Detection of Heterogeneities in a Multiple Text Database Environment

Weiyi Meng

Dept. of Computer Science
SUNY – Binghamton
Binghamton, NY 13902
meng@cs.binghamton.edu

Clement Yu, King-Lup Liu

Dept. of EECS
University of Illinois at Chicago
Chicago, IL 60607
{yu, kliu}@eecs.uic.edu

Abstract

As the number of text retrieval systems (search engines) grows rapidly on the World Wide Web, there is an increasing need to build search brokers (metasearch engines) on top of them. Often, the task of building an effective and efficient metasearch engine is hindered by the heterogeneities among the underlying local search engines. In this paper, we first analyze the impact of various heterogeneities on building a metasearch engine. We then present some techniques that can be used to detect the most prominent heterogeneities among multiple search engines. Applications of utilizing the detected heterogeneities in building better metasearch engines will be provided.

Keywords: *Metasearch Engine, Heterogeneities, Distributed Document Collection, Knowledge Discovery.*

1 Introduction

The Internet has become a vast information resource in recent years. In particular, the World Wide Web (WWW or Web) has become increasingly popular for exchanging information among different organizations and individuals. Millions of people use the Web on a regular basis and the number is increasing rapidly. Finding desired data is one of the most popular ways the Internet is utilized. Many search engines have been created to facilitate the retrieval of web pages on the Web. Each search engine has a *text database* that is defined by the set of documents that can be searched by the search engine. 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. In this paper, we consider only search engines that support vector queries (i.e., queries that can be represented as a set of terms with no boolean operators).

Although general-purpose search engines that attempt to provide searching capabilities for all documents on the Web, like Excite, Lycos, HotBot, and Alta Vista, are quite popular, most search engines on the Web are special-purpose search engines that focus on documents in confined domains such as documents in an organization or of a specific subject area. 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 research papers in some subject area. It is likely that the desired papers are scattered in a number of publishers' and/or universities' databases. It is very inconvenient and inefficient for the user to determine useful databases, search them individually and identify useful documents all by him/herself. A solution to this problem is to implement a *metasearch engine* on top of many local search engines. A metasearch engine is just an interface. It does not maintain its own index on documents. However, a sophisticated metasearch engine may maintain information about the contents of its underlying search engines to provide better service. When a metasearch engine receives a user query, it first passes the query (with necessary reformatting) to the appropriate local search engines, and then collects (sometimes, reorganizes) the results from its local search engines. Clearly, with such a metasearch engine, the above user's task will be drastically simplified.

A substantial body of research work addressing different aspects of building an effective and efficient metasearch engine has been accumulated in recent years. Among the main challenges, the *database selec-*

tion problem is to identify, for a given user query, the local search engines that are likely to contain useful documents for the query [1, 6, 10, 13, 16, 18, 21, 24, 25, 29, 35, 37]. The objective of performing database selection is to improve efficiency as the metasearch engine can send each query to only potentially useful search engines, cutting down network traffic and the cost of searching useless databases. The *document selection problem* is to determine what documents should be retrieved from each search engine invoked [6, 15, 24, 33, 35, 37]. This is to avoid retrieving an excessive number of useless documents from local databases as retrieving these documents may have several negative effects (higher local cost, higher communication cost for shipping these documents and higher cost to merge them). The *result merging problem* is to combine the documents returned from multiple search engines into a single ranked list [6, 10, 29, 37]. A good metasearch engine should have the retrieval effectiveness close to that as if all documents were in a single database while minimizing the access cost.

Search engines in the Internet are usually designed and implemented independently. As a consequence, substantial heterogeneities exist among these search engines. For example, different similarity functions (ranking algorithms) may be used by different search engines. The existence of these heterogeneities is often the primary source of difficulties in developing effective and efficient metasearch engines. The detection of specific heterogeneities among a set of local search engines can facilitate the development of a metasearch engine on top of these local search engines. In this paper, we present techniques for detecting specific heterogeneities among multiple search engines. These techniques are based on using carefully designed probe queries to retrieve documents and analyzing the retrieval results for all search engines.

This paper has two main contributions. First, we identify major heterogeneities that may exist among local search engines and analyze the impact of these heterogeneities on the effective and efficient retrieval of documents in a metasearch engine environment. While heterogeneities among traditional database systems (relational, object-oriented, ...) and their impact on building multidatabase systems have been studied extensively (see for example, [12, 17, 31, 36]), there have been relatively few studies for text database systems. Second, we present techniques to detect specific heterogeneities among multiple text retrieval systems. Applying probe queries to discover knowledge about a search engine is a new research area. Most recently, the authors of [7] used probe queries to discover the terms in a local database and some statistical information about these terms.

The rest of the paper is organized as follows. In Section 2, we identify major heterogeneities among local search engines and analyze their impact on building an effective and efficient metasearch engine. The impact of autonomy of local systems will also be discussed. In Section 3, we present techniques to detect specific heterogeneities. In Section 4, we discuss applications of the knowledge discovered in Section 3 to help developing better metasearch engines. We conclude the paper in Section 5.

2 Heterogeneities and Their Impacts

In Section 2.1, we identify major heterogeneities that are unique in the metasearch engine environment. Heterogeneities that are common to other automonous systems (e.g., regular multidatabase systems) such as different OS platforms will not be described. In Section 2.2, we discuss the impact of these heterogeneities as well as the autonomy of local search engines on building an effective and efficient metasearch engine.

2.1 Heterogeneities

Local 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. In addition, 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 search engines, the following heterogeneities may exist among different local search engines.

Indexing Method: Different search engines may have different ways to determine what terms should be used to index or 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 [22], 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 [4, 9, 23] while most other search engines do not. Other examples of different indexing techniques involve whether or not to remove *stopwords* (i.e., non-content words such as "the", "and", etc) and whether or not to perform *stemming* (i.e., whether or not to transform words such as "mountainous" to its stem "mountain"). Furthermore, different stopword lists and stemming algorithms could be used by different search engines.

Document Term Weighting Scheme: The importance of a term in representing or identifying a document is represented as a numeric value (called the *weight*). Different methods exist for determining the weight. One popular scheme uses the number of times that a term appears in a document (this is known as the *term frequency* of the term in the document) as the weight. Intuitively, the more frequently that a term appears in a document, the more important the term is in representing the contents of the document. Therefore, the weight of a term in a document should be an increasing function of the *term frequency* of the term. Several variations of this scheme exist [27] (also see Section 3). Another popular scheme uses both the *term frequency* and the *document frequency* of a term to determine the weight of the term. The latter is the number of documents in the database that contain the term. Intuitively, if fewer documents have a term, then the term is more useful in differentiating these documents from other documents. Therefore, the weight of a term in a document should be a decreasing function of the *document frequency* of the term. There are a number of variations for incorporating the document frequency of a term into the computation of the weight of the term (see Section 3). There are also systems that distinguish different occurrences of the same term [3, 9, 34] or different fonts of the same term [4]. 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 (such a distinction is made by AltaVista, HotBot, Yahoo, SIBRIS [34], and Webor [9]).

Query Term Weighting Scheme: In the vector 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. Different local search engines may employ different query term weighting schemes.

Similarity Function: Different search engines may employ different similarity functions to measure the similarity between a user query and a document. For example, some search engines may use the dot product of the term weight vectors of a query and a document to compute the similarity between the query and the document while some other search engines may divide the dot product by the product of the lengths of the two vectors to normalize similarities between 0 and 1. The latter similarity function is known as the $Cosine$ function. Other similarity functions, see for example [32], are also possible.

Inverted File Implementation: Inverted file index is the standard data structure for supporting efficient evaluation of user queries against large text databases. Conceptually, such an index for a database contains an inverted list for each distinct term in the database. The list contains a list of pairs (d_i, w_i), where d_i is the id of a document containing the term and w_i is the weight of the term in the document. (Sometimes, locations of terms in documents are also stored to facilitate the evaluation of phrase queries and proximity queries. This aspect will not be addressed in this paper as our focus is on vector queries.) In practice, the inverted file index may be implemented in a variety of ways. For example, one possibility is to store the actual weights directly and another possibility is to store only raw statistical data such as term frequencies and document frequencies and then compute the weights when queries are being processed. The former implementation can evaluate user queries faster as much computation has been done in advance. The latter can support updates to the database (addition, removal and updates of documents) better and is also more flexible in terms of supporting changes of the term weighting scheme of a search engine. Therefore, the first implementation is more suitable for more static databases where no or little changes are expected while the second implementation is better for more dynamic databases.

Document Database: The text databases of different search engines may differ at two levels. The first level is the *domain* (subject area) of database. For example, one database may contain medical documents and another may contain legal documents. 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 as 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.

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, attempt 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: 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 while some do not.

2.2 The Impacts

We now analyze the impact of the above heterogeneities among different search engines as well as the local system autonomy on the development of effective and efficient metasearch engines. In particular, we discuss the impact on the implementation of database selection, document selection and result merging strategies.

2.2.1 Impacts on Database Selection

Database selection is to determine which databases should be searched with respect to a given query. The determination is usually made by estimating the *usefulness* of each search engine for the query, where the usefulness could be some ranking score [6, 13, 37] or the number of potentially useful documents (whose similarities with the queries are sufficiently high) in a search engine [14, 24, 25]. In order to estimate the usefulness of a database to a query, the metasearch engine often needs to know some information about a database that characterizes the contents of the documents in the database. We call the characteristic information about a database the *representative* of the database. Depending on the database selection methods used, the required database representative may contain detailed statistical information about the terms in a database such as the document frequency of each term [6, 13, 37], the sum or the average of the weights of each term [13, 24, 25, 35] and the maximum weight of each term [25, 35].

Database selection can be affected by both the autonomy of local search engines and the heterogeneities among them.

1. The need for database selection is largely due to the fact that there are heterogeneous document databases. If the databases of all local search engines have the same domain (subdomain) such that for each query useful documents are likely to be found from all databases, then the need to do database selection is diminished.

2. Due to its autonomy, a local search engine may be unwilling to provide the representative of its database. In this case, the metasearch engine may be forced to send every user query to this search engine (i.e., this search engine is always selected).

   There are two possible solutions to this problem. The first is to keep track of past retrieval experiences with the search engine and use the experiences to predict the usefulness of the search engine for future queries. SavvySearch is a metasearch engine that uses this solution [10]. The second solution is to submit probe queries to the search engine and extract a database representative from the retrieved documents [7].

3. Due to both autonomy and heterogeneity, different types of database representatives for different search engines may be available in the metasearch engine. First, we may have representatives extracted from past experiences or retrieved documents for search engines that do not want to provide their database representatives. Second, some search engines may be willing and able to provide database representatives preferred by the metasearch engine. Third, some search engines may not be able to provide representatives that are desired by the metasearch engine. For example, suppose a search engine stores pre-computed document term weights in its inverted file index and the metasearch engine wants in the representative the average of the weights of each term that are computed using a particular formula. If the formula desired by the metasearch engine and that in the local search engine are different, then the local search engine may not be able to provide the representative wanted by the metasearch engine. In general, since different search engines have different ways to represent their documents, to compute their term weights and to implement their inverted file indexes, the database representatives that can be provided by them could be very different. As a result of the diversity of database representatives, different database selection techniques need to be developed.

2.2.2 Impacts on Document Selection

Document selection is to determine what documents should be retrieved from each selected search engine. Ideally, only potentially useful documents with respect to a given query should be retrieved from a local search engine. Consider the scenario when a user submits a query to the metasearch engine, he/she indicates that n documents are desired for some positive integer n. In this case, the n documents returned to the user should be the n most useful documents to the query across all local search engines. In practice, we would like to find the n documents that are most similar to the query across all local search engines. In other words, a document can be said to be potentially useful if it is among the n most similar documents across all local search engines. Heterogeneities among different local search engines have at least the following impact on the document selection problem.

1. *How to determine potentially useful documents.* Let us continue the example of retrieving n most similar documents to a given query across all selected search engines. A question that comes into mind is how do we define the similarity. Since different similarity functions may be used in different local search engines, similarities computed by different local search engines are not directly comparable. (Other factors such as different indexing and term weighting methods, for both query and documents, may also make local similarities not or less comparable even if the same similarity function is used by all local search engines. See Section 2.2.3.) As a result, local similarities should not be used solely to determine which documents are among the n most similar documents to the query across all local search engines. A solution to this problem is to employ a global similarity function and use global similarities computed by this function to determine the n most similar documents to a query.

2. *How to find potentially useful documents.* Since the global similarity of a document and the local similarity of the document in a local search engine may be computed very differently, a potentially useful document may have a rather low local similarity. A question here is how do we ensure that all the n globally most similar documents are retrieved from local search engines and at the same time minimize the retrieval of useless documents. The retrieval of an excessive number of useless documents from local search engines will incur higher local processing cost for retrieving more documents, higher communication cost for returning more documents to the metasearch engine and higher global cost for finding the n globally most similar documents from more documents.

   A solution to this problem is as follows. First, a global threshold GT is estimated such that the total number of documents from all search engines whose global similarities are greater than GT is n. Next, for each local search engine, determine a local threshold LT such that all documents in the search engine whose global similarities are higher than GT have local similarities higher than LT. In other words, the set of documents with local similarities greater than LT in the local search engine contains all the documents in the search engine whose global similarities are higher than GT. Clearly, in order to minimize the number of useless documents to be retrieved from a local search engine, we need to find the largest LT for the local search engine. The problem of determining LTs from GT is studied in [15, 24].

Because different local search engines have different ways to compute local similarities, different methods may be needed to determine the *LT*s for different local search engines.

2.2.3 Impacts on Result Merging

To provide local system transparency to the global users, the results returned from local 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 heterogeneities among the local systems. Specifically, local document similarities from different local search engines may not be comparable due to differences in similarity function, in term weighting schemes (for both query and documents), in indexing method and in document version, and therefore cannot be used directly for ranking returned documents. Moreover, some local search engines may not provide local similarities for returned documents.

Consider first the scenario where different versions of the same document (in terms of a unique document id, e.g., the URL of a web page) are indexed by different local search engines and the same document (id) is returned from more than one local search engine. The problem is how to provide a sensible estimate of the global similarity of this document in this situation. A number of solutions are possible. If each local search engine keeps the time when the document is indexed by the system and the time can be made available to the metasearch engine, then the similarity of the document from the local system that indexed the document most recently may be used. If several search engines have indexed the most recent version of the document and they have rather different ways to compute document similarities, then the local similarities of the same document can be combined to generate a global similarity to reflect the fact that the same document is retrieved using different methods. Another possibility is to fetch the document and compute its global similarity directly.

Different term weighting schemes can also affect the comparability of local similarities. The similarity between a query and a document is computed using the weights of the terms appearing in the query and the weights of the terms appearing in the document. As a result, different term weights will yield different similarities. Clearly, if a local search engine uses the inverse document frequency (*idf*) information of a term to compute the document term weight while another local search engine does not use the information, then the same document (the same version) will likely be represented by different weight vectors in the two search engines. In fact, a closer look can reveal that sometimes even when the same term weighting scheme is used in two local search engines, the same document may still be represented differently. As an example, consider again the case where the *idf* information of a term is used to compute the weight of the term in each document. It has been observed [11, 19] that the use of local *idf*'s has the tendency to reward the rare use of a term in one local system and penalize the common use of the term in another local system. For example, consider two local systems, D1 and D2, such that D1 contains research papers in computer science and D2 contains research papers in medical science. The term "computer" is likely to be mentioned in almost all papers in D1 and only a few papers in D2. As a result, if local *idf*'s are used, then the weights of "computer" in the documents in D1 will be zero or close to zero while those in D2 will be much larger. Suppose a query containing a single term "computer" is issued and a document containing the term "computer" appears in both D1 and D2. Then the similarity of the query with the document from D1 will be lower than that with the document from D2 if all other conditions are the same. In general, it is highly likely that the *idf*s of a term in different local systems are different and all of them are different from the *idf* of the term across all databases. In summary, term weighting schemes can have a big impact on the comparability of local similarities.

We now consider the problem caused by different document indexing methods. Two indexing methods may differ in a variety of ways. For example, one local system may perform full text indexing and another system employs partial text indexing. Partial text indexing may affect the term frequency and document frequency of a term. As another example, if one local system employs *stemming* and another system does not (or they employ a different stemming algorithm), then again the term frequency and document frequency of a term may be affected. In each of the above examples, different similarities may be returned for the same document that appears in different local systems even when the same similarity function is employed by all local systems.

3 Detection of Heterogeneities

In this section, we investigate the problem of detecting heterogeneities among multiple local search engines. Our solution to this problem can be summarized as follows. First, we discover specific methods (indexing, term weighting, similarity function, ...) and situations (e.g., document database) that are used in or associated with each local search engine. Then we compare these specific methods and situations to determine what types of heterogeneities exist among these search engines. In Section 4, we will discuss how knowing the specific nature of various heterogeneities can help us develop appropriate solutions to many problems caused by these heterogeneities in a metasearch engine environment.

Among the heterogeneities discussed in Section 2, some can be identified easily (e.g., result presentation) and some may never be detected (e.g., inverted file implementation). A recent paper [7] used sampling queries to discover the list of terms that appear in the documents of a database and also some statistical information for each term. The discovered information can be used as the *representative* of the database. For a specialized database, this method may be used to find the domain of the database, for example, by examining the most frequent content-words discovered.

In this section, we focus on the discovering of specific implementational methods (document indexing methods, term weighting schemes, similarity function, ...) that are used in local search engines. The technique that we employ for the discovery is also the query sampling technique. The basic idea is to submit carefully chosen queries to a search engine and then analyze the retrieval results. We are currently developing a tool called SEAnalyzer (Search Engine Analyzer) for discovering implementational information about a search engine. In this section, we report some of the discovering techniques that have been/are being implemented. In particular, we discuss discovering document indexing methods in Section 3.1 and discovering document term weighting schemes in Section 3.2.

3.1 Discovering Document Indexing Methods

As described in Section 2, different document indexing methods exist. In this paper, we consider the following three aspects: (1) Whether stopwords are removed. (2) Whether stemming is implemented. (3) Whether full-text or partial-text indexing is used.

3.1.1 Stopword Removal

Stopwords are non-content words such as "the" and "of" which frequently appear in most documents but do not convey much information of the documents they are in. Removing stopwords not only reduces the storage space needed to store document index but can also improve retrieval effectiveness as stopwords may result in false matches. A drawback with removing stopwords is that matches between phrases may be lost as phrases frequently consist of stopwords (e.g., the phrase "out of your mind" may become "your mind" after stopword removal). As a result, some search engines support stopword removal (e.g., AltaVista, Excite and HotBot) and some don't (e.g., Infoseek and WebCrawler). Although most stopwords are recognized universally, it is quite possible that the stopword lists used by different search engines are somewhat different due to different application domains and other considerations.

A simple method to determine whether or not stopwords are removed by a search engine is to use a few most common stopwords to form a few queries and submit them to a search engine. If no documents are retrieved, then stopwords are probably used. A more rigorous method is to first retrieve a document, say d, using any query. Then we identify commonly used stopwords in d and submit a query consisting of these stopwords. If no document is retrieved, then stopwords are removed. Otherwise, stopwords are not removed.

To determine the exact stopword list used by a search engine, we first construct a superset of the set of stopwords used by the search engine. In theory, the superset could be the set of all terms in the database of the search engine. In practice, the union of several widely used stopword lists can be used. Next, each term in the superset is used as a single-term query for the search engine. If no document is returned for a query and there exists at least one document that contains the term, then this term can be determined to be a stopword of the search engine. If some documents are retrieved, then the term is not a stopword. To reduce the number of queries that need to be evaluated in this process, we can group the stopwords in the superset (say 20 terms a group) and form a query using the words in each group. Submit each query to the search engine. Two cases could occur. In the first case, no document is returned and each of these words is known to be in the database. This indicates that all words in the query are stopwords. In the second case, some documents are returned, indicating that some words in the query are not stopwords. In this case, divide the words in the query into multiple smaller groups and repeat the above process until all actual stopwords are identified.

3.1.2 Use of Stemming

Many words have different variations. For example, the word "compute" has variations such as "computing", "computed" and "computation" (and to some extent also "computer"). Often these variations have the same or similar meaning(s). However, since they have different spellings, they cannot be matched to each other directly. By performing stemming, different variations of the same word can be mapped to the same word stem. As a result, more useful documents are likely to be retrieved for a given query.

To determine whether or not stemming is implemented by a local search engine, we proceed as follows. First, collect a few words and their variations (e.g., "compute" and its variations "computed", "computing", etc.). Next, submit one of these words, say w, as a single-term query to the local search engine. We seek one of the following cases.

1. If a document d is retrieved such that w is not in d but one or more of its variations are in d, then we can assume that stemming is implemented in the search engine.

   Note that, in this case, it is still possible that stemming is not actually implemented by the local search engine as the search engine may have implemented some *query expansion* scheme (e.g., use a thesaurus to bring variations of query terms into the query before it is processed). Nevertheless, the effect of stemming is present in this local search engine. Thus, the local search engine can still be treated as if stemming were implemented.

2. If a document d' is retrieved such that w is in d' but none of its variations are in d', then each variation of w can be used as a query to attempt to retrieve d'. If d' cannot be retrieved, then we can conclude that no stemming is done.

If none of the two cases occurs for w, then the above process is repeated for a different word until one of the above cases is encountered.

Determining exactly which stemming algorithm is used by a local search engine is a very difficult task. This is because two different stemming algorithms often differ only on the stemming of a small number of words. As a result, a large number of words may have to be examined in order to differentiate different stemming algorithms. Further research is needed for finding efficient methods to solve this problem.

3.1.3 Full-text vs. Partial-text Indexing

We assume that by now, we already know whether or not a search engine removes stopwords (as well as what words are considered as stopwords) and/or performs stemming. Without loss of generality, we assume that stopwords have been removed and stemming has been performed for the search engine under consideration.

Often, when a search engine employs partial-text indexing for its documents, it will try to index *important* terms in each document. Although the word "important" is subject to different interpretations, the following terms can be considered as important: (1) Terms with special tags (say HTML tags) such as those in the title, in headers, in bold face or large fonts, ... (2) Terms that appear near the beginning or the end of a document can be considered to be important. Texts at the two ends of a typical article usually correspond to the introduction and the conclusion of the article. (3) Terms in short documents. (4) Terms that occur frequently in a document.

Based on the above discussion, we determine whether or not partial-text indexing is employed by a local search engine as follows.

1. Submit a query to the local search engine and select a large document (say with more than 100 lines or more than 10KB in size) from the result. Let d be the selected document.

2. Remove all *important terms* from d and list the remaining terms in ascending term frequencies.

3. Use each term in the list to form a single-term query and submit it to the search engine. If d cannot be retrieved by a query, then we can conclude that partial-text indexing is used. Otherwise, if d is retrieved by every query, then full-text indexing is used.

The reason to start with terms that have low term frequencies is an attempt to reduce the number of queries to be processed to reach the conclusion. Terms with lower term frequencies are less likely to be important than terms with higher term frequencies, and therefore, they are more likely to be discarded by partial-text indexing schemes.

One potential problem in Step 3 is that d may contain an unimportant term t but d cannot be retrieved when t is used as a query. This is possible if the search engine does not return documents with very small similarities. This problem can be overcome by forming queries that contain several un-important terms.

3.2 Discovering Document Term Weighting Schemes

As discussed in Section 2, there are many possible ways to assign weights to terms in a document. Due to limited space, we will not be able to discuss how to discover all possible term weighting schemes. We will focus on a popular scheme. This term weighting scheme assigns a weight to term t in document d of a database D as follows. The weight is the product of two factors, i.e., tf-factor and *idf*-factor. The tf-factor is computed based on the term frequency (tf) of t in d using a tf formula and is an increasing function of tf. The *idf*-factor is computed based on the document frequency (*df*) of t in D using an *idf* formula and is a decreasing function of *df*. Different tf formulas and *idf* formulas exist. In addition, each formula may have one or more constant parameters that may take on different values in different systems. The following are some examples of various tf formulas and *idf* formulas.

Different tf formulas: Let $tf_t(d)$ denote the term frequency of term t in document d.

1. $a_1 + a_2 \times \dfrac{tf_t(d)}{max\_tf(d)}$ (Smart system) [28]
2. $a_1 + a_2 \times \dfrac{tf_t(d)}{tf_t(d) + a_3 + a_4 \times \frac{dl(d)}{avg\_dl}}$ (INQUERY system) [5]
3. $a_1 + a_2 \times \dfrac{a_3 + \log tf_t(d)}{a_4 + \log max\_tf(d)}$ [32]

where $max\_tf(d)$ is the maximum frequency of all terms in d, $dl(d)$ is the number of terms in document d, avg_dl is the average number of terms in a document in database D, and each a_i is a constant parameter ($i = 1, 2, 3, 4$) with $a_2 > 0$.

Different *idf* formulas: Let the df_t denote the document frequency of t in database D.

1. $\dfrac{\log \frac{(N+b_1)}{df_t}}{\log(N + b_2)}$ (INQUERY system) [6]
2. $b_1 + \log \dfrac{N - df_t}{df_t}$ [8]
3. $b_1 + b_2 \times \log \dfrac{N}{df_t}$ $(b_2 > 0)$ [2]

where N is the number of documents in database D, and b_1 and b_2 are constant parameters.

¿From the above examples, we can see that there are potentially infinite number of ways (thanks to those parameters) to compute the tf-factor and the *idf*-factor of term t. We are interested in discovering, for a given local search engine, first what formulas are used to compute the tf-factors and the *idf*-factors, and second what are the values of the constant parameters.

In order to carry out the discovery, we need to understand more precisely how similarities are computed. Conceptually, each document d is represented as a vector of weights $(w_1, w_2, ..., w_m)$, where w_i is the weight of term t_i and the term space consists of all distinct terms in a database D. Each w_i can be computed as the product of a tf-factor and an *idf*-factor of t_i as discussed above. Each user query q is also represented as a vector of weights $(q_1, q_2, ..., q_m)$ over the same term space used for documents. Note that a query term may appear multiple times in a query. As a result, q_i may not all be 0's or 1's. In fact, q_i is often computed just as the tf-factor of a term in each document is computed and most tf formulas for documents can also be used for queries. The similarity between d and q can be computed as the dot product of the two vectors, i.e., $sim(d, q) = \sum_{i=1}^{m} w_i q_i$. This simple dot product function tends to yield larger similarities for longer documents. To remedy this problem, similarities computed by the simple dot product function are often normalized by the lengths of their documents and frequently also by the length of the query. A widely used similarity function (known as the *Cosine* function [28]) that incorporates both the document length and the query length is $sim(d, q) = \dfrac{\sum_{i=1}^{m} w_i q_i}{|d| \cdot |q|}$, where $|x|$ denotes the *norm* of vector x. Note that *Cosine* function can be considered as a special case of the simple dot product function between two new vectors $(w_1/|d|, w_2/|d|, ..., w_m/|d|)$ and $(q_1/|q|, q_2/|q|, ..., q_m/|q|)$. In general, although different similarity functions with different normalization formulas exist, most can be reduced to the dot product function by computing document term weights and query term weights in a special way [20]. In this section, we assume that the similarity function is the dot product function.

Based on the above discussion, we can see that a typical search engine computes the similarity between a document and a query based on the following values: (1) query term weights (query term tf-factors), (2) document term tf-factors, (3) document term *idf*-factors, (4) document length normalization factor, and (5) query length normalization factor. (Note that it is also possible to incorporate *idf*-factors into query term weights rather than document term weights. This case is not considered in this paper for ease of presentation.) For each of the above five types of values, there is a corresponding formula with zero or more parameters. In general, we need to discover each of these formulas and their parameter values. Due to space limitation, we will only present our method for discovering the document term tf formula in this paper. The methods for discovering other formulas are similar [20]. Several reasonable assumptions will be used to facilitate the discovery. First, for a given query, the tf-factor of a term is a strictly increasing function of the tf of the term in the query. This simply means that if we increase the frequency of a term while fix the frequencies of other terms in a query, then the tf-factor of the term in the query will increase. Second, the formula for query tf-factor has already been discovered and became known. It is shown in [20] that the formula for computing query tf-factor can be discovered before other formulas are discovered.

Our methodology for discovering document tf formula consists of the following steps.

1. Create a knowledge base of different known tf formulas. This is done by surveying research papers and reports.

2. Design a set of queries and submit them to the local search engine.

3. Analyze the retrieval results to determine which formula in the knowledge base is used. If no formula in the knowledge base is found to be the correct formula, then one of the following two things can be done: (a) Declare that the discovering failed. (b) Create a new formula that can explain the retrieval results and at the same time satisfies some basic properties of tf formulas (such as producing non-negative values and being an increasing function of tf). If a formula is found, add it to the knowledge base. In this paper, we assume that one of the formulas in the knowledge base is correct.

4. Determine the values of those constant parameters in the identified formula.

Note that if all the formulas (document tf and *idf* formulas, query tf formula, document and query length normalization formulas) are known and the similarities of retrieved documents are available, then the values of all the constant parameters in these formulas can be determined when a sufficient number of documents are retrieved. This is because for each returned document, an equation involving these unknown constants can be formed. When enough equations are formed, these unknown constants can be found by solving these equations (either analytically or using numerical methods). In other words, the fourth step of the above methodology can be solved after all formulas with unknown parameter values have been identified.

The rest of our discussion concentrates on the second and the third steps of the above methodology.

The second step is carried out as follows.

(a). Find a set of terms $t_2, \ldots, t_k$ such that all of them have the same document frequency for some integer k.

(b). Find two documents d_1 and d_2 such that d_2 contains all $t_2, \ldots, t_k$ but d_1 contains none of these terms.

(c). Find a term t_1 that appears in d_1 but not in d_2.

(d). For each pair of terms t_1 and t_j $(j = 2, ..., k)$, submit a sequence of queries that contain only the two terms but with different term frequencies for them to the search engine. The objective of using these queries for a given pair of terms t_1 and t_j is to find a two-term query $q(t_1, t_j)$ such that the similarities of d_1 and d_2 to $q(t_1, t_j)$ are equal (approximately). This is possible because when we increase the frequency of t_1 in $q(t_1, t_j)$, $sim(d_1, q(t_1, t_j))$ will increase (but there will be no effect on $sim(d_2, q(t_1, t_j))$ as d_2 does not have t_1) and when we increase the frequency of t_j in $q(t_1, t_j)$, $sim(d_2, q(t_1, t_j))$ will increase (but there will be no effect on $sim(d_1, q(t_1, t_j))$ as d_1 does not have t_j). The sequence of queries is used to find the right ratio between the frequencies of t_1 and t_j in $q(t_1, t_j)$ such that $sim(d_1, q(t_1, t_j)) = sim(d_2, q(t_1, t_j))$ (approximately).

The third step of our methodology is outlined below. Because document d_1 does not have t_j $(j = 2, ..., k)$ and document d_2 does not have t_1, we have

$$sim(d_1, q(t_1, t_j)) = \frac{qtf_{t_1}(q(t_1, t_j)) \times idf_{t_1} \times dtf_{t_1}(d_1)}{n_q(q(t_1, t_j)) \times n_d(d_1)}$$

and

$$sim(d_2, q(t_1, t_j)) = \frac{qtf_{t_j}(q(t_1, t_j)) \times idf_{t_j} \times dtf_{t_j}(d_2)}{n_q(q(t_1, t_j)) \times n_d(d_2)}$$

where $qtf_t(q)$ denotes the frequency of term t in query q, idf_t denotes the idf-factor of t, $dtf_t(d)$ denotes the frequency of term t in document d, $n_q(q)$ denotes the normalization factor of query q and $n_d(d)$ denotes the normalization factor of document d. As the two similarities are the same (step (d)), equating the two expressions on the left-hand side, we obtain

$$dtf_{t_j}(d_2) = \lambda \times \frac{qtf_{t_1}(q(t_1,t_j))}{qtf_{t_j}(q(t_1,t_j))} \tag{1}$$

where $\lambda = \dfrac{n_d(d_2) \times idf_{t_1} \times dftt_{t_1}(d_1)}{n_d(d_1) \times idf_{t_j}}$, which is the same for all j as the document frequencies of $t_2, \ldots, t_k$ are the same.

Based on our assumption, the formula for computing the query term tf-factor has already been determined. ¿From the query term frequencies obtained in step (d), $\dfrac{qtf_{t_1}(q(t_1,t_j))}{qtf_{t_j}(q(t_1,t_j))}$ can be determined. Let x_j denote the computed value. Let u_j denote the term frequency of t_j in d_2. Clearly, the tf-factor of t_j in d_2, namely $dtf_{t_j}(d_2)$, is a function of u_j. We denote this function by $F(u_j)$. Using the notation just defined, from (1), we have $F(u_j) = \lambda \times x_j$ ($j = 2, \ldots, k$). By studying the $(k-1)$ pairs of values (u_j and x_j), we can often determine the form of the mathematical expression of $F()$ (or the formula for the document tf-factor). For example, if there is a linear relationship among (u_j, x_j) (i.e., the $(k-1)$ points are along a straight line), then $F()$ is a linear function of the term frequency (an example for this is the first sample tf formula given at the beginning of this subsection). More generally, if there is a linear relationship among $(\theta(u_j), x_j)$ for some known function $\theta()$, then the expression for the formula $dtf_t(d)$ is a linear function of $\theta(tf_t(d))$. The third sample tf formula given at the beginning of this subsection is an example in which $\theta()$ is the logarithm function.

The above discussion assumed that local similarities of returned documents are provided by the local search engine. A more general solution that uses only the rank order of retrieved documents can be found in [20]. We experimented with ranking documents using similarities that are computed from discovered formulas for WebCrawler. Our ranking achieved on the average 85% accuracy against the ranking generated by WebCrawler [20].

4 Usefulness of Discovered Knowledge

The detection of specific heterogeneities among multiple search engines and the identification of specific methods used in and situations associated with individual search engines can have many positive effects on building a better metasearch engine. In this section, we discuss/illustrate some of these effects.

Effects on Database Selection

The discovery of the list of terms that appear in the documents of each database and useful statistical information associated with these terms can have the following benefits.

1. The knowledge can be used to help decide whether or not database selection should be performed and what database selection method is appropriate. For example, if the databases are highly homogeneous (i.e., have the same or very similar domains), then database selection may not be useful. On the other hand, if the databases are highly heterogeneous (i.e., highly specialized), then database selection methods based on short descriptive representatives may be sufficient.

2. The database representatives produced through this discovering process will be more or less independent of the specific implementation of different search engines. As a result, using these database representatives to determine which databases should be searched is more objective and fair (i.e., *cheating* can be prevented to some extent [7]).

3. The database representatives produced through the same discovering process will contain the same types of information. This means that the same method can be used to estimate the usefulness of all databases with respect to a query.

Effects on Document Selection

As we mentioned in Section 2.2.2, one interesting issue in document selection when documents have different local and global similarities is to retrieve all potentially useful documents while minimize the retrieval of useless documents. Suppose, for a given query q, the metasearch engine sets a global threshold GT and uses a global similarity function G such that any document d that satisfies $G(q,d) > GT$ is to be retrieved (i.e., the document is potentially useful). The problem then is to determine a proper local threshold LT for each local search engine such that all potentially useful documents in the local search engine can be retrieved using its local similarity function L. That is, if $G(q,d) > GT$, then $L(q,d) > LT$. Note that in order to guarantee that all potentially useful documents be retrieved from a local system, many unwanted documents may also have to be retrieved from the local system. The challenge is to minimize the number of documents to retrieve from each local system while still guarantee that all potentially useful documents are retrieved. In other words, for a given query and a local database, it is desirable to determine the tightest (largest) local threshold LT such that if $G(q,d) > GT$, then $L(q,d) > LT$.

In [15, 24], several techniques are proposed to tackle the above problem. However, all these solutions require that we know how similarities are computed in the local search engine. This means that the discovery of similarity functions and other formulas used in local search engines can help solve the above document selection problem.

Effects on Result Merging

As discussed in Section 2, one difficulty with merging returned documents into a single ranked list is that local similarities may be incomparable because the documents may be indexed differently and the similarities may be computed using different methods (term weighting schemes, similarity functions, etc.). If we know the specific document indexing and similarity computation methods used in different local search engines, 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 can become more comparable with others; and (3) how to compute new and comparable similarities. This is illustrated by the following example.

Example 1 Suppose it is discovered that all the local search engines selected for answering a user query employ the same methods for indexing local documents and computing local similarities, and the *idf* information is not used (i.e., *idf*-factor is 1), then the similarities from these local search engines can be considered as comparable and be used directly to merge the returned documents.

If the only difference among these local 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. As an example, suppose a term t in query q is a stopword in local search engine e1 but not a stopword in local search engine e2. In order to generate more comparable similarities, we can remove t from q and submit the modified query to e2 (it does not matter whether the original q or the modified q is submitted to e1).

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. Note that ideally global similarities of documents should be used to rank the returned documents. Consider the following two cases.

Case 1: Query q consists of a single term t. The similarity of q with a document d in a local database can be computed by $sim(d,q) = \frac{qtf_t(q) \times lidf_t \times dtf_t(d)}{n_q(q) \times n_d(d)}$, where $lidf_t$ is the local *idf*-factor of t (see Section 3.2 for other notations). If the local *idf* formula has been discovered and the global document frequency of t is known (it can be estimated from the local document frequencies of t in all local search engines), then this similarity can be adjusted to a global similarity by multiplying it by $\frac{gidf_t}{lidf_t}$, where $gidf_t$ is the global *idf*-factor of t.

Note that if some local search engines employ stemming but some do not, then we need to be careful about determining the local document frequencies of a term. For example, if stemming is not used in local search engine e1 but in other local search engines. Then the desired local *df* of each query term t in e1 (i.e., the *df* used to compute the $lidf_t$ in the above adjustment) should be the number of documents in e1 that contain at least one of the variations of t. This *df* can be estimated from the *df*s of the variations of t in e1 under some assumptions.

Case 2: Query q has multiple terms $t_1, ..., t_m$. The global similarity between d and q is $s = \frac{\sum_{i=1}^{m} qtf_{t_i}(q) \times gidf_{t_i} \times dtf_{t_i}(d)}{n_q(q) \times n_d(d)} = \sum_{i=1}^{m} \frac{qtf_{t_i}(q)}{n_q(q)} \cdot \frac{dtf_{t_i}(d)}{n_d(d)} \cdot gidf_{t_i}$. Since we know all the formulas, $\frac{qtf_{t_i}(q)}{n_q(q)}$ and $gidf_{t_i}$, $i = 1, ..., m$, can all be computed by the metasearch engine. Therefore, in order to find s, we need to find $\frac{dtf_{t_i}(d)}{n_d(d)}$, $i = 1, ..., m$. To find $\frac{dtf_{t_i}(d)}{n_d(d)}$ for a given i without retrieving document d, we can submit t_i as a single-term query. Let $s_i = sim(d, q(t_i)) = \frac{qtf_{t_i}(q(t_i)) \times lidf_{t_i} \times dtf_{t_i}(d)}{n_q(q(t_i)) \times n_d(d)}$ be the local similarity returned. Then $\frac{dtf_{t_i}(d)}{n_d(d)} = \frac{s_i \times n_q(q(t_i))}{qtf_{t_i}(q(t_i)) \times lidf_{t_i}}$. Note that the right-hand 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, m additional single-term queries can be used to compute the global similarities between q and all documents retrieved by q.

5 Conclusions

In this paper, we identified various heterogeneities unique to heterogeneous multiple text database systems (search engines) and analyzed the impact of these heterogeneities on building an effective and efficient metasearch engine. We also presented techniques based on the query sampling method for detecting various heterogeneities among multiple search engines. We also discussed/illustrated the usefulness of discovered knowledge in solving various problems in metasearch engines.

To understand the various aspects of each local search engine is essential to developing effective and efficient metasearch engines. Using sampling queries to discover various needed knowledge about a search engine is a promising approach. Very little research on this technique has been reported so far. Further research is needed to find more efficient and more automated algorithms for more knowledge in this area.

Acknowledgement: This work is supported in part by the following NSF grants: CCR-9816633 and CCR-9803974.

References

[1] C. Baumgarten. *A Probabilistic Model for Distributed Information Retrieval.* ACM SIGIR Conference, pp. 258-266, 1997.

[2] M. Boughanem, and C. Soulé-Depuy. *Mercure at TREC-6.* Sixth Text REtrieval Conference (TREC-6), pp. 187-193, 1997.

[3] J. Boyan, D. Freitag, and T. Joachims. *A Machine Learning Architecture for Optimizing Web Search Engines.* AAAI Workshop on Internet-based Information Systems, Portland, Oregon, 1996.

[4] S. Brin, and L. Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine.* WWW7 Conference, 1998.

[5] J. Broglio, J. Callan, W. B. Croft, and D. Nachbar. *Document Retrieval and Routing Using the INQUERY system.* Third Text REtrieval Conference. NIST Special Publication 500-225, 1994.

[6] J. Callan, Z. Lu, and. W. Croft. *Searching Distributed Collections with Inference Networks.* ACM SIGIR, 1995, pp. 21-28.

[7] J. Callan, M. Connell, and A. Du. *Automatic Discovery of Language Models for Text Databases.* ACM SIGMOD Conference, 1999.

[8] W. B. Croft. *Experiments with Representation in a Document Retrieval System.* Information Technology: Research and Development, 2(1), pp. 1-21, 1983.

[9] M. Cutler, Y. Shih, and W. Meng. *Using the Structures of HTML Documents to Improve Retrieval.* USENIX Symposium on Internet Technologies and Systems (NSITS'97), Monterey, California, 1997.

[10] D. Dreilinger, and A. Howe. *Experiences with Selecting Search Engines Using Metasearch.* ACM TOIS, 15(3), July 1997, pp. 195-222.

[11] S. Dumais. Latent Semantic Indexing (LSI) and TREC-2. TREC-2 Conference, 1994, pp. 105-115.

[12] M. Garcia-Solace, F. Saltor, and M. Castellanos, *Semantic Heterogeneity in Multidatabase Systems*, in OO Multidatabase Systems, edited by O. Bukhres and A. Elmagarmid, Prentice Hall, 1996.

[13] L. Gravano, and H. Garcia-Molina. *Generalizing GlOSS to Vector-Space databases and Broker Hierarchies.* VLDB, 1995.

[14] L. Gravano, and H. Garcia-Molina. *Generalizing GlOSS to Vector-Space databases and Broker Hierarchies.* Technical Report, Computer Science Dept., Stanford University, 1995.

[15] L. Gravano, and H. Garcia-Molina. *Merging Ranks from Heterogeneous Internet Sources.* Very Large Data Bases Conference, 1997.

[16] B. Kahle, and A. Medlar. *An Information System for Corporate Users: Wide Area Information Servers.* Technical Report TMC199, Thinking Machine Corporation, April 1991.

[17] W. Kim, I. Choi, S. Gala, and M. Scheevel, *On Resolving Schematic Heterogeneity in Multidatabase Systems*, in Modern Database Systems edited by W. Kim, Addison-Wesley, 1995.

[18] M. Koster, *ALIWEB: Archie-Like Indexing in the Web*. Computer Networks and ISDN Systems, 27:2, 1994, pp. 175-182.

[19] K. Kwok, L. Grunfeld, and D. Lewis, *TREC-3 Ad-hoc, Routing Retrieval and Thresholding Experiments Using PIRCS*. TREC-3, Gaithersburg, 1995.

[20] K. Liu, W. Meng, C. Yu, and N. Rishe. *Discovery of Similarity Computations in the Internet*. Technical Report, Department of EECS, University of Illinois at Chicago, 1999.

[21] U. Manber, and P. Bigot. *The Search Broker*. USENIX Symposium on Internet Technologies and Systems (NSITS'97), Monterey, California, 1997, pp. 231-239.

[22] M. Mauldin. *Lycos: Design Choices in An Internet Search Service*. IEEE Expert Online, February 1997.

[23] O. McBryan. *GENVL and WWWW: Tools for Training the Web*. WWW1 Conf., Geneva, 1994.

[24] W. Meng, K. Liu, C. Yu, X. Wang, Y. Chang, N. Rishe. *Determine Text Databases to Search in the Internet*. International Conference on Very Large Data Bases, New York City, August 1998, pp. 14-25.

[25] W. Meng, K. Liu, C. Yu, W. Wu, and N. Rishe. Estimating the Usefulness of Search Engines. 15th International Conference on Data Engineering (ICDE'99), Sydney, Australia, March 1999.

[26] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*. New York: McCraw-Hill, 1983.

[27] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison Wesley, 1989.

[28] G. Salton and C. Buckley. *Term-Weighting Approaches in Automatic Text Retrieval*. Information Processing and Management, 24(5), pp. 513-523, 1988.

[29] E. Selberg, and O. Etzioni. *The MetaCrawler Architecture for Resource Aggregation on the Web*. IEEE Expert, 1997.

[30] M. Sheldon, A. Duda, R. Weiss, J. O'Toole, and D. Gifford. *A Content Routing System for Distributed Information Servers*. 4th Int'l Conf. on Extending Database Technology, Cambridge, England, 1994.

[31] A. Sheth, and L. Larson. *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*. ACM Computing Surveys, 22:3, September 1990, pp. 183-236.

[32] A. Singhal, C. Buckley. and M. Mitra. *Pivoted Document Length Normalization*. ACM SIGIR Conference, Zurich, 1996.

[33] E. Voorhees, N. Gupta, and B. Johnson-Laird. *The Collection Fusion Problem*. TREC-3 Conference, Gaithersburg, 1995.

[34] S. Wade, P. Willett, D. Bawden. *SIBRIS: the Sandwich Interactive Browing and Ranking Information System*. Journal of Information Science, 15, 1989, pp. 249-260.

[35] C. Yu, K. Liu, W. Wu, W. Meng, and N. Rishe. *Finding the Most Similar Documents across Multiple Text Databases*. IEEE Conference on Advances in Digital Libraries (ADL'99), Baltimore, Maryland, May 1999.

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

[37] B. Yuwono, and D. Lee. *Server Ranking for Distributed Text Resource Systems on the Internet*. 5th Int'l Conf. On DB Systems For Adv. Appli. (DASFAA'97), Melbourne, Australia, April 1997, pp. 391-400.