Many believe that employing a single general-purpose search engine for all data on the Web is unrealistic [Hawking and Thistlewaite 1999; Sugiura and Etzioni 2000; Wu et al. 2001]. First, its processing power may not scale to the rapidly increasing and virtually unlimited amount of data. Second, gathering all the data on the Web and keeping them reasonably up-to-date are extremely difficult if not impossible objectives. Programs (e.g. *Web robots*) used by major search engines to gather data automatically may slow down local servers and are increasingly unpopular. Furthermore, many sites may not allow their documents to be indexed but instead may allow the documents to be accessed through their search engines only (these sites are part of the so-called *Deep Web* [Bergman 2000]). Consequently, we have to live with the reality of having a large number of special-purpose search engines that each covers a portion of the Web.

A *metasearch engine* is a system that provides unified access to multiple existing search engines. A metasearch engine does not maintain its own index of documents.

However, a sophisticated metasearch engine may maintain information about the contents of its underlying search engines to provide better service. In a nutshell, when a metasearch engine receives a user query, it first passes the query (with necessary reformatting) to the appropriate underlying search engines, and then collects and reorganizes the results received from them. A simple two-level architecture of a metasearch engine is depicted in Figure 1. This two-level architecture can be generalized to a hierarchy of more than two levels when the number of underlying search engines becomes large [Baumgarten 1997; Gravano and Garcia-Molina 1995; Sheldon et al. 1994; Yu et al. 1999b].

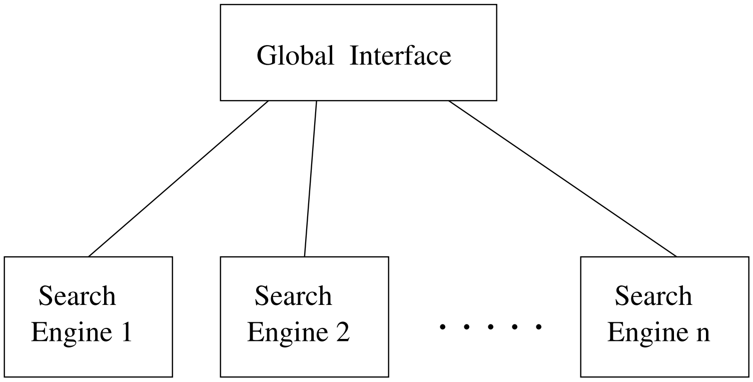

Fig. 1. A Simple Metasearch Architecture

There are a number of reasons for the development of a metasearch engine and we discuss these reasons below.

(1) *Increase the search coverage of the Web.* A recent study [Lawrence and Lee Giles 1999] indicates that the coverage of the Web by individual major general-purpose search engines has been decreasing steadily. This is mainly due to the fact that the Web has been increasing at a much faster rate than the indexing capability of any single search engine. By combining the coverages of multiple search engines through a metasearch engine, a much higher percentage of the Web can be searched. While the largest general-purpose search engines index less than 2 billion Web pages, all special-purpose search engines combined may index up to 500 billion Web pages [Bergman 2000].

(2) *Solve the scalability of searching the Web.* As we mentioned earlier, the approach of employing a single general-purpose search engine for the entire Web has poor scalability. In contrast, if a metasearch engine on top of all the special-purpose search engines can be created as an alternative to search the entire Web, then the problems associated with employing a single general-purpose search engine will either disappear or be significantly alleviated. the size of a typical special-purpose search engine is much smaller than that of a major general-purpose search engine. Therefore, it is much easier for it to keep its index data more up to date (i.e. updating of index data to reflect the changes of documents can be carried out more frequently). It is also much easier to build the

necessary hardware and software infrastructure for a special-purpose search engine. As a result, the metasearch engine approach for searching the entire Web is likely to be significantly more scalable than the centralized general-purpose search engine approach.

(3) *Facilitate the invocation of multiple search engines.* The information needed by a user is frequently stored in the databases of multiple search engines. As an example, consider the case when a user wants to find the best 10 newspaper articles about a special event. It is likely that the desired articles are scattered across the databases of a number of newspapers. The user can send his/her query to every newspaper database and examine the retrieved articles from each database to identify the 10 best articles. This is a formidable task. First, the user will have to identify the sites of the newspapers. Second, the user will need to send the query to each of these databases. Since different databases may accept queries in different formats, the user will have to format the query correctly for each database. Third, there will be no overall quality ranking among the articles returned from these databases even though the retrieved articles from each individual database may be ranked. As a result, it will be difficult for the user, without reading the contents of the articles, to determine which articles are likely to be among the most useful ones. If there are a large number of databases, each returning some articles to the user, then the user will simply be overwhelmed. If a metasearch engine on top of these local search engines is built, then the user only needs to submit one query to invoke all local search engines via the metasearch engine. A good metasearch engine can rank the documents returned from different search engines properly. Clearly, such a metasearch engine makes the user's task much easier.

(4) *Improve the retrieval effectiveness.* Consider the scenario where a user needs to find documents in a specific subject area. Suppose that there is a special-purpose search engine for this subject area and there is also a general-purpose search engine that contains all the documents indexed by the special-purpose search engine in addition to many documents unrelated to this subject area. It is usually true that if the user submits the same query to both of the two search engines, the user is likely to obtain better results from the special-purpose search engine than the general-purpose search engine. In other words, the existence of a large number of unrelated documents in the general-purpose search engine may hinder the retrieval of desired documents. In text retrieval, documents in the same collection can be grouped into clusters such that the documents in the same cluster are more related than documents across different clusters. When evaluating a query, clusters related to the query can be identified first and then the search can be carried out for these clusters. This method has been shown to improve the retrieval effectiveness of the system [Xu and Croft 1999]. For documents on the Web, the databases in different special-purpose search engines are natural clusters. As a result, if for any given query submitted to the metasearch engine, the search can be restricted to only special-purpose search engines related to the query, then it is likely that better retrieval effectiveness can be achieved using the metasearch engine than using a general-purpose search engine. While it may be possible for a general-purpose search

engine to cluster its documents to improve retrieval effectiveness, the quality of these clusters may not be as good as the ones corresponding to special-purpose search engines. Furthermore, constructing and maintaining the clusters consumes more resources of the general-purpose search engine.

This article has three objectives. First, we review the main technical issues in building a good metasearch engine. Second, we survey different proposed techniques for tackling these issues. Third, we point out new challenges and research directions in the metasearch engine area.

The rest of the article is organized as follows. In Section 2, we provide a short overview of some basic concepts on information retrieval (IR). These concepts are important for the discussions in this article. In Section 3, we outline the main software components of a metasearch engine. In Section 4, we discuss how the autonomy of different local search engines, as well as the heterogeneities among them, may affect the building of a good metasearch engine. In Section 5, we survey reported techniques for the database selection problem (i.e. determining which databases to search for a given user query). In Section 6, we survey known methods for the document selection problem (i.e. determining what documents to retrieve from each selected database for a user query). In Section 7, we report different techniques for the result merging problem (i.e. combining results returned from different local databases into a single ranked list). In Section 8, we present some new challenges for building a good metasearch engine.

## 2. BASIC INFORMATION RETRIEVAL

Information retrieval deals with techniques for finding relevant (useful) documents for any given query from a collection of documents. Documents are typically preprocessed and represented in a form that facilitates efficient and accurate retrieval. In this section, we first overview some basic concepts in classical information retrieval and then point out several features specifically associated with Web search engines.

### 2.1 Classical Information 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.* After removing stop words and stemming, each document can be logically represented by a vector of $n$ terms [Salton and McGill 1983; Yu and Meng 1998], 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 0 because most terms are absent from any given document. When a term is present in a document, the weight assigned to the term is usually based on two factors. The *term frequency* ($tf$) of a term in a document is the number of times the term occurs 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. As a consequence, the *term frequency weight* ($tfw$) of the term in the document is usually a monotonically increasing function of its term frequency. The second factor affecting the weight of a term is the *document frequency* ($df$), which is the number of documents having the term. 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 based on its document frequency is usually monotonically decreasing 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 query is simply a question written in text[1]. It can be transformed into an n-dimensional vector as well. Specifically, the non-content words are eliminated by comparing the words in the query against the stop word list. Then, words in the query are mapped into terms and finally, terms are weighted based on term frequency and/or document frequency information.

After the vectors of all documents and a query are formed, document vectors which are close to the query vector are retrieved. A *similarity function* can be used to measure the degree of closeness between two vectors. One simple function is the dot product function, $dot(q, d) = \sum_{k=1}^{n} q_i * d_i$, where $q = (q_1, ..., q_n)$ is the vector of a query and $d = (d_1, ..., d_n)$ is the vector of a document. The dot product function is a weighted sum of the terms in common between the two vectors. The dot product function tends to favor long documents having many terms, because the chance of having more terms in common between a document and a given query is higher for a longer document than a shorter document. In order that all documents have a fair chance of being retrieved, the $Cosine$ function can be utilized. It is 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 and it always returns a value between 0 and 1 when the weights are non-negative. It gets the value 0 if there is no term in common between the query and the document; its value is 1 if the query and the document vectors are identical or one vector is a positive constant multiple of the other.

A common measure for retrieval effectiveness is *recall* and *precision*. For a given query submitted by a user, suppose that the set of relevant documents with respect to the query in the document collection can be determined. The two quantities recall and precision can be defined as follows:

$$recall = \frac{the\ number\ of\ retrieved\ relevant\ documents}{the\ number\ of\ relevant\ documents} \tag{1}$$

[1] We note that Boolean queries are also supported by many IR systems. In this article, we concentrate on vector space queries only unless other types of queries are explicitly identified. A study of 51,473 real user queries submitted to the Excite search engine indicates that less than 10% of these queries are Boolean queries [Jansen et al. 1998].

$$precision = \frac{the\ number\ of\ retrieved\ relevant\ documents}{the\ number\ of\ retrieved\ documents} \tag{2}$$

To evaluate the effectiveness of a text retrieval system, a set of test queries is used. For each query, the set of relevant documents is identified in advance. For each such query, a precision value for each distinct recall value is obtained. When these sets of recall-precision values are averaged over the set of test queries, an average recall-precision curve is obtained. This curve is used as the measure of the effectiveness of the system.

An ideal information retrieval system retrieves all relevant documents and nothing else (i.e. both recall and precision equal to 1). In practice, this is not possible, as a user's needs may be incorrectly or imprecisely specified by his/her query and the user's concept of relevance varies over time and is difficult to capture. Thus, the retrieval of documents is implemented by employing some similarity function which approximates the degrees of relevance of documents with respect to a given query. Relevance information due to previous retrieval results may be utilized by systems with learning capabilities to improve retrieval effectiveness. In the remaining portion of this paper, we shall restrict ourselves to the use of similarity functions in achieving high retrieval effectiveness, except for certain situations where users' feedback information is incorporated.

### 2.2 Web Search Engines

A Web search engine is essentially an information retrieval system for Web pages. However, Web pages have several features that are not usually associated with documents in traditional IR systems and these features have been explored by search engine developers to improve the retrieval effectiveness of search engines.

The first special feature of Web pages is that they are highly tagged documents. At present, most Web pages are in HTML format. In the foreseeable future, XML documents may be widely used. These tags often convey rich information regarding the terms used in documents. For example, a term appearing in the title of a document or emphasized with a special font can provide hint that the term is rather important in indicating the contents of the document. Tag information has been used by a number of search engines such as Google and AltaVista to better determine the importance of a term in representing the contents of a page. For example, a term occurring in the title or the header of a page may be considered to be more important than the same term occurring in the main text. As another example, a term typed in a special font such as bold face and large fonts is likely to be more important than the same term not in any special font. Studies indicate that the higher weights assigned to terms due to their locations or their special fonts or tags can yield higher retrieval effectiveness than schemes which do not take advantage of the location or tag information [Cutler et al. 1997].

The second special feature of Web pages is that they are extensively linked. A link from page A to page B provides a convenient path for a Web user to navigate from page A to page B. Careful analysis can reveal that such a simple link could contain several pieces of information that may be made use of to improve retrieval effectiveness. First, such a link indicates a good likelihood that the contents of the two pages are related. Second, the author of page A values the contents of page

B. The linkage information has been used to compute the global importance (i.e. *PageRank*) of Web pages based on whether a page is pointed to by many pages and/or by important pages [Page et al. 1998]. This has been successfully used in the Google search engine to improve retrieval effectiveness. The linkage information has also been used to compute the authority (the degree of importance) of Web pages with respect to a given topic [Kleinberg 1998]. IBM's *Clever Project* is to develop a search engine that employs the technique of computing the authorities of Web page for a given query [Chakrabarti et al. 1999].

Another way to utilize the linkage information is as follows. When a page A has a link to page B, a set of terms known as *anchor terms* is usually associated with the link. The purpose of using the anchor terms is to provide information regarding the contents of page B to facilitate the navigation by human users. The anchor terms often provide related terms or synonyms to the terms used to index page B. To utilize such valuable information, several search engines like Google [Brin and Page 1998] and WWWW [McBryan 1994] have suggested also using anchor terms to represent linked pages (e.g. page B). In general, a Web page may be linked by many other Web pages and has many associated anchor terms.

## 3. METASEARCH ENGINE COMPONENTS

In a typical session of using a metasearch engine, a user submits a query to the metasearch engine through a user friendly interface. The metasearch engine then sends the user query to a number of underlying search engines (which will be called *component search engines* in this article). Different component search engines may accept queries in different formats. The user query may thus need to be translated to an appropriate format for each local system. After the retrieval results from the local search engines are received, the metasearch engine merges the results into a single ranked list and presents the merged result, possibly only the top portion of the merged result, to the user. The result could be a list of documents or more likely a list of document identifiers (e.g. URLs for web pages on the Web) with possibly short companion descriptions. In this article, we use "documents" and "document identifiers" interchangeably unless it is important to distinguish them.

Now let us introduce the concept of *potentially useful documents*.

DEFINITION 1. *Suppose there is a* similarity function *that computes the similarities between documents and any given query and the similarity of a document with a given query approximates the degree of the relevance of the document to the "average user" who submits the query. For a given query, a document d is said to be* **potentially useful** *if it satisfies one of the following conditions:*

*(1) If m documents are desired in the final result for some positive integer m, then the similarity between d and the query is among the m highest of all similarities between all documents and the query.*

*(2) If every document whose similarity with the query exceeds a pre-specified threshold is desired, then the similarity between d and the query is greater than the threshold.*

In a metasearch engine environment, different component search engines may employ different similarity functions. For a given query and a document, their similarities computed by different local similarity functions are likely to be different and incomparable. To overcome this problem, the similarities in the above definition are computed using a similarity function defined in the metasearch engine. In other words, global similarities are used.

Note that, in principle, the two conditions in Definition 1 are mutually translatable. In other words, for a given $m$ in Condition 1, a threshold in Condition 2 can be determined such that the number of documents whose similarities exceed the threshold is $m$, and vice versa. However, in practice, the translation can only be done when substantial statistical information about the text database is available. Usually, a user specifies the number of documents he or she would like to view. The system uses a threshold to determine what documents should be retrieved and displays only the desired number of documents to the user.

The goal of text retrieval is to maximize the retrieval effectiveness while minimizing the cost. For a centralized retrieval system, this can be implemented by retrieving as many potentially useful documents as possible while retrieving as few non-potentially useful documents as possible. In a metasearch engine environment, the implementation should be carried in two levels. First, we should select as many potentially useful databases (these databases contain potentially useful documents) to search as possible while minimizing the search of useless databases. Second, for each selected database, we should retrieve as many potentially useful documents as possible while minimizing the retrieval of useless documents.

A reference software component architecture of a metasearch engine is illustrated in Figure 2. The numbers on the edges indicate the sequence of actions for a query to be processed. We now discuss the functionality of each software component and the interactions among these components.

**Database selector** If the number of component search engines in a metasearch engine is small, it may be reasonable to send each user query to all of them. However, if the number is large, say in the thousands, then sending each query to all component search engines is no longer a reasonable strategy. This is because in this case, a large percentage of the local databases will be useless with respect to the query. Suppose a user is interested in only the 10 best matched documents for a query. Clearly, the 10 desired documents are contained in at most 10 databases. Consequently, if the number of databases is much larger than 10, then a large number of databases will be useless with respect to this query. Sending a query to the search engines of useless databases has several problems. First, dispatching the query to useless databases wastes the resources at the metasearch engine site. Second, transmitting the query to useless component search engines from the metasearch engine and transmitting useless documents from these search engines to the metasearch engine would incur unnecessary network traffic. Third, when a query is evaluated against useless component databases, resources at these local systems would be wasted. Fourth, when a large number of documents are returned from useless databases, more effort would be needed by the metasearch engine to identify useful documents. Therefore, it is important to send each user query to only potentially useful databases. The problem of identifying potentially useful databases to search for a given query is known as the *database selection*

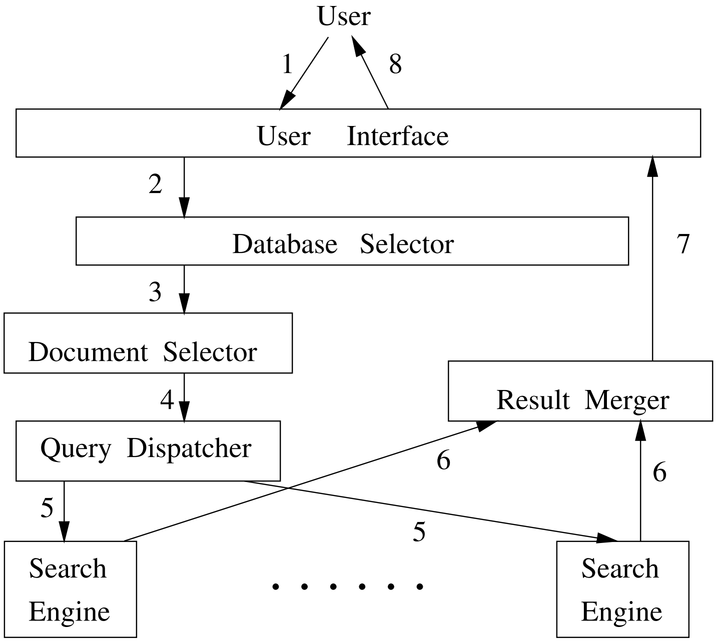

Fig. 2. Metasearch Software Component Architecture

*problem*. The software component *database selector* is responsible for identifying potentially useful databases for each user query. A good database selector should correctly identify as many potentially useful databases as possible while minimizing wrongly identifying useless databases as potentially useful ones. Techniques for database selection will be covered in Section 5.

**Document selector** For each search engine selected by the *database selector*, the component *document selector* determines what documents to retrieve from the database of the search engine. The goal is to retrieve as many potentially useful documents from the search engine as possible while minimizing the retrieval of useless documents. When a large number of useless documents are returned from a search engine, more effort would be needed by the metasearch engine to identify potentially useful documents. Several factors may affect the selection of documents to retrieve from a component search engine such as the number of potentially useful documents in the database and the similarity function used by the component system. These factors help determine either the number of documents that should be retrieved from the component search engine or a local similarity threshold such that only those documents whose local similarity with the given query is higher than or equal to the threshold should be retrieved from the component search engine. Different methods for selecting documents to retrieve from local search engines will be described in Section 6.

**Query dispatcher** The *query dispatcher* is responsible for establishing a connection with the server of each selected search engine and passing the query to it. HTTP (HyperText Transfer Protocol) is used for the connection and data transfer (sending queries and receiving results). Each search engine has its own requirements on the HTTP request method (e.g. the GET method or the POST method) and query format (e.g. the specific query box name). The query dispatcher must follow the requirements of each search engine correctly. Note that in general, the query sent to a particular search engine may or may not be the same as that received by the metasearch engine. In other words, the original query may be translated to a new query before being sent to a search engine. The translation of Boolean queries across heterogeneous information sources is studied in [Chang and Garcia-Molina 1999].

For vector space queries, query translation is usually as straightforward as just retaining all the terms in the user query. There are two exceptions, however. First, the relative weights of query terms in the original user query may be adjusted before the query is sent to a component search engine. This is to adjust the relative importance of different query terms and the adjustment can be accomplished by repeating some query terms an appropriate number of times. Second, the number of documents to be retrieved from a component 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 $m$ documents should be retrieved. The document selector may decide that $k$ documents should be retrieved from a particular component search engine. In this case, the number $k$, usually different from $m$, should be part of the translated query to be sent to the component search engine.

**Result merger** After the results from selected component search engines are returned to the metasearch engine, the *result merger* combines the results into a single ranked list. The top $m$ documents in the list are then forwarded to the user interface to be displayed, where $m$ is the number of documents desired by the user. A good result merger should rank all returned documents in descending order of their global similarities with the user query. Different result merging techniques will be discussed in Section 7.

In the remaining discussions, we will concentrate on the following three main components, namely, the *database selector*, the *document selector* and the *result merger*. Except for the query translation problem, the component *query dispatcher* will not be discussed further in this survey. Query translation for Boolean queries will not be discussed in this article as we focus on vector space queries only. More discussions on query translation for vector space queries will be provided at appropriate places while discussing other software components.

## 4. SOURCES OF CHALLENGES

In this section, we first review the environment in which a metasearch engine is to be built and then analyze why such an environment causes tremendous difficulties to building an effective and efficient metasearch engine.

Component search engines that participate in a metasearch engine are often built and maintained independently. Each search engine decides the set of documents it wants to index and provide search service to. It also decides how documents should be represented/indexed and when the index should be updated. Similarities between documents and user queries are computed using a similarity function. It is completely up to each search engine to decide what similarity function to use. Commercial search engines often regard the similarity functions they use and other implementational decisions as proprietary information and do not make them available to the general public.

As a direct consequence of the autonomy of component search engines, a number of heterogeneities exist. In this section, we first identify major heterogeneities that are unique in the metasearch engine environment. Heterogeneities that are common to other automonous systems (e.g. multidatabase systems) such as different OS platforms will not be described. Then we discuss the impact of these heterogeneities as well as the autonomy of component search engines on building an effective and efficient metasearch engine.

### 4.1 Heterogeneous Environment

The following heterogeneities can be identified among autonomous component search engines [Meng et al. 1999b].

**Indexing Method:** Different search engines may have different ways to determine what terms should be used to represent a given document. For example, some may consider all terms in the document (i.e. *full-text indexing*) while others may use only a subset of the terms (i.e. *partial-text indexing*. Lycos [Mauldin 1997], for example, employs partial-text indexing.) in order to save storage space and be more scalable. Some search engines on the Web use the *anchor terms* in a web page to index the referenced web page [Brin and Page 1998; Cutler et al. 1997; McBryan 1994] while most other search engines do not. Other examples of different indexing techniques involve whether or not to remove *stopwords* and whether or not to perform *stemming*. Furthermore, different stopword lists and stemming algorithms may be used by different search engines.

**Document Term Weighting Scheme:** Different methods exist for determining the weight of a term in a document. For example, one method is to use the *term frequency weight* and another is to use the product of the term frequency weight and the *inverse document frequency weight* (see Section 2). Several variations of these schemes exist [Salton 1989]. There are also systems that distinguish different occurrences of the same term [Boyan et al. 1996; Cutler et al. 1997; Wade et al. 1989] or different fonts of the same term [Brin and Page 1998]. For example, the occurrence of a term appearing in the title of a web page may be considered to be more important than another occurrence of the same term not appearing in the title.

**Query Term Weighting Scheme:** In the vector space model for text retrieval, a query can be considered as a special document (a very short document typically). It is possible for a term to appear multiple times in a query. Different query term weighting schemes may utilize the frequency of a term in a query differently for computing the weight of the term in the query.

**Similarity Function:** Different search engines may employ different similarity functions to measure the similarity between a user query and a document. Some popular similarity functions were mentioned in Section 2 but other similarity functions, see for example [Robertson et al. 1999; Singhal et al. 1996], are also possible.

**Document Database:** The text databases of different search engines may differ at two levels. The first level is the *domain* (subject area) of a database. For example, one database may contain medical documents (e.g. www.medisearch.co.uk) and another may contain legal documents (e.g. lawcrawler.lp.findlaw.com). In this case, the two databases can be said to have different domains. In practice, the domain of a database may not be easily determined since some databases may contain documents from multiple domains. Furthermore, a domain may be further divided into multiple subdomains. The second level is the *set* of documents. Even when two databases have the same domain, the sets of documents in the two databases can still be substantially different or even disjoint. For example, Echidna Medical Search (www.drsref.com.au) and Medisearch (www.medisearch.co.uk) are both search engines for medical information but the former is for Web pages from Australia and the latter from the United Kingdom.

**Document Version:** Documents in a database may be modified. This is especially true in the World Wide Web environment where Web pages can often be modified at the wish of their authors. Typically, when a Web page is modified, those search engines that indexed the Web page will not be notified of the modification. Some search engines use *robots* to detect modified pages and re-index them. However, due to the high cost and/or the enormous amount of work involved, attempts to revisit a page can only be made periodically (say from one week to one month). As a result, depending on when a document is fetched (or refetched) and indexed (or reindexed), its representation in a search engine may be based on an older version or a newer version of the document. Since local search engines are autonomous, it is highly likely that different systems may have indexed different versions of the same document (in the case of WWW, the web page can still be uniquely identified by its URL).

**Result Presentation:** Almost all search engines present their retrieval result in descending order of local similarities/ranking scores. However, some search engines also provide the similarities of returned documents (e.g. FirstGov (www.firstgov.gov) and Northern Light) while some do not (e.g. AltaVista and Google).

In addition to heterogeneities between component search engines, there are also heterogeneities between the metasearch engine and the local systems. For example, the metasearch engine uses a global similarity function to compute the global similarities of documents. It is very likely that the global similarity function is different from the similarity functions in some (or even all) component search engines.

### 4.2 Impact of Heterogeneities

In this subsection, we show that the autonomy of and the heterogeneities among different component search engines and between the metasearch engine and the component search engines have a profound impact on how to evaluate global queries in a metasearch engine.

(1) In order to estimate the usefulness of a database to a given query, the database selector needs to know some information about the database that characterizes the contents of the database. We call the characteristic information about a

database the *representative* of the database. In the metasearch engine environment, different types of database representatives for different search engines may be available to the metasearch engine. For cooperative search engines, they may provide database representatives desired by the database selector. For uncooperative search engines that follow a certain standard, say the proposed STARTS standard [Gravano et al. 1997], the database representatives 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. But the representatives may not contain certain information desired by a particular database selector. For uncooperative search engines that do not follow any standard, their representatives may have to be extracted from past retrieval experiences (e.g. SavvySearch [Dreilinger and Howe 1997]) or from sampled documents (e.g. [Callan et al. 1999; Callan 2000]).

There are two major challenges in developing good database selection algorithms. One is to identify appropriate database representatives. A good representative should permit fast and accurate estimation of database usefulness. At the same time, a good representative should have a small size in comparison to the size of the database and should be easy to obtain and maintain. As we will see in Section 5, proposed database selection algorithms often employ different types of representatives. The second challenge is to develop ways to obtain the desired representatives. As mentioned above, a number of solutions exist depending on whether a search engine follows some standard or is cooperative. The issue of obtaining the desired representatives will not be discussed further in this article.

(2) The challenges of the document selection problem and the result merging problem lie mainly in the fact that the same document may have different global and local similarities with a given query due to various heterogeneities. For example, for a given query $q$ submitted by a global user, whether or not a document $d$ in a component database $D$ is potentially useful depends on the global similarity of $d$ with $q$. It is highly likely that the similarity function and/or the term weighting scheme in $D$ are different from the global ones. As a result, the local similarity of $d$ is likely to be different from the global similarity of $d$. In fact, even when the same term weighting scheme and the same similarity function are used locally and globally, the global similarity and the local similarity of $d$ may still be different because the similarity computation may make use of certain database-specific information (such as the document frequencies of terms). This means that a globally highly ranked document in $D$ may not be a locally highly ranked document in $D$. Suppose the globally top ranked document $d$ is ranked i-th locally for some $i \geq 1$. In order to retrieve $d$ from $D$, the local system may have to also retrieve all documents that have a higher local similarity than that of $d$ (text retrieval systems are generally incapable of retrieving lower ranked documents without first retrieving higher ranked ones). It is quite possible that some of the documents that are ranked higher than $d$ locally are not potentially useful based on their global similarities. The main challenge for document selection is to develop methods that can maximize the retrieval of potentially useful documents while minimizing the retrieval of useless documents from component search engines. The main challenge for result

merging is to find ways to estimate the global similarities of documents so that documents returned from different component search engines can be properly merged.

In the next several sections, we examine the techniques that have been proposed to deal with the problems of database selection, document selection and result merging.

## 5. DATABASE SELECTION

When a metasearch engine receives a query from a user, it invokes the database selector to select component search engines to send the query to. A good database selection algorithm should identify potentially useful databases accurately. Many approaches have been proposed to tackle the database selection problem. These approaches differ on the database representatives they use to indicate the contents of each database, the measures they use to indicate the usefulness of each database with respect to a given query and the techniques they employ to estimate the usefulness. We classify these approaches into the following three categories.

**Rough representative approaches:** In these approaches, the contents of a local database are often represented by a few selected key words or paragraphs. Such a representative is only capable of providing a very general idea on what a database is about and consequently database selection methods using rough database representatives are not very accurate in estimating the true usefulness of databases with respect to a given query. Rough representatives are often manually generated.

**Statistical representative approaches:** These approaches usually represent the contents of a database using rather detailed statistical information. Typically, the representative of a database contains some statistical information for each term in the database such as the *document frequency* of the term and the average weight of the term among all documents that have the term. Detailed statistics allow more accurate estimation of database usefulness with respect to any user query. Scalability of such approaches is an important issue due to the amount of information that needs to be stored for each database.

**Learning-based approaches:** In these approaches, the knowledge about which databases are likely to return useful documents to what types of queries is learned from past retrieval experiences. Such knowledge is then used to determine the usefulness of databases for future queries. The retrieval experiences could be obtained through the use of training queries before the database selection algorithm is put to use and/or through the real user queries while database selection is in active use. The obtained experiences against a database will be saved as the representative of the database.

In the following subsections, we survey and discuss different database selection approaches based on the above classification.

### 5.1 Rough Representative Approaches

As mentioned earlier, a rough representative of a database uses only a few key words or a few sentences to describe the contents of the database. It is only capable of providing a very general idea on what the database is about.

In ALIWEB [Koster 1994], an often human-generated representative in a fixed format is used to represent the contents of each local database or a site. An example of the representative used to describe a site containing files for the Perl Programming Language is as follows (www.nexor.com/site.idx):

```
Template-Type:  DOCUMENT
Title:          Perl
URI:            /public/perl/perl.html
Description:    Information on the Perl Programming Language.
                Includes a local Hypertext Perl Manual,
                and the latest FAQ in Hypertext.
Keywords:       perl, perl-faq, language
Author-Handle:  m.koster@nexor.co.uk
```

The user query is matched with the representative of each component database to determine how suitable a database is for the query. The match can be against one or more fields (e.g. title, description, etc) of the representatives based on the user's choice. Component databases are ranked based on how closely they match with the query. The user then selects component databases to search from a ranked list of component databases, one database at a time. Note that ALIWEB is not a full blown metasearch engine as it only allows users to select one database to search at a time and it does not perform result merging.

Similar to ALIWEB, descriptive representations of the contents of component databases are also used in WAIS [Kahle and Medlar 1991]. For a given query, the descriptions are used to rank component databases according to how similar they are to the query. The user then selects component databases to search for the desired documents. In WAIS, more than one local database can be searched at the same time.

In Search Broker [Manber and Bigot 1997; Manber and Bigot 1998], each database is manually assigned one or two words as the subject or category keywords. Each user query consists of two parts: the subject part and the regular query part. When a query is received by the system, the subject part of the query is used to identify the component search engines covering the same subject and the regular query part is used to search documents from the identified search engines.

In NetSerf [Chakravarthy and Haase 1995], the text description of the contents of a database is transformed into a structured representative. The transformation is performed manually and WordNet [Miller 1990] is used in the transformation process to disambiguate topical words. As an example, the description "World facts listed by country" for the *World Factbook* archive is transformed into the following structured representation [Chakravarthy and Haase 1995]:

```
topic:   country
         synset: [nation, nationality, land, country, a_people]
         synset: [state, nation, country, land, commonwealth,
                  res_publica, body_politic]
         synset: [country, state, land, nation]
info-type: facts
```

Each word in WordNet has one or more synsets with each containing a set of synonyms that together defines a meaning. The topical word "country" has four synsets of which three are considered to be relevant, and are therefore used. The one synset (i.e. [rural_area, country]) whose meaning does not match the intended meaning of the "country" in the above description (i.e. "World facts listed by country") is omitted. Each user query is a sentence and is automatically converted into a structured and disambiguated representation similar to a database representation using a combination of several techniques. However, not all queries can be handled. The query representation is then matched with the representatives of local databases in order to identify potentially useful databases [Chakravarthy and Haase 1995].

While most rough database representatives are generated with human involvement, there exist automatically generated rough database representatives. In Q-Pilot [Sugiura and Etzioni 2000], each database is represented by a vector of terms with weights. The terms can either be obtained from the interface page of the search engine or from the pages that have links to the search engine. In the former case, all content words in the interface page are considered and the weights are the term frequencies. In the latter case, only terms that appear in the same line as the link to the search engine are used and the weight of each term is the document frequency of the term (i.e. the number of back link documents that contributed the term).

The main appeal of rough representative approaches is that the representatives can be obtained relatively easily and they require little storage space. If all component search engines are highly specialized with diversified topics and their contents can be easily summarized, then these approaches may work reasonably well. On the other hand, it is unlikely that the short description of a database can represent the database sufficiently comprehensively, especially when the database contains documents of diverse interests. As a result, missing potentially useful databases can occur easily with these approaches. To alleviate this problem, most such approaches involve users in the database selection process. For example, in ALIWEB and WAIS, users will make the final decision on which databases to select based on the preliminary selections by the metasearch engine. In Search Broker, users are required to specify the subject areas for their queries. As users often do not know the component databases well, their involvement in the database selection process can easily miss useful databases. Rough representative approaches are considered to be inadequate for large-scale metasearch engines.

### 5.2 Statistical Representative Approaches

A statistical representative of a database typically takes every term in every document in the database into consideration and keeps one or more pieces of statistical information for each such term. As a result, if done properly, a database selection approach employing this type of database representatives may detect the existence of individual potentially useful documents for any given query. A large number of approaches based on statistical representatives have been proposed. In this subsection, we describe five such approaches.

5.2.1 *D-WISE approach.* WISE (Web Index and Search Engine) is a centralized search engine [Yuwono and Lee 1996]. D-WISE is a proposed metasearch engine with a number of underlying search engines (i.e. distributed WISE) [Yuwono and Lee 1997]. In D-WISE, the representative of a component search engine consists of the document frequency of each term in the component database as well as the number of documents in the database. Therefore, the representative of a database with $n$ distinct terms will contain $n + 1$ quantities (the $n$ document frequencies and the cardinality of the database) in addition to the $n$ terms. Let $n_i$ denote the number of documents in the i-th component database and $df_{ij}$ be the document frequency of term $t_j$ in the i-th database.

Suppose $q$ is a user query. The representatives of all databases are used to compute the ranking score of each component search engine with respect to $q$. The scores measure the relative usefulness of all databases with respect to $q$. If the score of database A is higher than that of database B, then database A will be judged to be more relevant to $q$ than database B. The ranking scores are computed as follows. First, the *cue validity* of each query term, say term $t_j$, for the i-th component database, $CV_{ij}$, is computed using the following formula:

$$CV_{ij} = \frac{\frac{df_{ij}}{n_i}}{\frac{df_{ij}}{n_i} + \frac{\sum_{k \neq i}^{N} df_{kj}}{\sum_{k \neq i}^{N} n_k}} \tag{3}$$

where $N$ is the total number of component databases in the metasearch engine. Intuitively, $CV_{ij}$ measures the percentage of the documents in the i-th database that contain term $t_j$ relative to that in all other databases. If the i-th database has a higher percentage of documents containing $t_j$ in comparison to other databases, then $CV_{ij}$ tends to have a larger value. Next, the *variance* of the $CV_{ij}$'s of each query term $t_j$ for all component databases, $CVV_j$, is computed as follows:

$$CVV_j = \frac{\sum_{i=1}^{N} (CV_{ij} - ACV_j)^2}{N} \tag{4}$$

where $ACV_j$ is the average of all $CV_{ij}$'s for all component databases. The value $CVV_j$ measures the skew of the distribution of term $t_j$ across all component databases. For two terms $t_u$ and $t_v$, if $CVV_u$ is larger than $CVV_v$, then term $t_u$ is more useful to distinguish different component databases than term $t_v$. As an extreme case, if every database had the same percentage of documents containing a term, then the term would not be very useful for database selection (the $CVV$ of the term would be zero in this case). Finally, the ranking score of component database $i$ with respect to query $q$ is computed by:

$$r_i = \sum_{j=1}^{M} CVV_j \cdot df_{ij} \tag{5}$$

where $M$ is the number of terms in the query. It can be seen that the ranking score of database $i$ is the sum of the document frequencies of all query terms in the database weighted by each query term's $CVV$ (recall that the value of $CVV$ for a term reflects the distinguishing power of the term). Intuitively, the ranking scores provide clues to where useful query terms are concentrated. If a database has many useful query terms, each having a higher percentage of documents than other databases, then the ranking score of the database will be high. After the ranking scores of all databases are computed with respect to a given query, the databases with the highest scores will be selected for search for this query.

The representative of a database in D-WISE contains one quantity, i.e., the document frequency, per distinct term in the database, plus one additional quantity, i.e., the cardinality, for the entire database. As a result, this approach is easily scalable. The computation is also simple. However, there are two problems with this approach. First, the ranking scores are relative scores. As a result, it will be difficult to determine the real value of a database with respect to a given query. If there are no good databases for a given query, then even the first ranked database will have very little value. On the other hand, if there are many good databases for another query, then even the 10th ranked database can be very useful. Relative ranking scores are not very useful in differentiating these situations. Second, the accuracy of this approach is questionable as this approach does not distinguish a document containing, say, one occurrence of a term from a document containing 100 occurrences of the same term.

5.2.2 *CORI Net approach.* In the Collection Retrieval Inference Network (CORI Net) approach [Callan et al. 1995], the representative of a database consists of two pieces of information for each distinct term in the database: the *document frequency* and the *database frequency.* The latter is the number of component databases containing the term. Note that if a term appears in multiple databases, only one database frequency needs to be stored in the metasearch engine to save space.

In CORI Net, for a given query $q$, a document ranking technique known as *inference network* [Turtle and Croft 1991] used in the INQUERY document retrieval system [Callan et al. 1992] is extended to rank all component databases with respect to $q$. The extension is mostly conceptual and the main idea is to visualize the representative of a database as a (super) document and the set of all representatives as a collection/database of super documents. This is explained below. The representative of a database may be conceptually considered as a super document containing all distinct terms in the database. If a term appears in $k$ documents in the database, we repeat the term $k$ times in the super document. As a result, the document frequency of a term in the database becomes the *term frequency* of the term in the super document. The set of all super documents of the component databases in the metasearch engine form a database of super documents. Let $D$ denote this database of all super documents. Note that the database frequency of a term becomes the document frequency of the term in $D$. Therefore, from the representatives of component databases, we can obtain the term frequency and document frequency of each term in each super document. In principle, the $tfw \cdot idfw$ (term frequency weight times inverse document frequency weight) formula could now be used to compute the weight of each term in each super document so as to represent each super document as a vector of weights. Furthermore, a similarity function such as the *Cosine* function may be used to compute the similarities (ranking scores) of all super documents (i.e. database representatives) with respect to query $q$ and these similarities could then be used to rank all component databases.

The approach employed in CORI Net is an inference network based probabilistic approach.

In CORI Net, the ranking score of a database with respect to query $q$ is an estimated belief that the database contains useful documents. The belief is essentially the combined probability that the database contains useful documents due to each query term. More specifically, the belief is computed as follows. Suppose the user query contains $k$ terms $t_1, ..., t_k$. Let $N$ be the number of databases in the metasearch engine. Let $df_{ij}$ be the document frequency of the j-th term in the i-th component database $D_i$ and $dbf_j$ be the database frequency of the j-th term. First, the belief that $D_i$ contains useful documents due to the j-th query term is computed by:

$$p(t_j|D_i) = c_1 + (1 - c_1) \cdot T_{ij} \cdot I_j \tag{6}$$

where

$$T_{ij} = c_2 + (1 - c_2) \cdot \frac{df_{ij}}{df_{ij} + K}$$

is a formula for computing the term frequency weight of the j-th term in the super document corresponding to $D_i$ and

$$I_j = \frac{log(\frac{N+0.5}{dbf_j})}{log(N + 1.0)}$$

is a formula for computing the inverse document frequency weight of the j-th term based on all super documents. In the above formulas, $c_1$ and $c_2$ are constants between 0 and 1, and $K = c_3 \cdot ((1 - c_4) + c_4 \cdot dw_i/adw)$ is a function of the size of database $D_i$ with $c_3$ and $c_4$ being two constants, $dw_i$ being the number of words in $D_i$ and $adw$ being the average number of words in a database. The values of these constants ($c_1, c_2, c_3$ and $c_4$) can be determined empirically by performing experiments on actual test collections [Callan et al. 1995]. Note that the value of $p(t_j|D_i)$ is essentially the $tfw \cdot idfw$ weight of term $t_j$ in the super document corresponding to database $D_i$. Next, the significance of term $t_j$ in representing query $q$, denoted $p(q|t_j)$, can be estimated, for example, to be the query term weight of $t_j$ in $q$. Finally, the belief that database $D_i$ contains useful documents with respect to query $q$, or the ranking score of $D_i$ with respect to $q$, can be estimated to be:

$$r_i = p(q|D_i) = \sum_{j=1}^{k} p(q|t_j) \cdot p(t_j|D_i) \tag{7}$$

In CORI Net, the representative of a database contains slightly more than 1 piece of information per term (i.e. the *document frequency* plus the shared *database frequency* across all databases). Therefore, the CORI Net approach also has rather good scalability. The information for representing each component database can also be obtained and maintained easily. An advantage of the CORI Net approach is that the same method can be used to compute the ranking score of a document with a query as well as the ranking score of a database (through the database representative or super document) with a query. Recently, it was shown in [Xu and Callan

1998] that if phrase information is collected and stored in each database representative and queries are expanded based on a technique called *local context analysis* [Xu and Croft 1996], then the CORI Net approach can select useful databases more accurately.

5.2.3 *gGlOSS approach.* The gGlOSS (generalized Glossary Of Servers' Server) system is a research prototype [Gravano and Garcia-Molina 1995]. In gGlOSS, each component database is represented by a set of pairs $(df_i, W_i)$, where $df_i$ is the document frequency of the i-th term and $W_i$ is the sum of the weights of the i-th term over all documents in the component database. A threshold is associated with each query in gGlOSS to indicate that only documents whose similarities with the query are higher than the threshold are of interest. The usefulness of a component database with respect to a query in gGlOSS is defined to be the sum of the similarities of the documents in the component database with the query that are higher than the threshold associated with the query. The usefulness of a component database is used as the ranking score of the database. In gGlOSS, two estimation methods are employed based on two assumptions. One is the *high-correlation assumption* (for any given database, if query term $t_i$ appears in at least as many documents as query term $t_j$, then every document containing term $t_j$ also contains term $t_i$) and the other is the *disjoint assumption* (for a given database, for any two terms $t_i$ and $t_j$, the set of documents containing term $t_i$ is disjoint from the set of documents containing term $t_j$).

We now discuss the two estimation methods for a component database D. Suppose $q = (q_1, ..., q_k)$ is a query and T is the associated threshold, where $q_i$ is the weight of term $t_i$ in $q$.

**High-correlation case:** Let terms be arranged in ascending order of document frequency, i.e., $df_i \leq df_j$ for any $i < j$, where $df_i$ is the document frequency of term $t_i$. This means that every document containing $t_i$ also contains $t_j$ for any $j > i$. There are $df_1$ documents having similarity $\sum_{i=1}^{k} q_i \cdot \frac{W_i}{df_i}$ with $q$. In general, there are $df_j - df_{j-1}$ documents having similarity $\sum_{i=j}^{k} q_i \cdot \frac{W_i}{df_i}$ with $q$, $1 \leq j \leq k$ and $df_0$ is defined to be 0. Let $p$ be an integer between 1 and $k$ that satisfies $\sum_{i=p}^{k} q_i \cdot \frac{W_i}{df_i} > T$ and $\sum_{i=p+1}^{k} q_i \cdot \frac{W_i}{df_i} \leq T$. Then the estimated usefulness of this database is:

$$usefulness(D, q, T) = \sum_{j=1}^{p} (df_j - df_{j-1}) \cdot \left(\sum_{i=j}^{k} q_i \cdot \frac{W_i}{df_i}\right) = \sum_{j=1}^{p} q_j \cdot W_j + df_p \cdot \sum_{j=p+1}^{k} q_j \cdot \frac{W_j}{df_j}$$

**Disjoint case:** By the disjoint assumption, each document can contain at most one query term. Thus, there are $df_i$ documents that contain term $t_i$ and the similarity of these $df_i$ documents with query $q$ is $q_i \cdot \frac{W_i}{df_i}$. Therefore, the estimated usefulness of this database is:

$$usefulness(D, q, T) = \sum_{i=1,\ldots,k|(df_i>0)\wedge(q_i \cdot \frac{W_i}{df_i})>T} df_i \cdot q_i \cdot \frac{W_i}{df_i}$$

$$= \sum_{i=1,\ldots,k|(df_i>0)\wedge(q_i \cdot \frac{W_i}{df_i})>T} q_i \cdot W_i$$

In gGlOSS, the usefulness of a database is sensitive to the similarity threshold used. As a result, gGlOSS can differentiate a database with many moderately similar documents from a database with a few highly similar documents. This is not possible in D-WISE and CORI Net. However, the two assumptions used in gGlOSS are somewhat too restrictive. As a result, the estimated database usefulness may be inaccurate. It can be shown that, when the threshold $T$ is not too large, the estimation formula based on the high-correlation assumption tends to overestimate the usefulness and the estimation formula based on the disjoint assumption tends to underestimate the usefulness. Since the two estimates by the two formulas tend to form upper and lower bounds of the true usefulness, the two methods are more useful when used together than when used separately. For a given database, the size of the database representative in gGlOSS is twice the size of that in D-WISE. The computation for estimating the database usefulness in gGlOSS can be carried out efficiently.

5.2.4 *Estimating the number of potentially useful documents.* One database usefulness measure used is "the number of potentially useful documents with respect to a given query in a database". This measure can be very useful for search services that charge a fee for each search. For example, the Chicago Tribune Newspaper Company charges a certain fee to retrieving archival newspaper articles. Suppose the fee is independent of the number of retrieved documents. In this case, from the user's perspective, a component system which contains a large number of similar documents but not necessarily the most similar documents is preferable to another component system containing just a few most similar documents. On the other hand, if a fee is charged for each retrieved document, then the component system having the few most similar documents will be preferred. This type of charging policy can be incorporated into the database selector of a metasearch engine if the number of potentially useful documents in a database with respect to a given query can be estimated.

Let $D$ be a component database, $sim(q, d)$ be the global similarity between a query $q$ and a document $d$ in $D$, and $T$ be a similarity threshold. The number of potentially useful documents in $D$ with respect to $q$ can be defined precisely as follows:

$$NoDoc(D, q, T) = cardinality(\{d | d \in D \ and \ sim(q, d) > T\}) \qquad (8)$$

If $NoDoc(D, q, T)$ can be accurately estimated for each database with respect to a given query, then the database selector can simply select those databases with the most potentially useful documents to search for this query.

In [Meng et al. 1998], a generating-function based method is proposed to estimate $NoDoc(D, q, T)$ when the global similarity function is the dot product function (the widely used $Cosine$ function is a special case of the dot product function with each term weight divided by the document/query length). In this method, the representative of a database with $n$ distinct terms consists of $n$ pairs $\{(p_i, w_i)\}$, $i = 1, ..., n$, where $p_i$ is the probability that term $t_i$ appears in a document in $D$ (note that $p_i$ is simply the document frequency of term $t_i$ in the database divided by the number of documents in the database) and $w_i$ is the average of the weights of $t_i$ in the set of documents containing $t_i$. Let $(q_1, q_2, ..., q_k)$ be the query vector of query $q$, where $q_i$ is the weight of query term $t_i$.

Consider the following generating function:

$$(p_1 * X^{w_1 * q_1} + (1 - p_1)) * (p_2 * X^{w_2 * q_2} + (1 - p_2)) * \cdots * (p_k * X^{w_k * q_k} + (1 - p_k)) \quad (9)$$

After the generating function (9) is expanded and the terms with the same $X^s$ are combined, we obtain

$$a_1 * X^{b_1} + a_2 * X^{b_2} + \cdots + a_c * X^{b_c}, \; b_1 > b_2 > ... > b_c \quad (10)$$

It can be shown that, if the terms are independent and the weight of term $t_i$ whenever present in a document is $w_i$, which is given in the database representative $(1 \leq i \leq k)$, then $a_i$ is the probability that a document in the database has similarity $b_i$ with $q$ [Meng et al. 1998]. Therefore, if database $D$ contains $N$ documents, then $N * a_i$ is the expected number of documents that have similarity $b_i$ with query $q$. For a given similarity threshold $T$, let C be the largest integer to satisfy $b_C > T$. Then, $NoDoc(D, q, T)$ can be estimated by the following formula:

$$NoDoc(D, q, D) = \sum_{i=1}^{C} N * a_i = N \sum_{i=1}^{C} a_i \quad (11)$$

The above solution has two restrictive assumptions. The first is the *term independence assumption* and the second is the *uniform term weight assumption* (i.e. the weights of a term in all documents containing the term are the same — the average weight). These assumptions reduce the accuracy of the database usefulness estimation. One way to address the term independence assumption is to utilize *covariances* between term pairs, term triplets, and so on and to incorporate them into the generating-function (9) [Meng et al. 1998]. The problem with this approach is that the storage overhead for representing a component database may become too large because a very large number of covariances may be associated with each component database. A remedy is to use only significant covariances (those whose absolute values are significantly greater than zero). Another way to incorporate dependencies between terms is to combine certain adjacent terms into a single term [Liu et al. 2001]. This is similar to recognizing phrases.

In [Meng et al. 1999a], a method known as the *subrange-based estimation method* is proposed to deal with the uniform term weight assumption. This method partitions the actual weights of a term $t_i$ in the set of documents having the term into a number of disjoint subranges of possibly different lengths. For each subrange, the median of the weights in the subrange is estimated based on the assumption that the weight distribution of the term is *normal* (hence, the standard deviation of the weights of the term needs to be added to the database representative). Then, the weights of $t_i$ that fall in a given subrange are approximated by the median of the weights in the subrange. With this weight approximation, for a query containing term $t_i$, the polynomial $p_i * X^{w_i * q_i} + (1 - p_i)$ in the generating function (9) is replaced by the following polynomial:

$$p_{i1} * X^{wm_{i1} * q_i} + p_{i2} * X^{wm_{i2} * q_i} + \cdots + p_{il} * X^{wm_{il} * q_i} + (1 - p_i) \quad (12)$$

where $p_{ij}$ is the probability that term $t_i$ occurs in a document and has a weight in the j-th subrange, $wm_{ij}$ is the median of the weights of $t_i$ in the j-th subrange, $j = 1, ..., l$, and $l$ is the number of subranges used. After the generating function has been obtained, the rest of the estimation process is identical to that described earlier. It was shown in [Meng et al. 1999a] that if the maximum normalized weight of each term is used in the highest subrange, the estimation accuracy of the database usefulness can be drastically improved.

The above methods [Liu et al. 2001; Meng et al. 1998; Meng et al. 1999a], while being able to produce accurate estimation, have a large storage overhead. Furthermore, the computation complexity of expanding the generating function is exponential. As a result, they are more suitable for short queries.

5.2.5 *Estimating the similarity of the most similar document.* Another useful measure is the global similarity of the most similar document in a database with respect to a given query. On one hand, this measure indicates the best that we can expect from a database as no other documents in the database can have higher similarities with the query. On the other hand, for a given query, this measure can be used to rank databases optimally for retrieving the $m$ most similar documents across all databases.

Suppose a user wants the metasearch engine to find the $m$ most similar documents to his/her query $q$ across $M$ component databases $D_1, D_2, \ldots, D_M$. The following definition defines an optimal order of these databases for the query.

Definition 2. A set of $M$ databases is said to be optimally ranked in the order $[D_1, D_2, ..., D_M]$ with respect to query $q$ if there exists a $k$ such that $D_1, D_2, ..., D_k$ contain the $m$ most similar documents and each $D_i$, $1 \leq i \leq k$, contains at least one of the $m$ most similar documents.

Intuitively, the ordering is optimal because whenever the $m$ most similar documents to the query are desired, it is sufficient to examine the first $k$ databases. A necessary and sufficient condition for the databases $D_1, D_2, ..., D_M$ to be optimally ranked in the order $[D_1, D_2, ..., D_M]$ with respect to query $q$ is $msim(q, D_1) > msim(q, D_2) > ... > msim(q, D_M)$ [Yu et al. 1999b], where $msim(q, D_i)$ is the global similarity of the most similar document in database $D_i$ with the query $q$. Knowing an optimal rank of the databases with respect to query $q$, the database selector can select the top-ranked databases to search for $q$.

The challenge here is how to estimate $msim(q, D)$ for query $q$ and any database $D$. One method is to utilize the Expression (10) for $D$. We can scan this expression in descending order of the exponents until $\sum_{i=1}^{r} a_i * N$ is approximately 1 for some $r$, where $N$ is the number of documents in $D$. The exponent, $b_r$, is an estimate of $msim(q, D)$ as the expected number of documents in $D$ with similarity greater than or equal to $b_r$ is approximately 1. The drawback of this solution is that it requires a large database representative and the computation is of high complexity.

A more efficient method to estimate $msim(q, D)$ is proposed in [Yu et al. 1999b]. In this method, there are two types of representatives. There is a global representative for all component databases. For each distinct term $t_i$, the global inverse document frequency weight ($gidf_i$) is stored in this representative. There is a local representative for each component database $D$. For each distinct term $t_i$ in $D$, a pair of quantities $(mnw_i, anw_i)$ is stored, where $mnw_i$ and $anw_i$ are the *maximum* *normalized weight* and the *average normalized weight* of term $t_i$, respectively. Suppose $d_i$ is the weight of $t_i$ in a document $d$. Then the normalized weight of $t_i$ in $d$ is $d_i/|d|$, where $|d|$ denotes the length of $d$. The *maximum normalized weight* and the *average normalized weight* of $t_i$ in database $D$ are respectively the maximum and the average of the normalized weights of $t_i$ in all documents in $D$. Suppose $q = (q_1, ..., q_k)$ is the query vector. Then $msim(q, D)$ can be estimated as follows:

$$msim(q, D) = \max_{1 \le i \le k} \left\{ q_i * gidf_i * mnw_i + \sum_{\substack{j=1 \\ j \ne i}}^{k} q_j * gidf_j * anw_j \right\} / |q| \qquad (13)$$

The intuition for having this estimate is that the most similar document in a database is likely to have the maximum normalized weight of the i-th query term, for some $i$. This yields the first half of the above expression within the braces. For each of the other query terms, the document takes the average normalized weight. This yields the second half. Then, the maximum is taken over all $i$, since the most similar document may have the maximum normalized weight of any one of the $k$ query terms. Normalization by the query length, $|q|$, yields a value less than or equal to 1. The underlying assumption of Formula (13) is that terms in each query are independent. Dependencies between terms can be captured to a certain extent by storing the same statistics (i.e. *mnw*'s and *anw*'s) of phrases in the database representatives, i.e., treating each phrase as a term.

In this method, each database is represented by two quantities per term plus the global representative shared by all databases but the computation has linear complexity.

The maximum normalized weight of a term is typically two or more orders of magnitude larger than the average normalized weight of the term as the latter is computed over all documents, including those not containing the term. This observation implies that in Formula (13), if all query terms have the same $tf$ weight (a reasonable assumption, as in a typical query, each term appears once), $gidf_i * mnw_i$ is likely to dominate $\sum_{j=1, j \ne i}^{k} gidf_j * anw_j$, especially when the number of terms, $k$, in a query is small (which is typically true in the Internet environment [Jansen et al. 1998; Kirsch 1998]). In other words, the rank of database $D$ with respect to a given query $q$ is largely determined by the value of $\max_{1 \le i \le k} \left\{ q_i * gidf_i * mnw_i \right\}$. This leads to the following more scalable formula to estimate $msim(q, D)$ [Wu et al. 2001]: $\max_{1 \le i \le k} \{q_i * am_i\}/|q|$, where $am_i = gidf_i * mnw_i$ is the *adjusted maximum normalized weight* of term $t_i$ in $D$. This formula requires only one piece of information, namely $am_i$, to be kept in the database representative for each distinct term in the database.

### 5.3 Learning-based Approaches

These approaches predict the usefulness of a database for new queries based on the retrieval experiences with the database from past queries. The retrieval experiences may be obtained in a number of ways. First, *training queries* can be used and the retrieval knowledge of each component database with respect to these training queries can be obtained in advance (i.e. before the database selector is enabled). This type of approach will be called the *static learning* approach as in such an approach, the retrieval knowledge, once learned, will not be changed. The weakness of static learning is that it cannot adapt to the changes of database contents and query pattern. Second, real user queries (in contrast to training queries) can be used and the retrieval knowledge can be accumulated gradually and be updated continuously. This type of approach will be referred to as the *dynamic learning* approach. The problem with dynamic learning is that it may take a while to obtain sufficient knowledge useful to the database selector. Third, static learning and dynamic learning can be combined to form a *combined-learning* approach. In such an approach, initial knowledge may be obtained from training queries but the knowledge is updated continuously based on real user queries. Combined-learning can overcome the weaknesses of the other two learning approaches. In this subsection, we introduce several learning based database selection methods.

5.3.1 *MRDD approach.* The MRDD (Modeling Relevant Document Distribution) approach [Voorhees et al. 1995b] is a static learning approach. During learning, a set of training queries is utilized. Each training query is submitted to every component database. From the returned documents from a database for a given query, all relevant documents are identified and a vector reflecting the distribution of the relevant documents is obtained and stored. Specifically, the vector has the format $<r_1, r_2, ..., r_s>$, where $r_i$ is a positive integer indicating that $r_i$ top-ranked documents must be retrieved from the database in order to obtain $i$ relevant documents for the query. As an example, suppose for a training query $q$ and a component database $D$, 100 documents are retrieved in the order $(d_1, d_2, ..., d_{100})$. Among these documents, $d_1, d_4, d_{10}, d_{17}$ and $d_{30}$ are identified to be relevant. Then the corresponding distribution vector is $<r_1, r_2, r_3, r_4, r_5>$ = <1, 4, 10, 17, 30>.

With the vectors for all training queries and all databases obtained, the database selector is ready to select databases for user queries. When a user query is received, it is compared against all training queries and the $k$ most similar training queries are identified ($k = 8$ performed well as reported in [Voorhees et al. 1995b]). Next, for each database $D$, the average relevant document distribution vector over the $k$ vectors corresponding to the $k$ most similar training queries and $D$ is obtained. Finally, the average distribution vectors are used to select the databases to search and the documents to retrieve. The selection tries to maximize the precision for each recall point.

Example 1. Suppose for a given query $q$, the following three average distribution vectors have been obtained for three component databases:
D1: <1, 4, 6, 7, 10, 12, 17>
D2: <3, 5, 7, 9, 15, 20>
D3: <2, 3, 6, 9, 11, 16>

Consider the case when three relevant documents are to be retrieved. To maximize the precision (i.e. to reduce the retrieval of irrelevant documents), one document should be retrieved from D1 and three documents should be retrieved from D3 (two of the three are supposed to be relevant). In other words, databases D1 and D3 should be selected. This selection yields a precision of 0.75 as three out of the four retrieved documents are relevant. ■

In the MRDD approach, the representative of a component database is the set of distribution vectors for all training queries. The main weakness of this approach is that the learning has to be carried out manually for each training query. In addition, it may be difficult to identify appropriate training queries and the learned knowledge may become less accurate when the contents of the component databases change.

5.3.2 *SavvySearch approach.* SavvySearch (www.search.com) is a metasearch engine employing the dynamic learning approach. In SavvySearch [Dreilinger and Howe 1997], the ranking score of a component search engine with respect to a query is computed based on the past retrieval experience of using the terms in the query. More specifically, for each search engine, a weight vector $(w_1, ..., w_m)$ is maintained by the database selector, where each $w_i$ corresponds to the i-th term in the database of the search engine. Initially, all weights are zero. When a query containing term $t_i$ is used to retrieve documents from a component database $D$, the weight $w_i$ is adjusted according to the retrieval result. If no document is returned by the search engine, the weight is reduced by $1/k$, where $k$ is the number of terms in the query. On the other hand, if at least one returned document is read/clicked by the user (no relevance judgement is needed from the user), then the weight is increased by $1/k$. Intuitively, a large positive $w_i$ indicates that the database $D$ responded well to term $t_i$ in the past and a large negative $w_i$ indicates that $D$ responded poorly to $t_i$.

SavvySearch also tracks the recent performance of each search engine in terms of $h$, the average number of documents returned for the most recent five queries, and $r$, the average response time for the most recent five queries sent to the component search engine. If $h$ is below a threshold $T_h$ (the default is 1), then a penalty $p_h = \frac{(T_h - h)^2}{T_h^2}$ for the search engine is computed. Similarly, if the average response time $r$ is greater than a threshold $T_r$ (the default is 15 seconds), then a penalty $p_r = \frac{(r - T_r)^2}{(r_o - T_r)^2}$ is computed, where $r_o = 45$ (seconds) is the maximum allowed response time before a timeout.

For a new query $q$ with terms $t_1, ..., t_k$, the ranking score of database $D$ is computed by:

$$r(q, D) = \frac{\sum_{i=1}^{k} w_{t_i} \cdot \log(N/f_i)}{\sqrt{\sum_{i=1}^{k} |w_i|}} - (p_h + p_r) \tag{14}$$

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

The overhead of storing the representative information for each local search engine in SavvySearch is moderate (Essentially just one piece of information for each term, i.e., the weight. Only terms that have been used in previous queries need to be considered.). Moderate effort is needed to maintain the information. One weakness of SavvySearch is that it will not work well for new query terms or query terms that have been used only very few times. In addition, the user feedback process employed by SavvySearch is not rigorous and could easily lead to the misidentification of useful databases. Search engine users may have the tendency to check out top-ranked documents for their queries regardless of whether or not these documents are actually useful. This means that term weights in the database representative can easily be modified in a way not consistent with the meaning of the weights. As a result, it is possible that the weight of a term for a database does not sufficiently reflect how well the database will respond to the term.

5.3.3 *ProFusion approach.* ProFusion (www.profusion.com) is a metasearch engine employing the combined learning approach. In ProFusion [Fan and Gauch 1999; Gauch et al. 1996], 13 pre-set categories are utilized in the learning process. 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". A set of terms is associated with each category to reflect the topic of the category. For each category, a set of training queries is identified. The reason for using these categories and dedicated training queries is to learn how well each component database will respond to queries in different categories. For a given category $C$ and a given component database $D$, each associated training query is submitted to $D$. From the top 10 retrieved documents, relevant documents are identified. Then a score reflecting the performance of $D$ with respect to the query and the category $C$ is computed by $c * \frac{\sum_{i=1}^{10} N_i}{10} * \frac{R}{10}$, where $c$ is a constant; $N_i$ is set to $1/i$ if the i-th ranked document is relevant and $N_i$ is set to 0 if the document is not relevant; $R$ is the number of relevant documents in the 10 retrieved documents. It can be seen that this formula captures both the *rank order* of each relevant document and the *precision* of the top 10 retrieved documents. Finally, the scores of all training queries associated with the category $C$ is averaged for database $D$ and this average is the *confidence factor* of the database with respect to the category. At the end of the training, there is a confidence factor for each database with respect to each of the 13 categories.

When a user query $q$ is received by the metasearch engine, $q$ is first mapped to one or more categories. The query $q$ is mapped to a category $C$ if at least one term in $q$ belongs to the set of terms associated with $C$. Now the databases will be ranked based on the sum of the confidence factors of each database with respect to the mapped categories. Let this sum of the confidence factors of a database with respect to $q$ be called the *ranking score* of the database for $q$. In ProFusion, the three databases with the largest ranking scores are selected to search for a given query.

In ProFusion, documents retrieved from selected search engines are ranked based on the product of the local similarity of a document and the ranking score of the database. Let $d$ in database $D$ be the first document read/clicked by the user. If $d$ is not the top ranked document, then the ranking score of $D$ should be increased while the ranking scores of those databases whose documents are ranked higher than $d$ should be reduced. This is carried out by proportionally adjusting the confidence factors of $D$ in mapped categories. For example, suppose for a query

$q$ and a database $D$, two categories $C_1$ and $C_2$ are selected and the corresponding confidence factors are 0.6 and 0.4, respectively. To increase the ranking score of database $D$ by $x$, the confidence factors of $D$ in $C_1$ and $C_2$ are increased by $0.6x$ and $0.4x$, respectively. This ranking score adjustment policy tends to move $d$ higher in the rank if the same query is processed in the future. The rationale behind this policy is that if the ranking scores were perfect, then the top ranked document would be the first to be read by the user.

ProFusion combines static learning and dynamic learning, and as a result, overcomes some problems associated with employing static learning or dynamic learning alone. ProFusion has the following shortcomings. First, the static learning part is still done mostly manually, i.e., selecting training queries and identifying relevant documents are carried out manually. Second, the higher ranked documents from the same database as the first clicked document will remain as higher-ranked documents after the adjustment of confidence factors although they are of no interest to the user. This is a situation where the learning strategy does not help retrieve better documents for a repeating query. Third, the employed dynamic learning method seems to be too simplistic. For example, very little user feedback information is used and users' tendency of selecting the highest ranked document regardless of the relevance of the document is not taken into consideration. One way to alleviate this problem is to use the first clicked document that was read for a "significant" amount of time.

## 6. DOCUMENT SELECTION

After the database selector has chosen the component databases for a given query, the next task is to determine what documents to retrieve from each selected database. A naive approach is to let each selected component search engine return all documents that are retrieved from the search engine. The problem with this approach is that too many documents may be retrieved from the component systems unnecessarily. As a result, this approach will not only lead to higher communication cost but also require more effort from the result merger to identify the best matched documents. This naive approach will not be further discussed in this section.

As noted previously, a component search engine typically retrieves documents in descending order of local similarities. Consequently, the problem of selecting what documents to retrieve from a component database can be translated into one of the following two problems:

(1) Determine the number of documents to retrieve from the component database. If $k$ documents are to be retrieved from a component database, then the $k$ documents with the largest local similarities will be retrieved.

(2) Determine a local threshold for the component database such that a document from the component database is retrieved only if its local similarity with the query exceeds the threshold.

Both problems have been tackled in existing or proposed metasearch engines. For either problem, the goal is always to retrieve all or as many as possible potentially useful documents from each component database while minimizing the retrieval of useless documents. We classify the proposed approaches for the document selection problem into the following four categories.

**User Determination:** The metasearch engine lets the global user determine how many documents to retrieve from each component database.

**Weighted Allocation:** The number of documents to retrieve from a component database depends on the ranking score (or the rank) of the component database relative to the ranking scores (or ranks) of other component databases. As a result, proportionally more documents are retrieved from component databases that are ranked higher or have higher ranking scores.

**Learning-based Approaches:** These approaches determine the number of documents to retrieve from a component database based on past retrieval experiences with the component database.

**Guaranteed Retrieval:** This type of approach aims at guaranteeing the retrieval of all potentially useful documents with respect to any given query.

In the following subsections, we survey and discuss approaches from each of the categories.

### 6.1 User Determination

In MetaCrawler [Selberg and Etzioni 1995; Selberg and Etzioni 1997] and SavvySearch [Dreilinger and Howe 1997], the maximum number of documents to be returned from each component database can be customized by the user. Different numbers can be used for different queries. If a user does not select a number, then a query-independent default number set by the metasearch engine will be used. This approach may be reasonable if the number of component databases is small and the user is reasonably familiar with all of them. In this case, the user can choose an appropriate number of documents to retrieve for each component database and can afford to do so.

If the number of component databases is large, then this method has a serious problem. In this case, it is likely that the user will not be capable of selecting an appropriate number for each component database. Consequently, the user will be forced to choose one number and apply that number to all selected component databases. As the numbers of useful documents in different databases with respect to a given query are likely to be different, this method may retrieve too many useless documents from some component systems on one hand while retrieving too few useful documents from other component systems on the other hand. If $m$ documents are to be retrieved from $N$ selected databases, the number of documents to retrieve from each database may be set to be $\lceil \frac{m}{N} \rceil$ or slightly higher.

### 6.2 Weighted Allocation

For a given query, each component database has a rank (i.e. 1st, 2nd, ...) and a ranking score as determined by the database selection algorithm. Both the rank information and the ranking score information can be used to determine the number of documents to retrieve from different component systems. In principle, weighted allocation approaches attempt to retrieve more documents from component search engines that are ranked higher (or have larger ranking scores).

In D-WISE [Yuwono and Lee 1997], the ranking score information is used. For a given query $q$, let $r_i$ be the ranking score of component database $D_i$, $i = 1, ..., N$, where $N$ is the number of selected component databases for the query. Suppose $m$

documents across all selected component databases are desired. Then the number of documents to retrieve from database $D_i$ is $m \cdot r_i / \sum_{j=1}^{N} r_j$.

In CORI Net [Callan et al. 1995], the rank information is used. Specifically, if a total number of $m$ documents are to be retrieved from $N$ component databases, then $m \cdot \frac{2(1+N-i)}{N(N+1)}$ documents will be retrieved from the i-th ranked component database, $i = 1, ..., N$ (note that $\sum_{i=1}^{N} \frac{2(1+N-i)}{N(N+1)} = 1$). In CORI Net, $m$ could be chosen to be larger than the number of desired documents specified by the global user in order to reduce the likelihood of missing useful documents.

As a special case of the weighted allocation approach, if the ranking score of a component database is the estimated number of potentially useful documents in the database, then the ranking score of a component database can be used as the number of documents to retrieve from the database.

Weighted Allocation is a reasonably flexible and easy-to-implement approach based on good intuition (i.e. retrieve more documents from more highly ranked local databases).

### 6.3 Learning-based Approaches

It is possible to learn how many documents to retrieve from a component database for a given query from past retrieval experiences for similar queries. The following are two learning-based approaches [Towell et al. 1995; Voorhees et al. 1995a; Voorhees et al. 1995b; Voorhees 1996; Voorhees and Tong 1997].

In Section 5.3, we introduced a learning-based method, namely MRDD (Modeling Relevant Document Distribution), for database selection. In fact, this method combines the selection of databases and the determination of what documents to retrieve from databases. For a given query $q$, after the average distribution vectors have been obtained for all databases, the decision on what documents to retrieve from these databases is made to maximize the overall precision. In Example 1, when three relevant documents are desired from the given three databases, this method retrieves the top one document from database D1 and the top three documents from D3.

The second method, QC (Query Clustering), also performs document selection based on past retrieval experiences. Again, a set of training queries is utilized. In the training phase, for each component database, the training queries are grouped into a number of clusters. Two queries are placed in the same cluster if the number of common documents retrieved by the two queries is large. Next, the *centroid* of each query cluster is computed by averaging the vectors of the queries in the cluster. Furthermore, for each component database, a weight is computed for each cluster based on the average number of relevant documents among the top $T$ retrieved documents ($T = 8$ performed well as reported in [Voorhees et al. 1995b]) for each query in the query cluster. For a given database, the weight of a cluster indicates how well the database responds to queries in the cluster. When a user query is received, for each component database, the query cluster whose centroid is most similar to the query is selected. Then the weights associated with all selected query clusters across all databases are used to determine the number of documents to retrieve from each database. Suppose $w_i$ is the weight associated with the selected query cluster for component database $D_i$ and $m$ is the total number of documents desired.

Then the number of documents to retrieve from database $D_i$ is $m \cdot w_i / \sum_{j=1}^{N} w_j$, where $N$ is the number of component databases. It can be seen that this method is essentially a weighted allocation method and the weight of a database for a given query is the learned weight of the selected query cluster for the database.

For user queries that have very similar training queries, the above approaches may produce very good results. However, these approaches also have serious weaknesses that may prevent them from being used widely. First, they may not be suitable in environments where new component search engines may be frequently added to the metasearch engine because new training needs to be conducted whenever a new search engine is added. Second, it may not be easy to determine what training queries are appropriate to use. On the one hand, we would like to have some similar training queries for each potential user query. On the other hand, having too many training queries would consume a lot of resources. Third, it is too time consuming for users to identify relevant documents for a wide variety of training queries.

### 6.4 Guaranteed Retrieval

Since the similarity function used in a component database may be different from that used in the metasearch engine, it is possible for a document with low local similarity to have a high global similarity, and vice versa. In fact, even when the global and local similarity functions are identical, this scenario regarding local and global similarities may still occur due to the use of some database-specific statistical information in these functions. For example, the document frequency of a term in a component system is probably very different from that across all systems (i.e. the global document frequency). Consequently, if a component system only returns documents with high local similarities, globally potentially useful documents which are determined based on global similarities from the component database may be missed. The guaranteed retrieval approach tries to ensure that all globally potentially useful documents would be retrieved even when the global and local document similarities do not match. Note that none of the approaches in earlier subsections belongs to the guaranteed retrieval category because they do not take global similarities into consideration.

Many applications, especially those in medical and legal Fields, often desire to retrieve all documents (cases) that are similar to a given query (case). For these applications, the guaranteed retrieval approaches that can minimize the retrieval of useless documents would be appropriate. In this subsection, we introduce some proposed techniques in the guaranteed retrieval category.

6.4.1 *Query Modification.* Under certain conditions, a global query can be modified before it is submitted to a component database to yield the global similarities for returned documents. This technique is called *query modification* [Meng et al. 1998]. It is essentially a query translation method for vector queries. Clearly, if a component system can be tricked into returning documents in descending order of global similarities, guaranteeing the retrieval of globally most similar documents becomes trivial.

Let $D$ be a component database. Consider the case when both the local and the global similarity functions are the *Cosine* function [Salton and McGill 1983]. Note that although the same similarity function is used globally and locally, the same document may still have different global and local similarities due to the use of different local and global document frequencies of terms. Let $d = (w_1, \ldots, w_r)$ be the weight vector of a document in $D$. Suppose each $w_i$ is computed using only information in $d$ (such as term frequency) while a query may use both the term frequency and the inverse document frequency information. The *idf* information for each term in $D$ is incorporated into the similarity computation by modifying each query before it is processed [Buckley et al. 1993]. Consider a user query $q = (q_1, \ldots, q_r)$, where $q_j$ is the weight of term $t_j$ in the query, $j = 1, ..., r$. It is assumed that $q_j$ is either assigned by the user or computed using the term frequency of $t_j$ in the query. When the component system receives the query $q$, it first incorporates the local *idf* weight of each query term by modifying query $q$ to

$$q' = (q_1 * l_1, \ldots, q_r * l_r) \tag{15}$$

and then evaluates the modified query, where $l_j$ is the local *idf* weight of term $t_j$ in component system $D$, $j = 1, ..., r$. As a result, when the *Cosine* function is used, the local similarity of $d$ with $q$ in $D$ can be computed to be $sim_D(q, d) = (\sum_{j=1}^{r} q_j * l_j * w_j)/(|q'| \cdot |d|)$, where $|q'|$ and $|d|$ are the lengths of $q'$ and $d$, respectively. Let $l'_j$ be the global *idf* weight of term $t_j$. Then, when the *Cosine* function is used, the global similarity of $d$ with $q$ should be $sim_G(q, d) = (\sum_{j=1}^{r} q_j * l'_j * w_j)/(|q''| \cdot |d|)$, where $q'' = (q_1 * l'_1, \ldots, q_r * l'_r)$. In order to trick the component system $D$ into computing the global similarity for $d$, the following procedure is used. When query $q = (q_1, \ldots, q_r)$ is received by the metasearch engine, it is first modified to $q^* = (q_1 * (l'_1/l_1), \ldots, q_r * (l'_r/l_r))$. Then the modified query $q^*$ is sent to the component database $D$ for evaluation. According to (15), after $D$ receives $q^*$, it further modifies $q^*$ to $(q_1 * (l'_1/l_1) * l_1, \ldots, q_r * (l'_r/l_r) * l_r) = (q_1 * l'_1, \ldots, q_r * l'_r) = q''$. Finally, $q''$ is evaluated by $D$ to compute the global similarity of $d$ with $q$.

Unfortunately, query modification is not a technique that can work for any combinations of local and global similarity functions. In general, we still need to deal with the situations when documents have different local and global similarities. Furthermore, this approach requires knowledge of the similarity function and the term weighting formula used in a component system. The information is likely to be proprietary and may not be easily available. A study of discovering such information based on sampling queries is reported in [Liu et al. 2000].

6.4.2 *Computing the Tightest Local Threshold.* For a given query $q$, suppose the metasearch engine sets a threshold $T$ and uses a global similarity function $G$ such that any document $d$ that satisfies $G(q, d) > T$ is to be retrieved (i.e. the document is considered to be potentially useful). The problem is to determine a proper threshold $T'$ for each selected component database $D$ such that all potentially useful documents that exist in $D$ can be retrieved using its local similarity function $L$. That is, if $G(q, d) > T$, then $L(q, d) > T'$ for any document $d$ in $D$. Note that in order to guarantee that all potentially useful documents be retrieved from $D$, some unwanted documents from $D$ may also have to be retrieved. The challenge is to minimize the number of documents to retrieve from $D$ while still guaranteeing that all potentially useful documents from $D$ be retrieved. In other words, it is desirable to determine the tightest (largest) local threshold $T'$ such that if $G(q, d) > T$, then $L(q, d) > T'$.

In [Gravano and Garcia-Molina 1997], it is shown that if (1) the similarities computed by $G$ and $L$ are between 0 and 1, and (2) $G$ and $L$ are related by the inequality: $G(q,d) - \epsilon \leq L(q,d)$, where $\epsilon$ is a constant satisfying $0 \leq \epsilon < 1$, then a local threshold $T'$ can be determined. However, the local threshold determined using the method in [Gravano and Garcia-Molina 1997] is often not tight.

In [Meng et al. 1998], several techniques are proposed to find the tightest local threshold for some popular similarity function pairs. For a given global similarity threshold $T$, let $L(T)$ denote the tightest local threshold for a given component database $D$. Then one way to determine $L(T)$ is as follows.

(1) Find the function $f(t)$, the minimum of the local similarity function $L(q,d)$, over all documents $d$ in $D$, subject to $t = G(q,d)$. In this step, $t$ is fixed and $d$ varies over all possible documents in $D$.

(2) Minimize $f(t)$ in the range $t \geq T$. This minimum of $f(t)$ is the desired $L(T)$.

Let $\{t_i\}$ be the set of terms in the query $q$. If both $L(q,d)$ and $G(q,d)$ are differentiable with respect to the weight $w_i$ of each term $t_i$ of document $d$, then finding $f(t)$ in the above step 1 can generally be achieved using the method of Lagrange in calculus [Widder 1989]. Once $f(t)$ is found, its minimum value in the range $t \geq T$ can usually be computed easily. In particular, if $f(t)$ is non-decreasing, then $L(T)$ is simply $f(T)$. The example below illustrates this method.

Example 2. Let $d = (w_1, \ldots, w_r)$ be a document and $q = (u_1, \ldots, u_r)$ be a query. Let the global similarity function $G(q,d) = \sum_{i=1}^{r} u_i \cdot w_i$ and the local similarity function $L(q,d) = (\sum_{i=1}^{r} u_i^p w_i^p)^{\frac{1}{p}}$ (known as $p$-norm in [Salton and McGill 1983]), $p \geq 1$.

Step 1 is to find $f(t)$, which requires us to minimize $(\sum_{i=1}^{r} u_i^p w_i^p)^{\frac{1}{p}}$ subject to $\sum_{i=1}^{r} u_i \cdot w_i = t$. Using the Lagrange method, $f(t)$ is found to be $t \cdot r^{(\frac{1}{p}-1)}$. As this function is an increasing function of $t$, for a global threshold $T$, the tightest local threshold $L(T)$ is then $T \cdot r^{(\frac{1}{p}-1)}$. ■

While this method may provide the tightest local threshold for certain combinations of local and global similarity functions, it has two weaknesses. First, a separate solution needs to be found for each different pair of similarity functions and it is not clear whether a solution can always be found. Second, it is required that the local similarity function be known.

## 7. RESULT MERGING

To provide local system transparency to the global users, the results returned from component search engines should be combined into a single result. Ideally, documents in the merged result should be ranked in descending order of global similarities. However, such an ideal merge is very hard to achieve due to the various heterogeneities among the component systems. Usually, documents returned from each component search engine are ranked based on these documents' local ranking scores or similarities. Some component search engines make the local similarities of returned documents available to the user 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. Local similarities returned from different component search engines, even when made available, may be incomparable due to the heterogeneities among these search engines. Furthermore, the local similarities and the global similarities of the same document may be quite different.

The challenge here is to merge the documents returned from different 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. A further complication to the problem is that some documents may be returned from multiple component search engines. The question is whether and how this should affect the ranking of these documents.

Existing result merging approaches can be classified into the following two types.

**Local Similarity Adjustment:** This type of approaches adjusts local similarities using additional information such as the quality of component databases. A variation is to convert local document ranks to similarities.

**Global Similarity Estimation:** This type of approaches attempts to compute or estimate the true global similarities of the returned documents.

The first type is usually easier to implement but the merged ranking may be inaccurate as the merge is not based on the true global similarities of returned documents. The second type is more rigorous and has the potential to achieve the ideal merging. However, it typically needs more information from local systems. The two types of approaches are discussed in the following subsections.

### 7.1 Local Similarity Adjustment

Three cases can be identified depending on the degree of overlap among the selected databases for a given query.

**Case 1:** These databases are pair-wise disjoint or nearly disjoint. This occurs when disjoint special-purpose search engines or those with minimal overlap are selected.

**Case 2:** The selected databases overlap but are not identical. An example of this situation is when several general-purpose search engines are selected.

**Case 3:** These databases are identical.

Case 3 usually does not occur in a metasearch engine environment. Instead, it occurs when multiple ranking techniques are applied to the same collection of documents in order to improve the retrieval effectiveness. The result merging problem in this case is also known as *data fusion* [Vogt and Cottrell 1999]. Data fusion has been studied extensively in the last decade. One special property of the data fusion problem is that every document will be ranked or scored by each employed ranking technique. A number of functions have been proposed to combine individual ranking scores of the same document, including *min*, *max*, *average*, *sum*, *weighted average* and other linear combination functions [Cottrell and Belew 1994; Fox and Shaw 1994; Lee 1997; Vogt and Cottrell 1999]. One of the most effective functions for data fusion is known as *CombMNZ* which, for each document, sums individual scores and then multiplies the sum by the number of non-zero scores [Lee 1997]. This function emphasizes those documents that are ranked high by multiple systems. More data fusion techniques are surveyed in [Croft 2000].

We now consider more likely scenarios in a metasearch engine context, namely the selected databases are not identical. We first consider the case where the selected databases are disjoint. In this case, all returned documents will be unique. Let us first assume that all returned documents have local similarities attached. It is possible that different search engines normalize their local similarities in different ranges. For example, one search engine may normalize its similarities between 0 and 1 and another search engine 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]. In the following, we assume that all local similarities have been normalized based on a common range.

When database selection is performed for a given query, the usefulness or quality of each database is estimated and is represented as a score. The database scores can be used to adjust the local similarities. The idea is to give preference to documents from highly ranked databases. In CORI Net [Callan et al. 1995], the adjustment works as follows. Let $s$ be the ranking score of component database $D$ and $\bar{s}$ be the average of the scores of all databases searched. Then the following weight is assigned to $D$: $w = 1 + N \cdot \frac{s-\bar{s}}{\bar{s}}$, where $N$ is the number of component databases searched for the given query. Clearly, if $s > \bar{s}$, then $w$ will be greater than one. Furthermore, the larger the difference is, the larger the weight will be. On the other hand, if $s < \bar{s}$, then $w$ will be smaller than one. Moreover, the larger the difference is, the smaller the weight will be. Let $x$ be the local similarity of document $d$ from $D$. Then the adjusted similarity of $d$ is computed by $w \cdot x$. The result merger lists returned documents in descending order of adjusted similarities. Based on the way the weight of a database is computed, it is clear that documents from higher ranked databases have a better chance to be ranked higher in the merged result.

A similar method is used in ProFusion [Gauch et al. 1996]. For a given query, a ranking score is calculated for each database (see the discussion on ProFusion in Section 5.3.3). The adjusted similarity of a document $d$ from a database $D$ is the product of the local similarity of $d$ and the ranking score of $D$.

Now let us consider the situation where the local similarities of the returned documents from some component search engines are not available. In this case, one of the following two approaches could be applied to tackle the merging problem. Again, we assume that no document is returned from multiple search engines, i.e., all returned documents are unique.

(1) *Use the local document rank information directly to perform the merge.* Local similarities, if available, will be ignored in this approach. First, the searched databases are arranged in descending order of usefulness or quality scores obtained during the database selection step. Next, a round-robin method based on the database order and the local document rank order is used to merge the local document lists. Specifically, the first document in the merged list is the top ranked document from the highest ranked database and the second document in the merged list is the top ranked document from the second highest ranked database. After the top ranked documents from all searched databases have been selected, the next document in the merged list will be the second highest ranked document in the highest ranked database and the process con-

tinues until the desired number of documents are included in the merged list. One weakness of this solution is that it does not take into consideration the differences between the database scores (i.e. only the order information is utilized).

A randomized version of the above method is proposed in [Voorhees et al. 1995b]. Recall that in the MRDD database selection method, we first determine how many documents to retrieve from each component database for a given query to maximize the precision of the retrieval. Suppose the desired number of documents have been retrieved from each selected component database and $N$ local document lists have been obtained, where $N$ is the number of selected component databases. Let $L_i$ be the local document list for database $D_i$. To select the next document to be placed in the merged list, the rolling of a dice is simulated. The dice has $N$ faces corresponding to the $N$ local lists. Suppose $n$ is the total number of documents yet to be selected and $n_i$ documents are still in the list $L_i$. The dice is made biased such that the probability that the face corresponding to $L_i$ will be up when the dice is rolled is $n_i/n$. When the face for $L_i$ is up, the current top ranked document in the list $L_i$ will be selected as the next highest ranked document in the merged list. After the selection, the selected document is removed from $L_i$, and both $n_i$ and $n$ are reduced by 1. The probabilities are also updated accordingly. In this way, the retrieved documents are ranked based on the probabilistic model.

(2) *Convert local document ranks to similarities.* In D-WISE [Yuwono and Lee 1997], the following method is employed. For a given query, suppose $r_i$ is the ranking score of database $D_i$, $r_{min}$ is the lowest database ranking score (i.e. $r_{min} = \min\{r_i\}$), $r$ is the local rank of a document from database $D_i$ and $g$ is the converted similarity of the document. The conversion function is $g = 1 - (r - 1) \cdot F_i$, where $F_i$ is defined to be $(r_{min})/(m \cdot r_i)$ and $m$ is the number of documents desired across all searched databases. Intuitively, this conversion function has the following properties. First, all top-ranked documents from local systems will have the same converted similarity 1. This implies that all top-ranked documents from local systems are considered to be equally potentially useful. Second, $F_i$ is used to model the distance between the converted similarities of two consecutively ranked documents in database $D_i$. In other words, the difference between the converted similarities of the $j$-th and the $(j + 1)$th ranked documents from database $D_i$ is $F_i$. The distance is larger for databases with smaller ranking scores. As a result, if the rank of a document $d$ in a higher rank database is the same as the rank of document $d'$ in a lower rank database but none of $d$ and $d'$ is top-ranked, then the converted similarity of $d$ will be higher than that of $d'$. In addition, this method tends to select more documents from databases with higher scores into the merged result.

As an example, consider two databases $D_1$ and $D_2$. Suppose $r_1 = 0.2$ and $r_2 = 0.5$. Furthermore, suppose 4 documents are desired. Then, we have $r_{min} = 0.2$, $F_1 = 0.25$ and $F_2 = 0.1$. Based on the above conversion function, the top three ranked documents from $D_1$ will have converted similarities 1, 0.75, and 0.5, respectively, and the top three ranked documents from $D_2$ will have converted similarities 1, 0.9, and 0.8, respectively. As a result, the merged list will contain three documents from $D_2$ and one document from $D_1$. The documents will be ranked in descending order of converted similarities in the merged list.

Now let us consider the situation where the selected databases have overlap. For documents that are returned by a single search engine, the above discussed similarity adjustment techniques can be applied. We now consider how to deal with documents that are returned by multiple search engines. First, each local similarity can be adjusted using the techniques discussed above. Next, adjusted similarities for the same document can be combined in a certain way to produce an overall adjusted similarity for the document. The combination can be carried out by utilizing one of the combination functions proposed for data fusion. Indeed, this has been practiced by some 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]. It should be pointed out that an effective combination function in data fusion may not necessarily be effective in a metasearch engine environment. In data fusion, if a document is not retrieved by a retrieval technique, then it is because the document is not considered useful by the technique. In contrast, in a metasearch engine, there are two possible reasons for a document not to be retrieved by a selected search engine. The first is the same as in the data fusion case, namely the document is not considered sufficiently useful by the search engine. The second is that the document is not indexed by the search engine. In this case, the document did not have a chance to be judged for its usefulness by the search engine. Clearly, a document that is not retrieved due to the second reason will be put at a disadvantage if a combination function such as *sum* and $CombMNZ$ is used. Finding an effective combination function in a metasearch engine environment is an area that still needs further research.

### 7.2 Global Similarity Estimation

Under certain conditions, it is possible to compute or estimate the global similarities of returned documents. The following methods have been reported.

7.2.1 *Document Fetching.* That a document is returned by a search engine typically means that the URL of the document is returned. Sometimes, additional information associated with the document, such as a short summary or the first couple of sentences, is also returned. But the document itself is typically not returned.

The document fetching method downloads returned documents from their local servers and computes or estimates their global similarities in the metasearch engine. Consider the case in which the global similarity function is the *Cosine* function and the global document frequency of each term is known to the metasearch engine (note that if local databases have little or no overlap, then the global document frequency of a term can be computed or approximated as the sum of the local document frequencies of the term). After a document is downloaded, the term frequency of each term in the document can be obtained. As a result, all statistics needed to compute the global similarity of the document will be available and the global similarity can be computed. The Inquirus metasearch engine ranks documents returned from different search engines based on analyzing the contents of downloaded documents and a ranking formula that combines similarity and proximity matches is employed [Lawrence and Lee Giles 1998].

A document fetching based method that combines document selection and result merging is reported in [Yu et al. 1999b]. Suppose that the $m$ most similar documents across all databases with respect to a given query are desired for some positive integer $m$. In Section 5.2.5, we introduced a method to rank databases in descending order of the similarity of the most similar document in each database for a given query. Such a rank is an optimal rank for retrieving the $m$ most similar documents. This rank can also be used to perform document selection as follows.

First, for some small positive integer $s$ (e.g. $s$ can start from 2), each of the $s$ top ranked databases are searched to obtain the actual global similarity of its most similar document. This may require downloading some documents from these databases. Let $min\_sim$ be the minimum of these $s$ similarities. Next, from these $s$ databases, retrieve all documents whose actual global similarities are greater than or equal to the tentative threshold $min\_sim$. The tightest local threshold for each of these $s$ databases could be determined and used here. If $m$ or more documents have been retrieved, then this process stops. Otherwise, the next top ranked database (i.e. the $(s+1)$-th ranked database) will be considered and its most similar document will be retrieved. The actual global similarity of this document is then compared with $min\_sim$ and the minimum of these two similarities will be used as a new global threshold to retrieve all documents from these $s+1$ databases whose actual global similarities are greater than or equal to this threshold. This process is repeated until $m$ or more documents are retrieved. Retrieved documents are ranked in descending order of their actual global similarities. A potential problem with this approach is that the same database may be searched multiple times. This problem can be relieved to some extent by retrieving and caching a larger number of documents when searching a database.

This method has the following two properties [Yu et al. 1999b]. First, if the databases are ranked optimally, then all the $m$ most similar documents can be retrieved while accessing at most one unnecessary database, for any $m$. Second, for any single-term query, the optimal rank of databases can be achieved and, as a result, the $m$ most similar documents will be retrieved.

Downloading documents and analyzing them on the fly can be an expensive undertaking, especially when the number of documents to be downloaded is large and the documents have large sizes. A number of remedies have been proposed. First, downloading from different local systems can be carried out in parallel. Second, some documents can be analyzed first and displayed to the user so that further analysis can be done while the user reads the initial results [Lawrence and Lee Giles 1998]. The initially displayed results may not be correctly ranked and the overall rank needs to be adjusted when more documents are analyzed. Third, we may consider downloading only the beginning portion of each (large) document to analyze [Craswell et al. 1999].

On the other hand, downloading based approaches also have some clear advantages [Lawrence and Lee Giles 1998]. First, when trying to download documents, obsolete URLs can be identified. As a result, documents with dead URLs can be removed from the final result list. Second, by analyzing downloaded documents, documents will be ranked by their current contents. In contrast, local similarities may be computed based on old versions of these documents. Third, query terms in downloaded documents could be highlighted when displayed to the user.

7.2.2 *Use of Discovered Knowledge.* As discussed previously, one difficulty with result merging is that local document similarities may be incomparable because in different component search engines the documents may be indexed differently and the similarities may be computed using different methods (term weighting schemes, similarity functions, etc.). If the specific document indexing and similarity computation methods used in different component search engines can be discovered, for example, using the techniques proposed in [Liu et al. 2000], then we can be in a better position to figure out (1) what local similarities are reasonably comparable; (2) how to adjust some local similarities so that they will become more comparable with others; and (3) how to derive global similarities from local similarities. This is illustrated by the following example [Meng et al. 1999b].

EXAMPLE 3. Suppose it is discovered that all the component search engines selected to answer a given user query employ the same methods to index local documents and to compute local similarities, and no collection-dependent statistics such as the *idf* information are used, then the similarities from these local search engines can be considered as comparable. As a result, these similarities can be used directly to merge the returned documents.

If the only difference among these component search engines is that some remove stopwords and some do not (or the stopword lists are different), then a query may be adjusted to generate more comparable local similarities. For instance, suppose a term $t$ in query $q$ is a stopword in component search engine $E_1$ but not a stopword in component search engine $E_2$. In order to generate more comparable similarities, we can remove $t$ from $q$ and submit the modified query to $E_2$ (it does not matter whether the original $q$ or the modified $q$ is submitted to $E_1$).

If the *idf* information is also used, then we need to either adjust the local similarities or compute the global similarities directly to overcome the problem that the global *idf* and the local *idf*s of a term may be different. Consider the following two cases. It is assumed that both the local similarity function and the global similarity function are the $Cosine$ function.

**Case 1: Query $q$ consists of a single term $t$.** The similarity of $q$ with a document $d$ in a component database can be computed by

$$sim(d,q) = \frac{qtf_t(q) \times lidf_t \times dtf_t(d)}{|q| \cdot |d|}$$

where $qtf_t(q)$ and $dtf_t(d)$ are the $tf$ weights of term $t$ in $q$ and in $d$, respectively, and $lidf_t$ is the local *idf* weight of $t$. If the local *idf* formula has been discovered and the global document frequency of $t$ is known, then this local similarity can be adjusted to the global similarity by multiplying it by $\frac{gidf_t}{lidf_t}$, where $gidf_t$ is the global *idf* weight of $t$.

**Case 2: Query $q$ has multiple terms $t_1, ..., t_k$.** The global similarity between $d$ and $q$ in this case is

$$s = \frac{\sum_{i=1}^{k} qtf_{t_i}(q) \times gidf_{t_i} \times dtf_{t_i}(d)}{|q| \cdot |d|} = \sum_{i=1}^{k} \frac{qtf_{t_i}(q)}{|q|} \cdot \frac{dtf_{t_i}(d)}{|d|} \cdot gidf_{t_i}$$

Clearly, $\frac{qtf_{t_i}(q)}{|q|}$ and $gidf_{t_i}$, $i = 1, ..., k$, can all be computed by the metasearch engine as the formulas for computing them are known. Therefore, in order to find $s$, we need to find $\frac{dtf_{t_i}(d)}{|d|}$, $i = 1, ..., k$. To find $\frac{dtf_{t_i}(d)}{|d|}$ for a given term $t_i$ without downloading document $d$, we can submit $t_i$ as a single-term query. Let $s_i = sim(d, t_i) = \frac{qtf_{t_i}(t_i) \times lidf_{t_i} \times dtf_{t_i}(d)}{|t_i| \cdot |d|}$ be the local similarity returned. Then

$$\frac{dtf_{t_i}(d)}{|d|} = \frac{s_i \times |t_i|}{qtf_{t_i}(t_i) \times lidf_{t_i}} \tag{16}$$

Note that the expression on the right-hand side of the above formula can be computed by the metasearch engine when all the local formulas are known (i.e. have been discovered). In summary, $k$ additional single-term queries can be used to compute the global similarities between $q$ and all documents retrieved by $q$.

## 8. NEW CHALLENGES

As discussed in previous sections, much progress has been made to find efficient and accurate solutions to the problem of processing queries in a metasearch engine environment. However, as an emerging area, many outstanding problems remain to be solved. In this section, we list a few worthwhile challenges in this area.

(1) Integrate local systems employing different indexing techniques. Using different indexing techniques in different local systems can have serious impact on the compatibility of local similarities. Careful observation can reveal that using different indexing techniques can in fact affect the estimation accuracy in each of the three software components (i.e. database selection, document selection and result merging). New studies need to be carried out to investigate more precisely what impact it poses and how to overcome or alleviate the impact. Previous studies have largely been focused on different local similarity functions and local term weighting schemes.

(2) Integrate local systems supporting different types of queries (e.g. Boolean queries versus vector space queries). Most of our discussions in this article are based on queries in the vector space model [Salton and McGill 1983]. There exist metasearch engines that use Boolean queries [French et al. 1995; Li and Danzig 1997; NCSTRL ] and a number of works on dealing with Boolean queries in a metasearch engine have been reported [Gravano et al. 1994; Li and Danzig 1997; Sheldon et al. 1994]. Since very different methods may be used to rank documents for Boolean queries (traditional Boolean retrieval systems do not even rank retrieved documents) and vector space queries, we are likely to face

many new problems when integrating local systems that support both Boolean queries and vector space queries.

(3) Discover knowledge about component search engines. Many local systems are not willing to provide sufficient design and statistical information about their systems. They consider such information proprietary. However, without sufficient information about a local system, the estimation about the usefulness of the local system with respect to a given query may not be made accurately. One possible solution to this dilemma is to develop tools that can learn about a local system regarding the indexing terms used and certain statistical information about these terms as well as the similarity function used through probe queries. These learning or knowledge discovering tools can be used to facilitate not only the addition of new component search engines to an existing metasearch engine but also the detection of major upgrades or changes of existing component systems. Some preliminary work in this area has started to be reported. Using sampling technique to generate approximate database representatives for CORI Net is reported in [Callan et al. 1999]. In [Liu et al. 2000], a technique is proposed to discover how term weights are assigned in component search engines. New techniques need to be developed to discover knowledge about component search engines more accurately and more efficiently.

(4) Develop more effective result merging methods. As up to now, most result merging methods that have under gone extensive experimental evaluation are those proposed for data fusion. These methods may be unsuitable in the metasearch engine environment where databases of different component search engines are not identical. New methods that take into consideration the special characteristics of the metasearch engine environment need to be designed and evaluated. One such special characteristic is that when a document is not retrieved by a search engine, it may be because the document is not indexed by the search engine.

(5) There are two extreme ways to build a metasearch engine. One is to impose an interface on top of autonomous component search engines. In this case, no cooperation from these local systems can be expected. The other is to invite local systems to join a metasearch engine. In this case, the developer of the metasearch engine may set conditions, such as what similarity function(s) must be used and what information about the component databases must be provided, that must be satisfied for a local system to join the metasearch engine. Many possibilities exist between the two extremes. This means that it is likely, in a practical environment, that different types of database representatives will be available to the metasearch engine. How to use different types of database representatives to estimate comparable database usefulnesses is still a largely untouched problem.

An interesting issue is to come up with guidelines on what information from local systems are useful to facilitate the construction of a metasearch engine. Search engine developers may use such guidelines to design or upgrade their search engines. Multiple levels of compliance should be allowed with different compliance levels guaranteeing different levels of estimation accuracy. A serious initial effort in this regard can be found in [Gravano et al. 1997].

(6) Some new indexing and term weighting techniques have been developed for search engines for HTML documents. For example, some search engines (e.g. WWWW [McBryan 1994], Google [Brin and Page 1998] and Webor [Cutler et al. 1997]) use anchor terms in a web page to index the web page that is hyperlinked by the URL associated with the anchor. The rationale is that when authors of web pages add a hyperlink to another web page $p$, they include in the anchor tag a description of $p$ in addition to its URL. These descriptions have the potential of being very important for the retrieval of $p$ because they include the perception of these authors about the contents of $p$. As another example, some search engines also compute the weight of a term according to its position in the web page and its font type. In SIBRIS [Wade et al. 1989], the weight of a term in a page is increased if the term appears in the title of the page. A similar method is also employed in AltaVista, HotBot, and Yahoo. Google [Brin and Page 1998] assigns higher weights to terms in larger or bold fonts. It is known that co-occurrences and proximities of terms have significant influence on the relevance of documents. An interesting problem is how to incorporate these new techniques into the entire retrieval process and into the database representatives so that better metasearch engines can be built.

(7) Improve the effectiveness of metasearch. Most existing techniques rank databases and documents based on the similarities between the query and the documents in each database. Similarities are computed based on the match of terms in the query and documents. Studies in information retrieval indicate that when queries have a large number of terms, the correlation between highly similar documents and relevant documents exists provided appropriate similarity functions and term weighting schemes, such as the *Cosine* function and the $tfw * idfw$ weight formula, are used. However, for queries that are short, typical in the Internet environment [Jansen et al. 1998; Kirsch 1998], the above correlation is weak. The reason is that for a long query, the terms in the query provide context to each other to help disambiguate the meanings of different terms. In a short query, the particular meaning of a term often cannot be identified correctly. In summary, a similar document to a short query may not be useful to the user who submitted the query because the matching terms may have different meanings. Clearly, the same problem also exists for search engines. Methods need to be developed to address this issue. The following are some promising ideas. First, incorporate the importance of a document as determined by linkages between documents (e.g. PageRank [Page et al. 1998] and authority [Kleinberg 1998]) with the similarity of the document with a query [Yu et al. 2001]. Second, associate databases with concepts [Fan and Gauch 1999; Ipeirotis et al. 2001; Meng et al. 2001]. When a query is received by the metasearch engine, it is first mapped to a number of appropriate concepts and then those databases associated with the mapped concepts are used for database selection. The concepts associated with a database/query are used to provide some contexts for terms in the database/query. As a result, the meanings of terms can be more accurately determined. Third, user profiles may be utilized to support personalized metasearch. Fourth, collaborative filtering (CF) has been shown to be very effective for recommending useful documents [Konstan et al. 1997] and is employed by the DirectHit search engine

(www.directhit.com). The CF technique may also be useful for recommending databases to search for a given query.

(8) In Section 3, we identified the major software components for building a good metasearch engine. One issue that we have not discussed is where should these components be placed. An implicit assumption used in this article is that all components are placed at the site of the metasearch engine. However, valid alternatives exist. For example, instead of having the database selector at the global site, we could distribute it to all local sites. The representative of each local database can also be stored locally. In this scenario, each user query will be dispatched to all local sites for database selection. Each site then estimates the usefulness of its database with respect to the query to determine whether its local search engine should be invoked for the query. Although this placement of the database selector will incur a higher communication cost, it also has some appealing advantages. First, the estimation of database usefulness can now be carried out in parallel. Next, as database representatives are stored locally, the scalability issue becomes much less significant than when centralized database selection is employed. Other components such as the document selector may also have alternative placements. We need to investigate the pros and cons of different placements of these software components. New research issues may arise from these investigations.

(9) There is an urgent need to create a testbed to evaluate the proposed techniques for database selection, document selection and result merging. Although most papers that report these techniques include some experimental results, it is hard to draw general conclusions from these results due to the limitations of the documents and queries used. Some studies use various portions of some old TREC collections to conduct experiments [Callan et al. 1995; Voorhees et al. 1995b; Xu and Callan 1998] so that the information about the relevance of documents to each query can be utilized. However, the old TREC collections have several limitations. First, the number of queries that can be used for different portions of TREC collections is small (from 50 to 250). Second, these queries tend to be much longer on the average than typical queries encountered in the Internet environment [Abdulla et al. 1997; Jansen et al. 1998]. Third, the documents do not reflect more structured and more extensively hyperlinked web documents. In [Gravano and Garcia-Molina 1995; Meng et al. 1998; Meng et al. 1999a; Yu et al. 1999a], a collection of up to more than 6,000 real Internet queries is used. However, the database collection is small and there is no document relevance information. An ideal testbed should have a large collection of databases of various sizes, contents and structures, a large collection of queries of various lengths and with the relevant documents for each query identified. Recently, a testbed based on partitioning some old TREC collections into hundreds of databases is proposed for evaluating metasearch techniques [French et al. 1998; French et al. 1999]. However, this testbed is far from being ideal due to the problems inherited from the used TREC collections. Two new TREC collections consisting of Web documents (i.e. WT10g and VLC2; WT10g is a 10GB subset of the 100GB VLC2) have been created recently. The test queries are also typical Internet queries. It is possible that good testbeds can be derived

from them.

(10) Information sources on the Web often contain multimedia data such as text, image and video. Most work in metasearch deals with only text sources or the text aspect of multimedia sources. Database selection techniques have also been investigated for other media types. For example, selecting image databases in a metasearch context was studied in [Chang et al. 1998]. As another example, for data sources that can be described by attributes, such as book title and author name, a necessary and sufficient condition for ranking databases optimally was given in [Kirk et al. 1995]. The database selection method in [Liu 1999] also considered only data sources of mostly structured data. But there is a lack of research on providing metasearch capabilities for mixed media or multimedia sources.

The above list of challenges is by no means complete. New problems will arise with deeper understanding of the issues in metasearch.

## 9. CONCLUSIONS

With the increase of the number of search engines and digital libraries on the World Wide Web, providing easy, efficient and effective access to text information from multiple sources has increasingly become necessary. In this article, we presented an overview of existing metasearch techniques. Our overview concentrated on the problems of database selection, document selection and result merging. A wide variety of techniques for each of these problems was surveyed and analyzed. We also discussed the causes that make these problems very challenging. The causes include various heterogeneities among different component search engines due to the independent implementations of these search engines, and the lack of information about these implementations because they are mostly proprietary.

Our survey and investigation seem to indicate that better solutions to each of the three main problems, namely, database selection, document selection and result merging, require more information/knowledge about the component search engines such as more detailed database representatives, underlying similarity functions, term weighting schemes, indexing methods, and so on. There are currently no sufficiently efficient methods to find such information without the cooperation of the underlying search engines. A possible scenario is that we will need good solutions based on different degrees of knowledge about each local search engine and then apply these solutions accordingly.

Another important issue is the scalability of the solutions. Ultimately, we need to develop solutions that can scale in two orthogonal dimensions: data and access. Specifically, a good solution must scale to thousands of databases with many of them containing millions of documents and to millions of accesses a day. None of the proposed solutions has been evaluated under these conditions.

### ACKNOWLEDGMENTS

This work is supported in part by the following NSF grants: IIS-9902872, IIS-9902792 and EIA-9911099. We are very grateful to the anonymous reviewers and the editor, Michael Franklin, of the article for their invaluable suggestions and constructive comments. We also would like to thank Leslie Lander for reading

the manuscript and providing suggestions that have improved the quality of the manuscript.

## REFERENCES

Abdulla, G., Liu, B., Saad, R., and Fox, E. 1997. Characterizing world wide web queries. In *Technical Report TR-97-04, Virginia Tech.* (1997).

Baumgarten, C. 1997. A probabilistic model for distributed information retrieval. In *Proceedings of the ACM SIGIR Conference, Philadelphia, PA* (July 1997), pp. 258–266.

Bergman, M. 2000. *The Deep Web: Surfacing the hidden value.* BrightPlanet, www.completeplanet.com/Tutorials/DeepWeb/index.asp.

Boyan, J., Freitag, D., and Joachims, T. 1996. A machine learning architecture for optimizing web search engines. In *AAAI Workshop on Internet-based Information Systems, Portland, Oregon* (1996).

Brin, S. and Page, L. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh World Wide Web Conference, Brisbane, Australia* (April 1998), pp. 107–117.

Buckley, C., Salton, G., and Allan, J. 1993. Automatic retrieval with locality information using smart. In *Proceedings of the First Text REtrieval Conference, NIST Special Publication 500-207* (March 1993), pp. 59–72.

Callan, J. 2000. Distributed information retrieval. In *In Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval, edited by W. Bruce Croft. Kluwer Academic Publishers* (2000), pp. 127–150.

Callan, J., Connell, M., and Du, A. 1999. Automatic discovery of language models for text databases. In *Proceedings of the ACM SIGMOD Conference, Philadelphia, PA* (June 1999), pp. 479–490.

Callan, J., Croft, B., and Harding, S. 1992. The inquery retrieval system. In *Proceedings of the third DEXA Conference, Valencia, Spain* (1992), pp. 78–83.

Callan, J., Lu, Z., and Croft, W. 1995. Searching distributed collections with inference networks. In *Proceedings of the ACM SIGIR Conference, Seattle* (July 1995), pp. 21–28.

Chakrabarti, S., Dom, B., Kumar, S., Raghavan, P., Rajagopalan, S., Tomkins, A., Gibson, D., and Kleinberg, J. 1999. Mining the web's link structure. *IEEE Computer 32*, 8 (August), 60–67.

Chakravarthy, A. and Haase, K. 1995. Netserf: Using semantic knowledge to find internet information archives. In *Proceedings of the ACM SIGIR Conference, Seattle* (July 1995), pp. 4–11.

Chang, C. and Garcia-Molina, H. 1999. Mind your vocabulary: query mapping across heterogeneous information sources. In *Proceedings of the ACM SIGMOD Conference, Philadelphia, PA* (June 1999), pp. 335–346.

Chang, W., Murthy, D., Zhang, A., and Syeda-Mahmood, T. 1998. Global integration of visual databases. In *Proceedings of the IEEE International Conference on Data Engineering, Orlando, Florida* (February 1998), pp. 542–549.

Cottrell, G. and Belew, R. 1994. Automatic combination of multiple ranked retrieval systems. In *Proceedings of the ACM SIGIR Conference, Dublin, Ireland* (July 1994), pp. 173–181.

Craswell, N., Hawking, D., and Thistlewaite, P. 1999. Merging results from isolated search engines. In *Proceedings of the Tenth Australasian Database Conference, Auckland, New Zealand* (January 1999), pp. 189–200.

Croft, W. 2000. Combining approaches to information retrieval. In *In Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval, edited by W. Bruce Croft. Kluwer Academic Publishers* (2000), pp. 1–36.

Cutler, M., Shih, Y., and Meng, W. 1997. Using the structures of html documents to improve retrieval. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California* (December 1997), pp. 241–251.

DREILINGER, D. AND HOWE, A. 1997. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems 15*, 3 (July), 195–222.

FAN, Y. AND GAUCH, S. 1999. Adaptive agents for information gathering from multiple, distributed information sources. In *Proceedings of the 1999 AAAI Symposium on Intelligent Agents in Cyberspace, Stanford University* (March 1999), pp. 40–46.

FOX, E. AND SHAW, J. 1994. Combination of multiple searches. In *Proceedings of the Second Text REtrieval Conference, Gaithersburg, Maryland* (August 1994), pp. 243–252.

FRENCH, J., FOX, E., MALY, K., AND SELMAN, A. 1995. Wide area technical report service: technical report online. *Communications of the ACM 38*, 4 (April), 45–46.

FRENCH, J., POWELL, A., CALLAN, J., VILES, C., EMMITT, T., PREY, K., AND MOU, Y. 1999. Comparing the performance of database selection algorithms. In *Proceedings of the ACM SIGIR Conference, Berkeley, CA* (August 1999), pp. 238–245.

FRENCH, J., POWELL, A., AND VILES, C. 1998. Evaluating database selection techniques: a testbed and experiment. In *Proceedings of the ACM SIGIR Conference, Melbourne, Australia* (August 1998), pp. 121–129.

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. In *Proceedings of the ACM SIGMOD Conference, Tucson, Arizona* (May 1997), pp. 207–218.

GRAVANO, L. AND GARCIA-MOLINA, H. 1995. Generalizing gloss to vector-space databases and broker hierarchies. In *Proceedings of the International Conferences on Very Large Data Bases, Zurich, Switzerland* (September 1995), pp. 78–89.

GRAVANO, L. AND GARCIA-MOLINA, H. 1997. Merging ranks from heterogeneous internet sources. In *Proceedings of the International Conferences on Very Large Data Bases, Athens, Greece* (August 1997), pp. 196–205.

GRAVANO, L., GARCIA-MOLINA, H., AND TOMASIC, A. 1994. The effectiveness of gloss for the text database discovery problem. In *Proceedings of the ACM SIGMOD Conference, Minneapolis, Minnesota* (May 1994), pp. 126–137.

HAWKING, D. AND THISTLEWAITE, P. 1999. Methods for information server selection. *ACM Transactions on Information Systems 17*, 1 (January), 40–76.

IPEIROTIS, P., GRAVANO, L., AND SAHAMI, M. 2001. Probe, count, and classify: Categorizing hidden-web databases. In *Proceedings of the ACM SIGMOD Conference, Santa Barbara* (2001), pp. 67–78.

JANSEN, B., SPINK, A., BATEMAN, J., AND SARACEVIC, T. 1998. Real life information retrieval: A study of user queries on the web. *ACM SIGIR Forum 32*, 1, 5–17.

KAHLE, B. AND MEDLAR, A. 1991. An information system for corporate users: wide area information servers. In *Technical Report TMC199, Thinking Machine Corporation* (April 1991).

KIRK, T., LEVY, A., SAGIV, Y., AND SRIVASTAVA, D. 1995. The information manifold. In *AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments* (1995).

KIRSCH, S. 1998. Internet search: Infoseek's experiences searching the internet. *ACM SIGIR Forum 32*, 2, 3–7.

KLEINBERG, J. 1998. Authoritative sources in hyperlinked environment. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California* (January 1998), pp. 668–677.

KONSTAN, J., MILLER, B., MALTZ, D., HERLOCKER, J., GORDON, L., AND RIEDL, J. 1997. Grouplens: Applying collaborative filtering to usenet news. *Communication of the ACM 40*, 3, 77–87.

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. In *Proceedings of the Seventh International World Wide Web Conference, Brisbane, Australia* (April

1998), pp. 95–105.

Lawrence, S. and Lee Giles, C. 1999. Accessibility of information on the web. *Nature 400*, 107–109.

Lee, J.-H. 1997. Analyses of multiple evidence combination. In *Proceedings of the ACM SIGIR Conference, Philadelphia, PA* (July 1997), pp. 267–276.

Li, S. and Danzig, P. 1997. Boolean similarity measures for resource discovery. *IEEE Transactions on Knowledge and Data Engineering 9*, 6 (November), 863–876.

Liu, K., Meng, W., Yu, C., and Rishe, N. 2000. Discovery of similarity computations of search engines. In *Proceedings of the Nineth ACM International Conference on Information and Knowledge Management, Washington, D.C.* (November 2000), pp. 290–297.

Liu, K., Yu, C., Meng, W., Wu, W., and Rishe, N. 2001. A statistical method for estimating the usefulness of text databases. *IEEE Transactions on Knowledge and Data Engineering (to appear)*.

Liu, L. 1999. Query routing in large-scale digital library systems. In *Proceedings of the IEEE International Conference on Data Engineering, Sydney, Australia* (March 1999), pp. 154–163.

Manber, U. and Bigot, P. 1997. The search broker. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California* (December 1997), pp. 231–239.

Manber, U. and Bigot, P. 1998. Connecting diverse web search facilities. *Data Engineering Bulletin 21*, 2 (June), 21–27.

Mauldin, M. 1997. Lycos: design choices in an internet search service. *IEEE Expert 12*, 1 (February), 1–8.

McBryan, O. 1994. Genvl and wwww: Tools for training the web. In *Proceedings of the First World Wide Web Conference, Geneva, Switzerland* (May 1994), pp. 79–90.

Meng, M., Liu, K., Yu, C., Wang, X., Chang, Y., and Rishe, N. 1998. Determine text databases to search in the internet. In *Proceedings of the International Conferences on Very Large Data Bases, New York City* (August 1998), pp. 14–25.

Meng, M., Liu, K., Yu, C., Wu, W., and Rishe, N. 1999a. Estimating the usefulness of search engines. In *Proceedings of the IEEE International Conference on Data Engineering, Sydney, Australia* (March 1999), pp. 146–153.

Meng, W., Wang, W., Sun, H., and Yu, C. 2001. Concept hierarchy based text database categorization. *International Journal on Knowledge and Information Systems (to appear)*.

Meng, W., Yu, C., and Liu, K. 1999b. Detection of heterogeneities in a multiple text database environment. In *Proceedings of the Fourth IFCIS Conference on Cooperative Information Systems, Edinburgh, Scotland* (September 1999), pp. 22–33.

Miller, G. 1990. Wordnet: An on-line lexical database. *International Journal of Lexicography 3*, 4, 235–312.

NCSTRL. Networked computer science technical reference library. In *http://cs-tr.cs.cornell.edu*.

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

Robertson, S., Walker, S., and Beaulieu, M. 1999. Okapi at trec-7: Automatic ad hoc, filtering, vlc, and interactive track. In *Proceedings of the Seventh Text REtrieval Conference, Gaithersburg, Maryland* (November 1999), pp. 253–264.

Salton, G. 1989. *Automatic text processing: The transformation, analysis, and retrieval of information by Computer*. Addison Wesley.

Salton, G. and McGill, M. 1983. *Introduction to modern information retrieval*. McGraw-Hill, New York.

Selberg, E. and Etzioni, O. 1995. Multi-service search and comparison using the metacrawler. In *Proceedings of the Fourth World Wide Web Conference, Boston, Massachusetts* (December 1995), pp. 195–208.

Selberg, E. and Etzioni, O. 1997. The metacrawler architecture for resource aggregation on the web. *IEEE Expert 12*, 1, 8–14.

Sheldon, M., Duda, A., Weiss, R., O'Toole, J., and Gifford, D. 1994. A content routing system for distributed information servers. In *Proceedings of the Fourth International Conference on Extending Database Technology, Cambridge, England* (March 1994), pp. 109–122.

Singhal, A., Buckley, C., and Mitra, M. 1996. Pivoted document length normalization. In *Proceedings of the ACM SIGIR Conference, Zurich, Switzerland* (August 1996), pp. 21–29.

Sugiura, A. and Etzioni, O. 2000. Query routing for web search engines: architecture and experiments. In *Proceedings of the Ninth World Wide Web Conference, Amsterdam* (May 2000), pp. 417–429.

Towell, G., Voorhees, E., Gupta, N., and Johnson-Laird, B. 1995. Learning collection fusion strategies for information retrieval. In *Proceedings of the 12th International Conference on Machine Learning, Tahoe City, CA* (July 1995), pp. 540–548.

Turtle, H. and Croft, B. 1991. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems 9*, 3 (July), 8–14.

Vogt, C. and Cottrell, G. 1999. Fusion via a linear combination of scores. *Information Retrieval 1*, 3, 151–173.

Voorhees, E. 1996. Siemens trec-4 report: Further experiments with database merging. In *Proceedings of the Fourth Text REtrieval Conference, Gaithersburg, Maryland* (November 1996), pp. 121–130.

Voorhees, E., Gupta, N., and Johnson-Laird, B. 1995a. The collection fusion problem. In *Proceedings of the Third Text REtrieval Conference, Gaithersburg, Maryland* (November 1995), pp. 95–104.

Voorhees, E., Gupta, N., and Johnson-Laird, B. 1995b. Learning collection fusion strategies. In *Proceedings of the ACM SIGIR Conference, Seattle* (July 1995), pp. 172–179.

Voorhees, E. and Tong, R. 1997. Multiple search engines in database merging. In *Proceedings of the Second ACM International Conference on Digital Libraries, Philadelphia, PA* (July 1997), pp. 93–102.

Wade, S., Willett, P., and Bawden, D. 1989. Sibris: The sandwich interactive browing and ranking information system. *Journal of Information Science 15*, 249–260.

Widder, D. 1989. *Advanced Calculus. 2nd edition.* Dover Publications, Inc., New York.

Wu, Z., Meng, W., Yu, C., and Li, Z. 2001. Towards a highly-scalable and effective metasearch engine. In *Proceedings of the Tenth World Wide Web Conference, Hong Kong* (May 2001), pp. 386–395.

Xu, J. and Callan, J. 1998. Effective retrieval with distributed collections. In *Proceedings of the ACM SIGIR Conference, Melbourne, Australia* (1998), pp. 112–120.

Xu, J. and Croft, B. 1996. Query expansion using local and global document analysis. In *Proceedings of the ACM SIGIR Conference, Zurich, Switzerland* (August 1996), pp. 4–11.

Xu, J. and Croft, B. 1999. Cluster-based language models for distributed retrieval. In *Proceedings of the ACM SIGIR Conference, Berkeley, California* (August 1999), pp. 254–261.

Yu, C., Liu, K., Wu, M., W., W., and Rishe, N. 1999a. Finding the most similar documents across multiple text databases. In *Proceedings of the IEEE Conference on Advances in Digital Libraries, Baltimore, Maryland* (May 1999), pp. 150–162.

Yu, C. and Meng, W. 1998. *Principles of Database Query Processing for Advanced Applications.* Kaufmann Publishers, San Francisco, CA.

Yu, C., Meng, W., Liu, K., Wu, W., and Rishe, N. 1999b. Efficient and effective metasearch for a large number of text databases. In *Proceedings of the Eighth ACM International Conference on Information and Knowledge Management, Kansas City* (November 1999), pp. 217–224.

Yu, C., Meng, W., Wu, W., and Liu, K. 2001. Efficient and effective metasearch for text databases incorporating linkages among documents. In *Proceedings of the ACM SIGMOD Conference, Santa Barbara, CA* (May 2001), pp. 187–198.

YUWONO, B. AND LEE, D. 1996. Search and ranking algorithms for locating resources on the world wide web. In *Proceedings of the IEEE International Conference on Data Engineering, New Orleans, Louisiana* (February 1996), pp. 164–177.

YUWONO, B. AND LEE, D. 1997. Server ranking for distributed text resource systems on the internet. In *Proceedings of the 5th International Conference On Database Systems For Advanced Applications, Melbourne, Australia* (April 1997), pp. 391–400.