D

DATA SEARCH ENGINE

INTRODUCTION

The World Wide Web was first developed by Tim Berners-Lee and his colleagues in 1990. In just over a decade, it has become the largest information source in human history. The total number of documents and database records that are accessible via the Web is estimated to be in the hundreds of billions (1). By the end of 2005, there were already over 1 billion Internet users worldwide. Finding information on the Web has become an important part of our daily lives. Indeed, searching is the second most popular activity on the Web, behind e-mail, and about 550 million Web searches are performed every day.

The Web consists of the Surface Web and the Deep Web (Hidden Web or Invisible Web). Each page in the Surface Web has a logical address called Uniform Resource Locator (URL). The URL of a page allows the page to be fetched directly. In contrast, the Deep Web contains pages that cannot be directly fetched and database records stored in database systems. It is estimated that the size of the Deep Web is over 100 times larger than that of the Surface Web (1).

The tools that we use to find information on the Web are called *search engines*. Today, over 1 million search engines are believed to be operational on the Web (2). Search engines may be classified based on the type of data that are searched. Search engines that search text documents are called *document search engines*, whereas those that search structured data stored in database systems are called *database search engines*. Many popular search engines such as Google and Yahoo are document search engines, whereas many e-commerce search engines such as Amazon.com are considered to be database search engines. Document search engines usually have a simple interface with a textbox for users to enter a query, which typically contains some key words that reflect the user's information needs. Database search engines, on the other hand, usually have more complex interfaces to allow users to enter more specific and complex queries.

Most search engines cover only a small portion of the Web. To increase the coverage of the Web by a single search system, multiple search engines can be combined. A search system that uses other search engines to perform the search and combines their search results is called a *metasearch engine*. Mamma.com and dogpile.com are metasearch engines that combine multiple document search engines whereas addall.com is a metasearch engine that combines multiple database search engines for books. From a user's perspective, there is little difference between using a search engine and using a metasearch engine.

This article provides an overview of some of the main methods that are used to create search engines and metasearch engines. In the next section, we describe the basic techniques for creating a document search engine. Then we sketch the idea of building a database search engine. Finally, we introduce the key components of metasearch engines, including both document metasearch engines and database metasearch engines, and the techniques for building them.

DOCUMENT SEARCH ENGINE

Architecture

Although the architectures of different Web search engines may vary, a typical document search engine generally consists of the following four main components as shown in Fig. 1: Web crawler, Indexer, Index database, and Query engine. A Web crawler, also known as a Web spider or a Web robot, traverses the Web to fetch Web pages by following the URLs of Web pages. The Indexer is responsible for parsing the text of each Web page into word tokens and then creating the Index database using all the fetched Web pages. When a user query is received, the Query engine searches the Index database to find the matching Web pages for the query.

Crawling the Web

A Web crawler is a computer program that fetches Web pages from remote Web servers. The URL of each Web page identifies the location of the page on the Web. Given its URL, a Web page can be downloaded from a Web server using the HTTP (HyperText Transfer Protocol). Starting from some initial URLs, a Web crawler repeatedly fetches Web pages based on their URLs and extracts new URLs from the downloaded pages so that more pages can be downloaded. This process ends when some termination conditions are satisfied. Some possible termination conditions include (1) no new URL remains and (2) a preset number of pages have been downloaded. As a Web crawler may interact with numerous autonomous Web servers, it is important to design scalable and efficient crawlers.

To crawl the Web quickly, multiple crawlers can be applied. These crawlers may operate in two different manners (i.e., centralized and distributed). Centralized crawlers are located at the same location running on different machines in parallel. Distributed crawlers are distributed at different locations of the Internet, and controlled by a central coordinator; each crawler just crawls the Web sites that are geographically close to the location of the crawler. The most significant benefit of distributed crawlers is the reduction in communication cost incurred by crawling activity. Centralized crawlers, however, are easier to implement and control than distributed crawlers.

As the Web grows and changes constantly, it is necessary to have the crawlers regularly re-crawl the Web and make the contents of the index database up to date. Frequent re-crawling of the Web will waste significant resources and make the network and Web servers over-

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

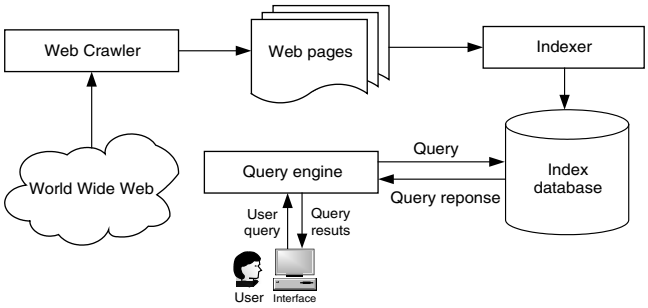

Figure 1. The general architecture of a document search engine.

loaded. Therefore, some incremental crawling strategies should be employed. One strategy is to re-crawl just the changed or newly added Web pages since the last crawling. The other strategy is to employ topic-specific crawlers to crawl the Web pages relevant to a pre-defined set of topics. Topic-specific crawling can also be used to build specialized search engines that are only interested in Web pages in some specific topics.

Conventional Web crawlers are capable of crawling only Web pages in the Surface Web. Deep Web crawlers are designed to crawl information in the Deep Web (3). As information in the Deep Web is often hidden behind the search interfaces of Deep Web data sources, Deep Web crawlers usually gather data by submitting queries to these search interfaces and collecting the returned results.

Indexing Web Pages

After Web pages are gathered to the site of a search engine, they are pre-processed into a format that is suitable for effective and efficient retrieval by search engines. The contents of a page may be represented by the words it has. Non-content words such as "the" and "is" are usually not used for page representation. Often, words are converted to their stems using a stemming program to facilitate the match of the different variations of the same word. For example, "comput" is the common stem of "compute" and "computing". After non-content word removal and stemming are performed on a page, the remaining words (called *terms* or *index terms*) are used to represent the page. Phrases may also be recognized as special terms. Furthermore, a weight is assigned to each term to reflect the importance of the term in representing the contents of the page.

The weight of a term t in a page p within a given set P of pages may be determined in a number of ways. If we treat each page as a plain text document, then the weight of t is usually computed based on two statistics. The first is its *term frequency* (*tf*) in p (i.e., the number of times t appears in p), and the second is its *document frequency* (*df*) in P (i.e., the number of pages in P that contain t). Intuitively, the more times a term appears in a page, the more important the term is in representing the contents of the page. Therefore, the weight of t in p should be a monotonically increasing function of its term frequency. On the other hand, the more pages that have a term, the less useful the term is in differentiating different pages. As a result, the weight of a term should be a monotonically decreasing function of its document frequency. Currently, most Web pages are formatted in HTML, which contains a set of tags such as *title* and *header*. The tag information can be used to influence the weights of the terms for representing Web pages. For example, terms in the title of a page or emphasized using bold and italic fonts are likely to be more important in representing a page than terms in the main body of the page with normal font.

To allow efficient search of Web pages for any given query, the representations of the fetched Web pages are organized into an *inverted file structure*. For each term t, an inverted list of the format $[(p_1, w_1), \ldots, (p_k, w_k)]$ is generated and stored, where each p_j is the identifier of a page containing t and w_j is the weight of t in p_j, $1 \leq j \leq k$. Only entries with positive weights are kept.

Ranking Pages for User Queries

A typical query submitted to a document search engine consists of some keywords. Such a query can also be represented as a set of terms with weights. The degree of match between a page and a query, often call the *similarity*, can be measured by the terms they share. A simple approach is to add up the products of the weights corresponding to the matching terms between the query and the page. This approach yields larger similarities for pages that share more important terms with a query. However, it tends to favor longer pages over shorter ones. This problem is often addressed by dividing the above similarity by the product of the lengths of the query and the page. The function that computes such type of similarities is called the *Cosine* function (4). The length of each page can be computed beforehand and stored at the search engine site.

Many methods exist for ranking Web pages for user queries, and different search engines likely employ different ranking techniques. For example, some ranking methods also consider the proximity of the query terms within a page. As another example, a search engine may keep track of the number of times each page has been accessed by users and use such information to help rank pages. Google (www.google.com) is one of the most popular search engines on the Web. A main reason why Google is successful is its powerful ranking method, which has the capability to differentiate more important pages from less important ones even when they all contain the query terms the same number of times. Google uses the linkage information among Web pages (i.e., how Web pages are linked) to derive the importance of each page. A link from page A to page B is placed by the author of page A. Intuitively, the existence of such a link is an indication that the author of page A considers page B to be of some value. On the Web, a page may be linked from many other pages and these links can be aggregated in some way to reflect the overall importance of the page. For a given page, *PageRank* is a measure of the relative importance of the page on the Web, and this measure is computed based on the linkage information (5). The following are the three main ideas behind the definition and computation of PageRank. (1) Pages that are linked from more pages are likely to be more important. In other words, the importance of a page should be reflected

by the popularity of the page among the authors of all Web pages. (2) Pages that are linked from more important pages are likely to be more important themselves. (3) Pages that have links to more pages have less influence over the importance of each of the linked pages. In other words, if a page has more child pages, then it can only propagate a smaller fraction of its importance to each child page. Based on the above insights, the founders of Google developed a method to calculate the importance (PageRank) of each page on the Web (5). The PageRanks of Web pages can be combined with other, say content-based, measures to indicate the overall relevance of a page with respect to a given query. For example, for a given query, a page may be ranked based on a weighted sum of its similarity with the query and its PageRank. Among pages with similar similarities, this method will rank those that have higher PageRanks.

Effective and Efficient Retrieval

For a given query, a page is said to be *relevant* if the sender of the query finds the page useful. For a given query submitted by a user against a fixed set of pages, the set of relevant pages is also fixed. A good retrieval system should return a high percentage of relevant pages to the user and rank them high in the search result for each query. Traditionally, the effectiveness of a text retrieval system is measured using two quantities known as *recall* and *precision*. For a given query and a set of documents, recall is the percentage of the relevant documents that are retrieved and precision is the percentage of the retrieved documents that are relevant. To evaluate the effectiveness of a text retrieval system, a set of test queries is often used. For each query, the set of relevant documents is identified in advance. For each test query, a precision value at a different recall point is obtained. When the precision values at different recall values are averaged over all test queries, an average recall-precision curve is obtained, which is used as the measure of the effectiveness of the system. A system is considered to be more effective than another system if the recall-precision curve of the former is above that of the latter. A perfect text retrieval system should have both recall and precision equal to 1 at the same time. In other words, such a system retrieves exactly the set of relevant documents for each query. In practice, perfect performance is not achievable for many reasons, for example, a user's information needs usually cannot be precisely specified by the used query and the contents of documents and queries cannot be completely represented by weighted terms.

Using both recall and precision to measure the effectiveness of traditional text retrieval systems requires knowing all the relevant documents for each test query in advance. This requirement, however, is not practical for independently evaluating large search engines because it is impossible to know the number of relevant pages in a search engine for a query unless all the pages are retrieved and manually examined. Without knowing the number of relevant pages for each test query, the recall measure cannot be computed. As a result of this practical constraint, search engines are often evaluated using the average precision based on the top k retrieved pages for a set of test queries, for some small integer k, say 20, or based on the average position of the first relevant page among the returned results for each test query (6).

A large search engine may index hundreds of millions or even billions of pages, and process millions of queries on a daily basis. For example, by the end of 2005, the Google search engine has indexed about 10 billion pages and processed over 200 million queries every day. To accommodate the high computation demand, a large search engine often employs a large number of computers and efficient query processing techniques. When a user query is received by a search engine, the inverted file structure of the pre-processed pages, not the pages themselves, are used to find matching pages. Computing the similarity between a query and every page directly is very inefficient because the vast majority of the pages likely do not share any term with the query and computing the similarities of these pages with the query is a waste of resources. To process a query, a *hash table* is first used to locate the storage location of the inverted file list of each query term. Based on the inverted file lists of all the terms in the query, the similarities of all the pages that contain at least one term in common with the query can be computed efficiently.

Result Organization

Most search engines display search results in descending order of their matching scores with respect to a given query. Some search engines, such as the Vivisimo search engine (www.vivisimo.com), organize their results into groups such that pages that have certain common features are placed into the same group. Clustering/categorizing search results is known to be effective in helping users identify relevant results in two situations. One is when the number of results returned for a query is large, which is mostly true for large search engines, and the other is when a query submitted by a user is short, which is also mostly true as the average number of terms in a search engine query is slightly over two. When the number of results is large, clustering allows the searcher to focus the attention on a small number of promising groups. When a query is short, the query may be interpreted in different ways, in this case, clustering can group results based on different interpretations that allow the searcher to focus on the group with desired interpretation. For example, when query "apple" is submitted to the Vivisimo search engine, results related to Apple computer (Macintosh) forms one group and results related to fruit forms another group, which makes it easy for a user to focus on the results he/she wants.

Challenges of Document Search Engines

Although Web search engines like Google, Yahoo, and MSN are widely used by numerous users to find the desired information on the Web, there are still a number of challenges for enhancing their quality (7,8). In the following, we briefly introduce some of these challenges.

Freshness. Currently, most search engines depend on Web crawlers to collect Web pages from numerous Web sites and build the index database based on the fetched Web pages. To refresh the index database so as to provide

up-to-date pages, they periodically (e.g., once every month) *recollect* Web pages from the Internet and *rebuild* the index database. As a result, pages that are added/deleted/changed since the last crawling are not reflected in the current index database, which makes some pages not accessible via the search engine, some retrieved pages not available on the Web (i.e., deadlinks), and the ranking of some pages based on obsolete contents. How to keep the index database up-to-date for large search engines is a challenging issue.

Coverage. It was estimated that no search engine indexes more than one-third of the "publicly indexable Web" (9). One important reason is that the Web crawlers can only crawl Web pages that are linked to the initial seed URLs. The "Bow Tie" theory about the Web structure (10) indicates that only 30% of the Web pages are strongly connected. This theory further proves the limitation of Web crawlers. How to fetch more Web pages, including those in the Deep Web, is a problem that needs further research.

Quality of Results. Quality of results refers to how well the returned pages match the given keywords query. Given a keywords query, a user wants the most relevant pages to be returned. Suppose a user submits "apple" as a query, a typical search engine will return all pages containing the word "apple" no matter if it is related to an apple pie recipe or Apple computer. Both the keywords-based similarity and the lack of context compromise the quality of returned pages. One promising technique for improving the quality of results is to perform a personalized search, in which a profile is maintained for each user that contains the user's personal information, such as specialty and interest, as well as some information obtained by tracking the user's Web surfing behaviors, such as which pages the user has clicked and how long the user spent on reading them; a user's query can be expanded based on his/her profile, and the pages are retrieved and ranked based on how well they match the expanded query.

Natural Language Query. Currently, most search engines accept only keywords queries. However, keywords cannot precisely express users' information needs. Natural language queries, such as "Who is the president of the United States?" often require clear answers that cannot be provided by most current search engines. Processing natural language queries requires not only the understanding of the semantics of a user query but also a different parsing and indexing mechanism of Web pages. Search engine ask.com can answer some simple natural language queries such as "Who is the president of the United States?" and "Where is Chicago?" using its *Web Answer* capability. However, ask.com does not yet have the capability to answer general natural language queries. There is still a long way to go before general natural language queries can be precisely answered.

Querying Non-Text Corpus. In addition to textual Web pages, a large amount of image, video, and audio data also exists on the Web. How to effectively and efficiently index and retrieve such data is also an open research problem in data search engines. Although some search engines such as Google and Yahoo can search images, their technologies are still mostly keywords-match based.

DATABASE SEARCH ENGINE

In comparison with document search engines, database search engines are much easier to build because they do not need crawlers to crawl the Web to build the index database. Instead, traditional database systems such as Oracle or SQL-server are usually used by database search engines to store and manage data. The stored data are often compiled and entered by human users. Unlike Web pages that have little structure, the data in database search engines are generally well structured. For example, the database of an online Web bookstore contains various books, and every book has attributes such as title, author, ISBN, publication date, and so on.

To make the data in a database search engine Web-accessible, an HTML form-based Web search interface like Fig. 2 is created on top of the underlying database system. The Web search interface often has multiple fields for users to specify queries that are more complex than the keywords queries for document Web search engines. For example, the search interface of bn.com (Fig. 2) contains fields like title, author, price, format, and so on. A user query submitted through the Web search interface of a database search engine is usually converted to a database query (e.g., SQL) that can be processed by the underlying database system; after the results that satisfy the query conditions are returned by the database system, they are wrapped by appropriate HTML tags and presented to the user on the dynamically generated Web page.

Database search engines are often used by organizations or companies that want to publish their compiled data on the Web for information sharing or business benefits. For example, a real estate company may employ a database search engine to post housing information, and an airline may use a database search engine to allow travelers to search and purchase airplane tickets.

It should be noted that structured data that are stored in database systems and are accessible via database search engines constitute a major portion of the Deep Web. A

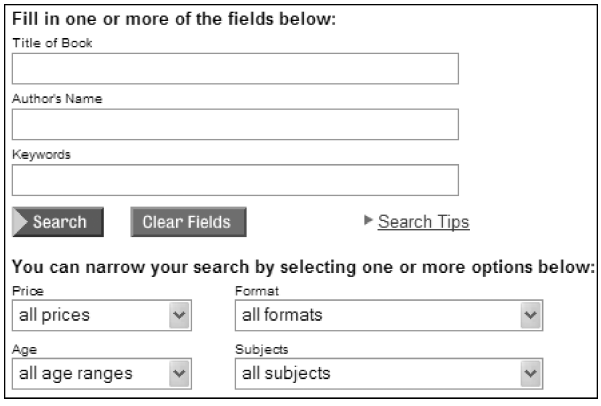

Figure 2. The book search interface of bn.com.engine.

recent survey (2) estimated that, by April 2004, among the 450,000 search engines for the Deep Web, 348,000 were database search engines.

METASEARCH ENGINE

A metasearch engine is a system that provides unified access to multiple existing search engines. When a metasearch engine receives a query from a user, it sends the query to multiple existing search engines, and it then combines the results returned by these search engines and displays the combined results to the user. A metasearch engine makes it easy for a user to search multiple search engines simultaneously while submitting just one query. A big benefit of a metasearch engine is its ability to combine the coverage of many search engines. As metasearch engines interact with the search interfaces of search engines, they can use Deep Web search engines just as easily as Surface Web search engines. Therefore, metasearch engine technology provides an effective mechanism to reach a large portion of the Deep Web by connecting to many Deep Web search engines.

Metasearch Engine Architecture

A simple metasearch engine consists of a user interface for users to submit queries, a search engine connection component for programmatically submitting queries to its employed search engines and receiving result pages from them, a result extraction component for extracting the search result records from the returned result pages, and a result merging component for combining the results (11). If a metasearch engine employs a large number of search engines, then a search engine selection component is needed. This component determines which search engines are likely to contain good matching results for any given user query so that only these search engines are used for this query. Search engine selection is necessary for efficiency considerations. For example, suppose only the 20 best-matched results are needed for a query and there are 1000 search engines in a metasearch engine. It is clear that the 20 best-matched results will come from at most 20 search engines, meaning that at least 980 search engines are not useful for this query. Sending a query to useless search engines will cause serious inefficiencies, such as heavy network traffic caused by transmitting unwanted results and the waste of system resources for evaluating the query.

We may have metasearch engines for document search engines and metasearch engines for database search engines. These two types of metasearch engines, though conceptually similar, need different techniques to build. They will be discussed in the next two subsections.

Document Metasearch Engine

A document metasearch engine employs document search engines as its underlying search engines. In this subsection, we discuss some aspects of building a document metasearch engine, including search engine selection, search engine connection, result extraction, and merging.

Search Engine Selection. When a metasearch engine receives a query from a user, the metasearch engine makes a determination on which search engines likely contain useful pages to the query and therefore should be used to process the query. Before search engine selection can be performed, some information representing the contents of the set of pages of each search engine is collected. The information about the pages in a search engine is called the *representative* of the search engine (11). The representatives of all search engines used by the metasearch engine are collected in advance and are stored with the metasearch engine. During search engine selection for a given query, search engines are ranked based on how well their representatives match with the query.

Different search engine selection techniques exist and they often employ different types of search engine representatives. A simple representative of a search engine may contain only a few selected key words or a short description. This type of representative is usually produced manually by someone who is familiar with the contents of the search engine. When a user query is received, the metasearch engine can compute the similarities between the query and the representatives, and then select the search engines with the highest similarities. Although this method is easy to implement, this type of representative provides only a general description about the contents of search engines. As a result, the accuracy of the selection may be low.

More elaborate representatives collect detailed statistical information about the pages in each search engine. These representatives typically collect one or several pieces of statistical information for each term in each search engine. As it is impractical to find out all the terms that appear in some pages in a search engine, an approximate vocabulary of terms for a search engine can be used. Such an approximate vocabulary can be obtained from pages retrieved from the search engine using sample queries (12). Some of the statistics that have been used in proposed search engine selection techniques include, for each term, its *document frequency*, its average or maximum weight in all pages having the term, and the number of search engines that have the term. With the detailed statistics, more accurate estimation of the usefulness of each search engine with respect to any user query can be obtained. The collected statistics may be used to compute the similarity between a query and each search engine, to estimate the number of pages in a search engine whose similarities with the query are above a threshold value, and to estimate the similarity of the most similar page in a search engine with respect to a query (11). These quantities allow search engines to be ranked for any given query and the top-ranked search engines can then be selected to process the query.

It is also possible to generate search engine representatives by learning from the search results of past queries. In this case, the representative of a search engine is simply the knowledge indicating its past performance with respect to different queries. In the SavvySearch metasearch engine (13) (now www.search.com), the learning is carried out as follows. For a search engine, a weight is maintained for each term that has appeared in previous queries. The

weight of a term for a search engine is increased or decreased depending on whether the search engine returns useful results for a query containing the term. Over time, if a search engine has a large positive (negative) weight for a term, the search engine is considered to have responded well (poorly) to the term in the past. When a new query is received by the metasearch engine, the weights of the query terms in the representatives of different search engines are aggregated to rank the search engines. The ProFusion metasearch engine also employs a learning-based approach to construct the search engine representatives (14). ProFusion uses training queries to find out how well each search engine responds to queries in 13 different subject categories. The knowledge learned about each search engine4 from training queries is used to select search engines to use for each user query and the knowledge is continuously updated based on the user's reaction to the search result (i.e., whether a particular page is clicked by the user).

Search Engine Connection. Usually, the search interface of a search engine is implemented using an HTML *form* tag with a query textbox. The form tag contains all information needed to connect to the search engine via a program. Such information includes the name and the location of the program (i.e., the search engine server) that processes user queries as well as the network connection method (i.e., the HTTP request method, usually GET or POST). The query textbox has an associated name and is used to fill out the query. The form tag of each search engine interface is pre-processed to extract the information needed for program connection. After a query is received by the metasearch engine and the decision is made to use a particular search engine, the query is assigned to the name of the query textbox of the search engine and sent to the server of the search engine using the HTTP request method supported by the search engine. After the query is processed by the search engine, a result page containing the search results is returned to the metasearch engine.

Search Result Extraction. A result page returned by a search engine is a dynamically generated HTML page. In addition to the search result records for a query, a result page usually also contains some unwanted information/links such as advertisements, search engine host information, or sponsored links. It is essential for the metasearch engine to correctly extract the search result records on each result page. A typical search result record corresponds to a Web page found by the search engine and it usually contains the URL and the title of the page as well as some additional information about the page (usually the first few sentences of the page plus the date at which the page was created, etc.; it is often called the *snippet* of the page).

As different search engines organize their result pages differently, a separate result extraction program (also called *extraction wrapper*) needs to be generated for each search engine. To extract the search result records of a search engine, the structure/format of its result pages needs to be analyzed to identify the region(s) that contain the records and separators that separate different records (15). As a result, a wrapper is constructed to extract the results of any query for the search engine. Extraction wrappers can be manually, semi-automatically, or automatically constructed.

Result Merging. Result merging is the task of combining the results returned from multiple search engines into a single ranked list. Ideally, pages in the merged result should be ranked in descending order of the global matching scores of the pages, which can be accomplished by fetching/downloading all returned pages from their local servers and computing their global matching scores in the metasearch engine. For example, the Inquirus metasearch engine employs such an approach (7). The main drawback of this approach is that the time it takes to fetch the pages might be long.

Most metasearch engines use the local ranks of the returned pages and their snippets to perform result merging to avoid fetching the actual pages (16). When snippets are used to perform the merging, a matching score of each snippet with the query can be computed based on several factors such as the number of unique query terms that appear in the snippet and the proximity of the query terms in the snippet. Recall that when search engine selection is performed for a given query, the usefulness of each search engine is estimated and is represented as a score. The search engine scores can be used to adjust the matching scores of retrieved search records, for example, by multiplying the matching score of each record by the score of the search engine that retrieved the record. Furthermore, if the same result is retrieved by multiple search engines, the multiplied scores of the result from these search engines are aggregated, or added up, to produce the final score for the result. This type of aggregation gives preference to those results that are retrieved by multiple search engines. The search results are then ranked in descending order of the final scores.

Database Metasearch Engine

A database metasearch engine provides a unified access to multiple database search engines. Usually, multiple database search engines in the same application domain (e.g., auto, book, real estate, flight) are integrated to create a database metasearch engine. Such a metasearch engine over multiple e-commerce sites allows users to do comparison-shopping across these sites. For example, a metasearch engine on top of all book search engines allows users to find desired books with the lowest price from all booksellers.

A database metasearch engine is similar to a document metasearch engine in architecture. Components such as search engine connection, result extraction, and result merging are common in both types of metasearch engines, but the corresponding components for database metasearch engines need to deal with more structured data. For example, result extraction needs to extract not only the returned search records (say books) but also lower level semantic data units within each record such as the titles and prices of books. One new component needed for a database metasearch engine is the search interface inte-

gration component. This component integrates the search interfaces of multiple database search engines in the same domain into a unified interface, which is then used by users to specify queries against the metasearch engine. This component is not needed for document metasearch engines because document search engines usually have very simple search interfaces (just a textbox). In the following subsections, we present some details about the search interface integration component and the result extraction component. For the latter, we focus on extracting lower level semantic data units within records.

Search Interface Integration. To integrate the search interfaces of database search engines, the first step is to extract the search fields on the search interfaces from the HTML Web pages of these interfaces. A typical search interface of a database search engine has multiple search fields. An example of such an interface is shown in Fig. 2. Each search field is implemented by text (i.e., field label) and one or more HTML form control elements such as textbox, selection list, radio button, and checkbox. The text indicates the semantic meaning of its corresponding search field. A search interface can be treated as a partial schema of the underlying database, and each search field can be considered as an attribute of the schema. Search interfaces can be manually extracted but recently there have been efforts to develop techniques to automate the extraction (17). The main challenge of automatic extraction of search interfaces is to group form control elements and field labels into logical attributes.

After all the search interfaces under consideration have been extracted, they are integrated into a unified search interface to serve as the interface of the database metasearch engine. Search interface integration consists of primarily two steps. In the first step, attributes that have similar semantics across different search interfaces are identified. In the second step, attributes with similar semantics are mapped to a single attribute on the unified interface. In general, it is not difficult for an experienced human user to identify matching attributes across different search interfaces when the number of search interfaces under consideration is small. For applications that need to integrate a large number of search interfaces or need to perform the integration for many domains, automatic integration tools are needed. WISE-Integrator (18) is a tool that is specifically designed to automate the integration of search interfaces. It can identify matching attributes across different interfaces and produce a unified interface automatically.

Result Extraction and Annotation. For a document search engine, a search result record corresponds to a retrieved Web page. For a database search engine, however, a search result record corresponds to a structured entity in the database. The problem of extracting search result records from the result pages of both types of search engines is similar (see search result extraction section). However, a search result record of a database entity is more structured than that of a Web page, and it usually consists of multiple lower level semantic data units that need to be extracted and annotated with appropriate labels to facilitate further data manipulation such as result merging.

Wrapper induction in (19) is a semi-automatic technique to extract the desired information from Web pages. It needs users to specify what information they want to extract, and then the wrapper induction system induces the rules to construct the wrapper for extracting the corresponding data. Much human work is involved in such wrapper construction. Recently, research efforts (20,21) have been put on how to automatically construct wrapper to extract structured data. To automatically annotate the extracted data instances, currently there are three basic approaches: ontology-based (22), search interface schema-based, and physical layout-based. In the ontology-based approach, a task-specific ontology (i.e., conceptual model instance) is usually predefined, which describes the data of interest, including relationships, lexical appearance, and context keywords. A database schema and recognizers for constants and keywords can be produced by parsing the ontology. Then the data units can be recognized and structured using the recognizers and the database schema. The search interface schema-based approach (23) is based on the observation that the complex Web search interfaces of database search engines usually partially reflect the schema of the data in the underlying databases. So the data units in the returned result record may be the values of a search field on search interfaces. The search field labels are thereby assigned to corresponding data units as meaningful labels. The physical layout-based approach assumes that data units usually occur together with their class labels; thus it annotates the data units in such a way that the closest label to the data units is treated as the class label. The headers of a visual table layout are also another clue for annotating the corresponding column of data units. As none of the three approaches is perfect, a combination of them will be a very promising approach to automatic annotation.

Challenges of Metasearch Engines

Metasearch engine technology faces very different challenges compared with search engine technology, and some of these challenges are briefly described below.

Automatic Maintenance. Search engines used by metasearch engines may change their connection parameters and result display format, and these changes can cause the search engines not usable from the metasearch engines unless the corresponding connection programs and result extraction wrappers are changed accordingly. It is very expensive to manually monitor the changes of search engines and make the corresponding changes in the metasearch engine. Techniques that can automatically detect search engine changes and make the necessary adjustments in the corresponding metasearch engine components are needed.

Scalability. Most of today's metasearch engines are very small in terms of the number of search engines used. A typical metasearch engine has only about a dozen search engines. The current largest metasearch engine AllInOneNews (www.allinonenews.com) connects to about 1500

news search engines. But it is still very far away from building a metasearch engine that uses all of the over half million document search engines currently on the Web. There are number of challenges in building very large-scale metasearch engines, including automatic maintenance described above, and automatic generation and maintenance of search engine representatives that are needed to enable efficient and effective search engine selection.

Entity Identification. For database metasearch engines, data records retrieved from different search engines that correspond to the same real-world entities must be correctly identified before any meaningful result merging can be performed. Automatic entity identification across autonomous information sources has been a very challenging issue for a long time. A related problem is the automatic and accurate data unit annotation problem as knowing the meanings of different data units in a record can greatly help match the records. Although research on both entity identification and data unit annotation is being actively pursued, there is still long way to go to solve these problems satisfactorily.

ACKNOWLEDGMENT

This work is supported in part by the following NSF grants: IIS-0414981, IIS-0414939, and CNS-0454298. We also would like to thank the anonymous reviewers for valuable suggestions to improve the manuscript.

BIBLIOGRAPHY

1. M. Bergman. The Deep Web: Surfacing the Hidden Value. BrightPlanet White Paper. Available: http://www.brightplanet.com/images/stories/pdf/deepwebwhitepaper.pdf, October 16, 2006.
2. K. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the Web: observations and Implications. *SIGMOD Record,* **33** (3): 61–70, 2004.
3. S. Raghavan, and H. Garcia-Molina. Crawling the hidden Web. VLDB Conference, 2001.
4. G. Salton, and M. McGill. *Introduction to modern information retrieval.* New York: McCraw-Hill, 2001.
5. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bring Order to the Web. Technical Report, Stanford University, 1998.
6. D. Hawking, N. Craswell, P. Bailey, and K. Griffiths. Measuring search engine quality. *J. Inf. Retriev.,* **4** (1): 33–59, 2001.
7. S. Lawrence, and C. Lee Giles. Inquirus, the NECi Meta Search Engine. Seventh International World Wide Web conference, Brisbane, Australia, 1998, pp. 95–105.
8. M. R. Henzinger, and R. Motwani, and C. Silverstein. Challenges in Web Search Engines. Available http://citeseer.ist.psu.edu/henzinger02challenges.html, 2002.
9. S. Lawrence, and C. Lee Giles. Accessibility of information on the Web. *Nature,* **400**: 1999.
10. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph Structure in the Web. The 9th International World Wide Web Conference, Amsterdam, 2001.
11. W. Meng, C. Yu, and K. Liu. Building Efficient and Effective Metasearch Engines. *ACM Comput. Surv.,* **34** (1), 48–84, 2002.
12. J. Callan, M. Connell, and A. Du, Automatic Discovery of Language Models for Text Databases. ACM SIGMOD Conference, Philadelphia, PA, 1999, pp. 479–490.
13. D. Dreilinger, and A. Howe. Experiences with selecting search engines using metasearch. *ACM Trans. Inf. Syst.,* **15** (3): 195–222.
14. Y. Fan, and S. Gauch. Adaptive Agents for Information Gathering from Multiple, Distributed Information Sources. AAAI Symposium on Intelligent Agents in Cyberspace Stanford University, 1999, pp. 40–46.
15. H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. World Wide Web Conference, 2005.
16. Y. Lu, W. Meng, L. Shu, C. Yu, and K. Liu. Evaluation of result merging strategies for metasearch engines. *WISE Conference* , 2005.
17. Z. Zhang, B. He, and K. Chang. Understanding Web query interfaces: best-effort parsing with hidden syntax. ACM SIGMOD Conference, 2004.
18. H. He, W. Meng, C. Yu, and Z. Wu. Automatic integration of web search interfaces with wise-integrator. *VLDB J.* **13** (3): 256–273, 2004.
19. C. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the Web: a machine learning approach. *IEEE Data Eng. Bull.* **23** (4):2000.
20. A. Arasu, and H. Garcia-Molina. Extracting Structured Data from Web pages. SIGMOD Conference, 2003.
21. V. Crescenzi, G. Mecca, and P. Merialdo. RoadRUNNER: Towards Automatic Data Extraction from Large Web Sites. VLDB Conference, Italy, 2001.
22. D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y. K. Ng, R. D. Smith, Conceptual-model-based data extraction from multiple-record Web pages. *Data Knowledge Eng.,* **31** *(3):* 227–251, 1999.
23. J. Wang, and F. H. Lochovsky. Data Extraction and Label Assignment for Web Databases. WWW Conference, 2003.

FURTHER READING

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

WEIYI MENG
HAI HE
State University of New York
Binghamton, New York