Efficient and Effective Metasearch for a Large Number of Text Databases*

Clement Yu[1], Weiyi Meng[2], King-Lup Liu[1], Wensheng Wu[1], Naphtali Rishe[3]

[1] Department of Electrical Engineering and Computer Science
University of Illinois at Chicago, Chicago, IL 60607
Email: yu@eecs.uic.edu

[2] Department of Computer Science
State University of New Yort at Binghamton, Binghamton, NY 13902

[3] School of Computer Science, Florida International University, Miami, FL 33199

April 28, 1999

Abstract

Metasearch engines can be used to facilitate ordinary users for retrieving information from multiple local sources (text databases). In a metasearch engine, the contents of each local database is represented by a representative. Each user query is evaluated against the set of representatives of all databases in order to determine the appropriate databases to search. When the number of databases is very large, say in the order of tens of thousands or more, then a traditional metasearch engine may become inefficient as each query needs to be evaluated against too many database representatives. Furthermore, the storage requirement on the site containing the metasearch engine can be very large. In this paper, we propose to use a hierarchy of database representatives to improve the efficiency. We provide an algorithm to search the hierarchy. We show that the retrieval effectiveness of our algorithm is the same as that of evaluating the user query against all database representatives. We also show that our algorithm is efficient. In addition, we propose an alternative way of allocating representatives to sites so that the storage burden on the site containing the metasearch engine is much reduced.

Keywords: Metasearch, Distributed Collection, Representative Hierarchy, Information Retrieval.

*This research is supported by the following organizations: NSF (IRI-9509253, CDA-9711582, HRD-9707076), NASA (NAGW-4080, NAG5-5095) and ARO (NAAH04-96-1-0049, DAAH04-96-1-0278).

1 Introduction

The Internet has become a vast information resource in recent years. To help ordinary users find desired data in this environment, many *search engines* have been created. 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 inverted file index for all documents in the database is created and stored in the search engine. For each *term* which can represent a significant word or a combination of several (usually adjacent) significant words, this index can identify the documents that contain the term quickly.

Frequently, the information needed by a user is 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. Substantial effort would be needed for the user to search each database and identify useful papers from the retrieved papers. 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 each of its underlying search engines (will be called the *representative* of the search engine/database) in order to provide better service. When a metasearch engine receives a user query, it first passes the query to the appropriate local search engines, and then collects (sometimes, reorganizes) the results from its local search engines. With such a metasearch engine, only one query is needed from the above user to invoke multiple search engines.

A closer examination of the metasearch approach reveals the following problems. If the number of local search engines in a metasearch engine is large, then it is likely that for a given query, only a small percentage of all search engines may contain sufficiently useful documents to the query. In order to avoid or reduce the possibility of invoking useless search engines for a query, we should first identify those search engines that are most likely to provide useful results to the query and then pass the query to only the identified search engines. Examples of systems that employ this approach include WAIS [12], ALIWEB [14], gGlOSS [7], SavvySearch [10], ProFusion [6, 5] and D-WISE [34]. The problem of identifying potentially useful databases to search is known as the *database selection problem.* Database selection is done by comparing each query with all database representatives. If a user only wants the m most similar documents across all local databases, for some positive integer m, then the m documents to be retrieved from the identified databases need to be specified and retrieved. This is the *collection fusion problem.*

In this paper, we present an integrated solution to the database selection problem and the collection fusion problem. When the number of databases is not too large, say no more than 100, we employ a two-level hierarchy of database representatives, where the root node represents the representative for the virtual global database that logically contains the documents in all local databases and each leaf node is a representative for a local database. A necessary and sufficient condition to rank the databases optimally is given. Experimental results are provided to show the superior performance of our approach. The experimental results essentially demonstrate that retrieval using our method in distributed environment will yield the same result as if all data were stored in one site. When the number of local databases is large, the two-level hierarchy is not efficient. In this case, we organize the database representatives into a hierarchy of more than 2 levels. We provide an efficient *best-first search* algorithm to search the hierarchy. We show that the search yields exactly the same retrieval results as if all database representatives were placed in a two-level hierarchy. Since the retrieval performance for the two-level hierarchy is very good, this indicates that our best-first search algorithm is guaranteed to yield equally good retrieval effectiveness. We also show that the searching process is efficient.

The rest of the paper is organized as follows. In Section 2, we present our solution to the database selection problem and the collection fusion problem when the database representatives are organized into a two-level hierarchy. In Section 3, our solution is extended to a hierarchy of more than two levels. The effectiveness and efficiency issues of our algorithm are addressed. We conclude the paper in Section 4.

1.1 Related Works

In the last several years, a large number of research papers on issues related to metasearch engine or distributed collections have been published. In this section, only the research results most closely related to ours and their differences are identified. Please see [20] for a more comprehensive review of other work in this area.

1. The gGlOSS project [7] is similar in the sense that it ranks databases according to some measure. However, there is no necessary and sufficient condition for optimal ranking of databases; there is no precise algorithm for determining which documents from which databases are to be returned. Organizing representatives in a hierarchy and searching them were mentioned in [7], but a precise algorithm for searching was lacking and there was no evidence that searching a hierarchy of more than two levels would yield the same retrieval effectiveness of searching a two-level hierarchy.

2. A necessary and sufficient condition for ranking databases optimally was given in [13]. However, [13] considered only the databases and queries that are for structured data. In contrast, unstructured text data is considered in this paper. In [2], a theoretical framework was provided for achieving *optimal* results in a distributed environment. However, such an approach

depended on parameters which may be difficult to estimate in practice and no experimental results were provided.

3. In [27], experimental results were given to demonstrate that it was possible to retrieve documents in distributed environments with essentially the same effectiveness as if all data were in one site. However, the results depended on the existence of a *versatile collection* where terms related to any given query needed to be extracted and then added to the query before an actual retrieval of documents take place. In the Internet environment where data are highly heterogeneous, it is unclear whether such a versatile collection can in fact be constructed. Even if such a collection can be constructed, the storage penalty should be very high in order to accommodate the heterogeneity. Furthermore, no algorithm for searching a hierarchy of representatives for large number of databases was given.

2 Retrieval from a Two-Level Hierarchy

In Section 2.1, the two-level metasearch architecture approach is reviewed. This architecture is suitable for a moderate number of search engines. Using this architecture, a representative which indicates the contents of a database is constructed for each local database and a representative for the global database is also formed. Each user query is compared against these database representatives. The databases are ranked such that databases which are most likely to contain the most similar documents to the user query are placed ahead of other databases. In Section 2.2, we provide a necessary and sufficient condition for ranking the databases optimally. The condition is simple: The databases should be arranged in descending order of the similarity of the most similar document in each database. In Section 2.3, we describe how the similarity of the most similar document in a database can be estimated from the user query, its database representative and the representative for the global database. Having the databases ranked, a strategy for retrieving documents from individual databases, for stopping searching additional databases and for merging the retrieved documents from the searched databases is sketched in Section 2.4. Experimental results are provided in Section 2.5. For ease of presentation, we assume that the similarities of documents to a given query are distinct.

A query in this paper is simply a set of words submitted by a user. It is transformed into a vector of *terms* with *weights* [21], where a term is essentially a content word and the dimension of the vector is the number of all distinct terms. When a term appears in a query, the component of the query vector corresponding to the term, which is the *term weight*, is positive; if it is absent, the corresponding term weight is zero. The weight of a term usually depends on the number of occurrences of the term in the query (relative to the total number of occurrences of all terms in the query) [21, 32]. This is the term frequency weight. The weight of a term may also depend on the number of documents having the term relative to the total number of documents in the database, which is the inverse document frequency weight. A document is similarly transformed into a vector with weights. The similarity between a query and a document can be measured by the dot product of their respective vectors. Often, the dot product is divided by the product of the *norms* of the two vectors, where the norm of a vector $(x_1, x_2, ..., x_n)$ is $\sqrt{\sum_{i=1}^{n} x_i^2}$. This is to normalize the similarity between 0 and 1. The similarity function with such a normalization is known as the $Cosine$ function [21, 32]. For the sake of concreteness, we shall use in this paper the version of the $Cosine$ function [26] where the query uses the inverse frequency weight and the term frequency weight and the document uses the term frequency weight. Other similarity functions, see for example [24], are also possible.

2.1 Two-Level Architecture of Metasearch

In this architecture, the highest level (the root node) contains the representative for the global database. There is only one additional level. This level contains a representative for each local database. When a user query is submitted, it is processed first by the metasearch engine against all these database representatives. The databases are then ranked. Finally, the metasearch engine invokes a subset of the search engines and co-ordinates the retrieval and merging of documents from individual search engines.

2.2 Optimal Ranking of Databases

We assume that a user is interested in retrieving the m most similar documents to his/her query for a given m. Databases $[D_1, D_2, \ldots, D_p]$ are ranked optimally if there exists a k such that the first k databases $D_1, D_2, \ldots, D_k$ contains the m most similar documents and each of these databases contains at least one such document. A necessary and sufficient condition to rank databases optimally [31] is as follows.

- Databases $[D_1, D_2, \ldots, D_p]$ are ranked optimally if the similarity of the most similar document in D_i is larger than that of the most similar document in D_j, if $i < j$.

Example 1 Suppose there are 4 databases. For a query Q, suppose the similarities of the most similar documents in databases D_1, D_2, D_3 and D_4 are 0.8, 0.5, 0.7 and 0.2, respectively. Then the databases should be ranked $[D_1, D_3, D_2, D_4]$. ■

This result applies to any similarity function as well as any function which assigns degrees of relevance to documents.

2.3 Estimation of the Similarity of the Most Similar Document

We now estimate the similarity of the most similar document in a database using its representative, the representative of the global database and the query. The representative of a database consists of all terms in the database. For each term, two quantities are kept. They are the *maximum normalized weight* and the *average normalized weight* of the term. For term t_i, they are denoted by mnw_i and aw_i, respectively. mnw_i is $\max\{d_i/|d|\}$, where $|d|$ is the norm (length) of the document $d = (d_1, ..., d_i, ..., d_n)$ and the maximum is over all documents in the database. $d_i/|d|$ is the normalized weight of the ith term. aw_i is simply the average of such values over all documents in the database, including documents not having the term. For the representative of all databases, it also consists of all terms appearing in any of the databases. For each term, only one quantity, namely the document frequency (the number of documents having the term in the global database) of the term, is kept. This generates a weight which is known as the inverse document frequency of the term and is denoted by idf_i for term t_i.

The similarity of the most similar document of a database D with respect to a query $q = (q_1 * idf_1, \ldots, q_k * idf_k)$, where q_i is the term frequency of the i-th query term, is estimated by

$$est(D) = \max_{1 \le i \le k} \Big\{ \sum_{\substack{j=1 \\ j \ne i}}^{k} q_j * idf_j * aw_j \; + \; q_i * idf_i * mnw_i \Big\} / |q| \tag{1}$$

The intuition for having this estimate is that the most similar document in a database is likely to have the maximum normalized weight of the ith query term, for some i. This yields the second half of the above expression within the braces. For each of the other query terms, the document takes the average normalized value. This yields the first half. Then, the maximum is taken over all i, since the most similar document may have the maximum normalized weight of any one of the k query terms. Normalization by the query norm, $|q|$, yields a value less than or equal to 1. We shall drop $|q|$ for ease of presentation. This will not have any impact as the relative values of the different databases are not changed. The estimate given here is slightly different from our earlier estimate in [31]. It is easy to see that $est(D)$ can be computed in time linear to the number of terms in the query. The estimate can be further improved by combining two terms which appear in adjacent positions in some previously processed queries. This is used in our experiments described in Section 2.5.

2.4 Coordinate the Retrieval from Different Search Engines

Suppose the databases have been ranked in the order $[D_1, \ldots, D_p]$, we now briefly review an algorithm [30] which determines (1) the value of k such that the first k databases are searched, and (2) which documents from these k databases should be used to form the list of m documents to be returned to the user. Suppose the first s databases have been invoked from the metasearch engine. Each of these search engines returns the similarity of the most similar document to the metasearch engine which computes the minimum of these s values. Each of the s search engines returns documents to the metasearch engine whose similarities are greater than or equal to the minimum value. If m or more documents have been returned, then they are sorted in descending order of similarity and the first m documents are returned to the user. Otherwise, the next database will be invoked and the process is repeated until m documents are returned to the user. It has been shown that if the databases have been ranked optimally (with the databases containing the desired documents ahead of other databases) for a given query, then this algorithm will retrieve all the m most similar documents with respect to the query.

2.5 Experimental results

15 databases are used in our experiments. These databases are formed from articles posed to 52 different newsgroups in the Internet. These articles were collected at Stanford University [7]. Each newsgroup that contains more than 500 articles forms a separate database. Smaller newsgroups are merged to produce larger databases. The table below shows the number of documents in each database.

database	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
#docs	761	1014	705	682	661	622	526	555	629	588	558	526	607	648	564

There are altogether 6,597 queries submitted by real users. Both the data and the queries were used in the gGlOSS project [7]. From these 6,597 queries, we obtain two subsets of queries. The first subset consists of the first 1,000 queries, each having no more than 6 terms. They will be referred later as *short queries.* The second subset consists all queries having 7 or more terms. There are 363 *long queries.*

The following tables provide a comparison of our method with the high-correlation method in gGlOSS [7], where each number in the table represents the percentage of the m most similar documents retrieved by a method.

m	our method	HC method
5	98.41%	69.73%
10	99.29%	75.64%
20	99.58%	82.87%
30	99.70%	87.68%

(a) Short Queries

m	our method	HC method
5	90.22%	61.44%
10	93.58%	67.64%
20	97.09%	75.82%
30	98.54%	81.82%

(b) Long Queries

For short queries, the improvements of our method over the high-correlation method range from 41.1% to 13.7%. For long queries, the improvements range from 46.8% to 20.4%. Furthermore, for short queries, our method retrieves on the average 98.4% to 99.7% of the most similar documents.

Since Internet queries are typically very short (the average number of terms per Internet query is about 2.2), our result indicates that it is possible to retrieve the most similar documents from multiple databases with essentially the same effectiveness as if all documents were stored in one database. The number of databases searched by our algorithm is only 7.5% to 14.0% more than the number of databases which contain the m most similar documents and the number of documents transmitted is about 11.2% to 24.2% beyond m.

3 Searching a General Hierarchy of Database Representatives

3.1 Definition of Representatives of Superdatabases

In the last section, we had a two-level architecture for database representatives. This architecture is suitable for a moderate number of search engines (or databases — we shall use search engine or database interchangeably since logically each search engine retrieves documents from a logical database), say 100 databases. However, when the number of databases is very large, say thousands or tens of thousands, then there will be storage and efficiency problems. First, the amount of storage to contain all database representatives could be enormous. Second, estimating the similarity of the most similar document for each database could be time consuming for large number of databases. For these reasons, we introduce a general hierarchy of database representatives and a search algorithm for such a hierarchy so that the number of estimations can be significantly reduced. We will also address the storage issue.

We first define this hierarchy of database representatives. As in the two-level architecture, the lowest level contains all representatives of individual local databases. Local databases can be logically grouped into *superdatabases*. For example, if superdatabase S_1 contains databases D_1, D_2 and D_3, then all documents in these databases are logically contained in S_1. Physically, superdatabases do not exist. The next level of the hierarchy contains representatives of superdatabases formed from local database representatives directly. The representative of a superdatabase will be called a *super-representative*. Each super-representative is constructed from the representatives which are one level below them and is kept physically. The super-representative, RS, which is constructed from a set, R, of representatives, consists of two quantities for each term which appears in any of the representatives in R. The two quantities are the *maximum normalized weight* and the *maximum averaged normalized weight* of the term. Let the maximum normalized weights of term t_i in the j-th representative in R be mnw_{ij}. Then, the maximum normalized weight of term t_i in the super-representative RS is $\max_{all\ j\ in\ R} mnw_{ij}$. Similarly, the maximum average normalized weight of term t_i in the super-representative RS is obtained by taking the maximum of the corresponding quantities in the component representatives. If the number of super-representatives at a given level

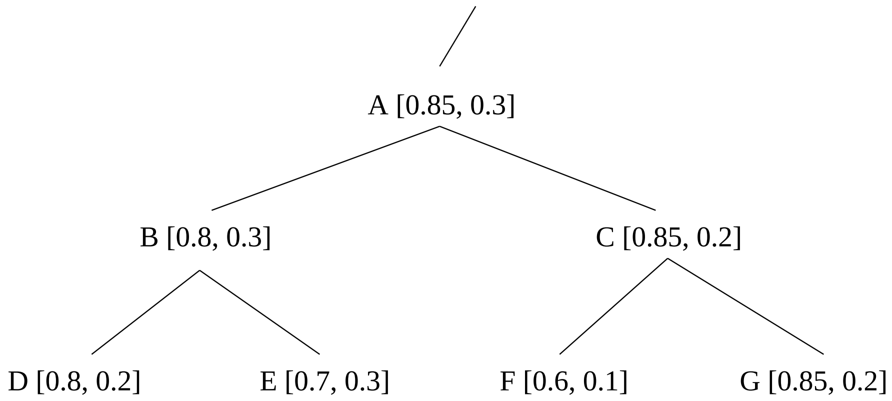

Figure 1: Illustrating the computation of super-representative for one term.

is still too large, then they can be grouped into higher level super-representatives by repeating the above process which ends when the number of super-representatives below the root node is sufficiently small. The root node representative contains the same information as that in the root node of the two-level hierarchy case discussed in Section 2, i.e., the global document frequency of each term which appears in any local database is kept.

Figure 1 illustrates the process of deriving the quantities for a single term in a hierarchy. The first quantity associated with each node is the maximum normalized weight and the second quantity is the maximum average normalized weight.

3.2 An Algorithm for Searching the Hierarchy

Let RS be the super-representative for superdatabase S that logically contains local databases $D_1, \ldots, D_r$. The similarity of the most similar document with the query $q = (q_1 * idf_1, \ldots, q_k * idf_k)$ in S can be estimated using RS and the root representative as follows. Let the estimate be

$$est(S) = \max_{1 \leq i \leq k} \left\{ mnw'_i * idf_i * q_i + \sum_{\substack{j=1 \\ j \neq i}}^{k} maw_j * idf_j * q_j \right\} \tag{2}$$

where $mnw'_i = \max\{mnw_i(1), \ldots, mnw_i(r)\}$, $mnw_i(e)$ is the maximum normalized weight of term t_i in database D_e, $1 \leq e \leq r$; $maw_j = \max\{aw_j(1), \ldots, aw_j(r)\}$, $aw_j(e)$ is the average normalized weight of term t_j in the database D_e, $1 \leq e \leq r$.

For every database D_e, we apply equation (1) to estimate the similarity of the most similar document in D_e. The estimate is $est(D_e)$ and is given by

$$\max_{1 \leq i \leq k} \left\{ mnw_i(e) * idf_i * q_i + \sum_{\substack{j=1 \\ j \neq i}}^{k} aw_j(e) * idf_j * q_j \right\} \tag{3}$$

Since $mnw'_i \geq mnw_i(e)$ and $maw_j \geq aw_j(e)$, by comparing equation (2) with equation (3), we have $est(S) \geq est(D_e)$. This result is summarized as follows.

Lemma 1 *Let S be a superdatabase containing databases $D_1, \ldots, D_r$. Let $est(S)$ and $est(D_e)$ be the estimated similarity of the most similar document in S and in D_e, respectively. Then, $est(S) \geq est(D_e)$.*

Suppose superdatabase S and other superdatabases are contained in a superdatabase T. Recall that the representative of T is formed from the representatives of the children of T in exactly the same way that the representative of S is formed from the representatives of $D_1, \ldots, D_r$. Thus, the estimated similarity of the most similar document in T, denoted by $est(T)$, satisfies $est(T) \geq est(S)$.

Consider a path P containing $A_1, A_2, \ldots, A_s$ in the hierarchy, where each A_i represents the representative of a superdatabase or a database and A_{i+1} is a child of A_i. With a slight abuse of notation, we also use A_i to denote a database since there is a 1–1 correspondence between a database and its representative. Based on our discussion, we have $est(A_i) \geq est(A_{i+1})$ for any given query. By transitivity of inequality, we have the following lemma.

Lemma 2 *In the hierarchy of representatives, if A_i is an ancestor of A_j, then $est(A_i) \geq est(A_j)$ for any given query.*

This property will allow us to devise a best-first search strategy which ranks databases exactly the same as if the similarity of the most similar document in each database D_i were computed and then the databases were ranked in descending order of $est(D_i)$. This best-first strategy avoids the computation of $est(D_j)$ for most databases D_j.

We now provide an algorithm to search this hierarchy of representatives. The main idea of the best-first search algorithm is as follows. For a given query, from the children of the root, it selects the representative which yields the largest estimated similarity. If it is the representative of a local database, then the corresponding search engine is invoked and documents are retrieved in the way as described in Section 2.4. If it is a super-representative of an intermediate node in the hierarchy, then the similarity of the most similar document in each of its child representatives is estimated. These child representative nodes are arranged in descending order of similarity and merged with the current list of nodes in descending order of similarity to form a list of representative nodes in the same order. In this list, the estimated similarities can be due to the representatives of local databases or from the non-leaf super-representatives. But, we always take the largest value. If the largest value is from the representative of a local database, then the corresponding search engine is invoked and documents are retrieved according to Section 2.4; otherwise, the best-first search process is executed on the node with the largest estimated similarity value. The details are as follows.

Search(m, q, Root) /* m is the number of documents to be retrieved; q is the query;
Root is the root node of the hierarchy */

1. Initialization: *min-sim := 1*; /* the minimum of the similarities of the retrieved documents from previously searched databases is initially set to be 1 (the highest possible similarity) */
2. The similarity of the most similar document in each child of the root node, *Root*, is estimated. These child nodes are arranged in a list L in descending order of the estimated similarities.
3. The first node, say N, is removed from L.
   If it is the representative of a local database D, then
   Step (3.1): {
   (a) local database D is searched;
   (b) the most similar document and its similarity with q, *csim*, are returned to the result-merger;
   (c) If *csim* > *min-sim* then {
      i. send from database D all documents with similarity $\geq$ *min-sim* to the result-merger;
      ii. if m or more documents have been received by the result-merger, then take the m most similar documents and stop;

   }
   else {for each local database D' which has been searched, do
      i. send all documents from D' which have similarities $\geq$ *csim* (but have not been transmitted) to the result-merger;
      ii. *min-sim := csim*;
      iii. if m or more documents have been received by the result-merger, then take the m most similar documents and stop;

   } } /* end of step (3.1) */

   else { Step (3.2):
   (a) the similarity of the most similar document in each child of N is estimated;
   (b) these child nodes are arranged in a list L_1 in descending order of estimated similarity;
   (c) L and L_1 are merged to form L in descending order of estimated similarity;

}

4. repeat Step (3);

The *result merger* mentioned in the algorithm is a software component of the metasearch engine which collects the transmitted documents from the searched databases to form the list of m documents to present to the user.

3.3 Effectiveness of Algorithm Search(m, q, Root)

In Algorithm $Search(m, q, Root)$, the result-merger gathers the documents retrieved from various local databases until m or more documents have been received. Step (3.1) is exactly the same process for deciding which documents from the selected databases will form the m documents to present to the user as that described in Section 2.4. In the two-level hierarchy, the databases (the leaf-nodes) are arranged in a list in descending order of estimated similarity and then Step (3.1) is executed to those databases in the ordered list. In Algorithm $Search(m, q, Root)$, the leaf-nodes and the intermediate nodes are interleaved in the list of nodes in descending order of estimated similarity. Whenever a leaf-node, i.e., a local database is reached, Step (3.1) is executed to determine the documents to retrieve from the local database. Whenever a non-leaf node is encountered, it is replaced by its children. If the leaf-nodes executed by Step (3.1) are exactly in the same order as if databases were arranged in descending order of estimated similarity, then this algorithm will give the same retrieval performance of a two-level hierarchy as described in Section 2. The following proposition establishes this fact.

Proposition 1 *Consider any two local databases D_i and D_j. Suppose the estimated similarity of the most similar document in D_i is higher than that of the most similar document in D_j, i.e., $est(D_i) > est(D_j)$. Then Step (3.1) of Algorithm Search(m, q, Root) will execute on database D_i before it executes on database D_j.*

Proof: Consider the parent P of database D_j in the hierarchy. Two cases may occur:
Case 1: D_i's parent is the root node. If $est(D_i) > est(P)$, then D_i will be ahead of P in the list L. As a result, D_i will be executed by Step (3.1) before D_j. Even if $est(P) > est(D_i)$, Step (3.2) will compute $est(D_j)$. Since $est(D_i) > est(D_j)$, D_i will be ahead of D_j in the list L, due to Step (3.2)(c).

Case 2: D_i's parent is not the root. Let C be a child of the root and also an ancestor of D_i. Consider the path, PATH, from C to D_i. For each node N in the PATH, including C but excluding D_i, $est(N) \geq est(D_i)$ by Lemma 2. Since $est(N) \geq est(D_i) > est(D_j)$, repeated application of Step 3 will place C, then the child of C in PATH, followed by the grandchild of C in PATH and eventually D_i in the list L. All these nodes are ahead of D_j, due to the inequality. Thus, D_i will be executed before D_j. ■

Observation: This proposition guarantees that the databases will be searched in descending order of estimated similarity using our method of estimating similarity of the most similar document in a database or superdatabase. The same result holds for any other estimation method as long as the estimation method is a non-decreasing function of the two parameters, namely the maximum normalized weight and the maximum average normalized weight (the average normalized weight in the case of a local database).

3.4 Efficiency of Algorithm Search(m, q, Root)

We first give a simple example to illustrate the efficiency of searching a hierarchy in comparison to having a two-level architecture.

Suppose there are 900 disjoint databases. Every 30 databases are grouped into a superdatabase. Then, there will be three levels. The root is at the highest level; the second level has 30 superdatabases (actually their super-representatives); the third level consists of the representatives of the 900 local databases. If the two-level architecture were used, all 900 estimation computations would have been carried out for each query. In the three-level architecture, 30 estimation computations are carried out for the 30 superdatabases. For the superdatabase having the largest estimated similarity, another 30 estimation computations are done for the local databases. Suppose the estimated similarity of a rather highly ranked local database is less than the estimated similarity of the most similar document in another superdatabase, then an additional 30 estimation computations will be carried out. Suppose the m most similar documents are found from these 60 local databases that are searched, then altogether there are 90 estimation computations. In comparison to the 900 estimation comparisons, one order of magnitude of savings is achieved.

In general, if there are h levels in a hierarchy and each superdatabase is made up of r databases or superdatabases of the next level. Then, there are altogether r^{h-1} local databases. In a two-level architecture, this requires r^{h-1} estimation computations. Suppose using the best-first search algorithm, there are r estimation computations for the children of the root node, $2r$ estimation computations for their children, $\ldots, (i-1) * r$ estimation computations for the nodes at the ith level and finally $(h-1) * r$ estimation computations at the local database level. Thus, altogether the number of estimation computations is $r + 2r + \cdots + (h-1) * r = r * h * (h-1)/2$. For $h = 3$ and $r = 30$, the number is 90, as given by the above example. For $h = 4$ and $r = 30$, the two-level architecture takes 27,000 computations while the four-level hierarchy takes 180 computations. As h increases, the savings will be magnified.

We now establish a result to show that our best-first search algorithm is reasonably efficient. Let s be the number of databases containing the m most similar documents. Let h be the height of the hierarchy and r be the number of children of each non-leaf node in the hierarchy. The result is:

Proposition 2 *For any single term query q, the number of estimation computations is no more than $(s+1)*r*(h-1)$. (There is no assumption required for this result.)*

We first establish a lemma before we prove this proposition. The following Lemma says that the estimation of the similarity of the most similar document in a database or a superdatabase is exact for any single term query and is given by the maximum normalized weight of the term of the query which is stored in the database representative.

Let S be a database or a superdatabase and $mnw(t\ in\ S)$ be the maximum normalized weight of term t in S.

Lemma 3 *For any single term query q containing term t with inverse document frequency weight idf and term frequency weight tf, let d be the most similar document to q in a database or superdatabase S. Then $sim(q,d) = mnw(t\ in\ S) = est(S)$.*

Proof: Let d appear in a database D which is either S or contained in the superdatabase S. In the computation of $est(D)$, the first part of the expression within braces in equation (1) disappears for a single term query. Since q has a single term, $|q| = tf * idf$. Thus, $est(D) = mnw(t\ in\ D)$. By direct computation, $sim(q,d) = \frac{q \cdot d}{|q| * |d|} = 1$ * normalized weight of t in d, as $q/|q|$ is a vector with one entry equals to 1 for term t and all other entries being zero and the "$\cdot$" represents the dot product between two vectors. Since d is the most similar document for the single term query having term t, $sim(q,d) = mnw(t\ in\ D)$. Thus, $sim(q,d) = mnw(t\ in\ D) = est(D)$.

Let P be the parent superdatabase containing database D. Since d is the most similar document in S and P is in S, d is the most similar document in P. For any database D' in P and the most similar document d' in D', $mnw(t\ in\ D') = sim(q,d') \leq mnw(t\ in\ D) = sim(q,d)$, since d is most similar. Thus, for the representative of P, $mnw(t\ in\ P) = mnw(t\ in\ D)$. This implies (by equation (2), but with the expression devided by $|q|$) $est(P) = mnw(t\ in\ P) = sim(q,d)$.

By repeating the same argument to the parent of P and its parent etc until the superdatabase S is reached, we then have $sim(q,d) = mnw(t\ in\ S) = est(S)$. ■

Proof of Proposition 2: Let ls be the similarity of the most similar document, d, in all the databases. By Lemma 3, $est(A) = ls$, for each non-root ancestor A of the database containing d. Let P be a superdatabase which is one level below the root but does not contain the document d.

Let the most similar document, d', in P has similarity s'. Since d is the most similar document in all databases, $ls > s'$. Thus, $est(A) > est(P)$. Consider the path $A_1, A_2, ..., A_{h-1}$ in which A_1 is the child of the root, A_{i+1} is the child of A_i and A_{h-1} contains d. When the search algorithm is applied, all children of the root undergo estimation computations. Since $est(A_1) > est(P)$ for each child P of the root, which is different from A_1, all children of A_1 undergo estimation computations. Again, since $est(A_2) = est(A_1) > est(P)$, all children of A_2 undergo estimation computations. This is repeated until the database A_{h-1} containing d is reached. When database A_{h-1} is searched, d is returned. Let this database also contain the second, the third, ..., the p-th most similar document, then the $(p+1)$-th most similar document, say d^*, is in a different database which is either in one of the superdatabases $A_1, A_2, ..., A_{h-2}$ or in a superdatabase P' which is a child of the root. In the former case, let A_i be the lowest ancestor of the database containing d^*. Let B_1 be the child of A_i and is an ancestor of the database containing d^*. Then $est(B_1) > est(P^*)$ for any child of the root P^* not containing d or d^*. This causes the path $B_1, B_2, ..., B_q$ to be traversed, where B_{i+1} is a child of B_i and B_q contains d^*, as $est(B_i) = est(B_{i+1})$. Eventually, the database B_q is searched, causing d^* to be returned. In the latter situation, the path from P' to the database containing d^* is traversed by the same argument given above. By repeating the same argument, up to $(s+1)$ databases will be searched where the first s databases contain the m most similar documents.

The reason why the $(s+1)$-th database may be searched is as follows. Suppose the s-th database T contains 2 or more of the m most similar documents. When T is searched using Algorithm $Search(m, q, Root)$, the most similar document in T and its similarity are returned to the result merger. Since there is at least one document in T which has not been returned, the number of documents received by the result merger is strictly less than m. This triggers the $(s+1)$-th database to be searched and its most similar document and its similarity s' to be returned. Since s' is less than the similarity of the most similar document in the s-th database, the documents in T with similarities greater than or equal to s' are returned, resulting in the retrieval of all of the m most similar documents.

Each of the $(s+1)$ databases is reached after going through at most $(h-1)$ of its ancestors. For each ancestor, r estimation computations are performed. This results in at most $(s+1)*(h-1)*r$ estimation computations. ■

In the situation that a query has multiple terms, the above result is true if the databases are properly clustered together in the sense that similar databases belong to the same logical superdatabases. For a database or a superdatabase S, let $mnw(t\ in\ S)$ denote the maximum normalized weight of term t in S. $aw(t\ in\ S)$ and $maw(t\ in\ S)$ can be defined similarly.

Definition 1 *A set of databases and superdatabases are suitably clustered in the hierarchy H with*

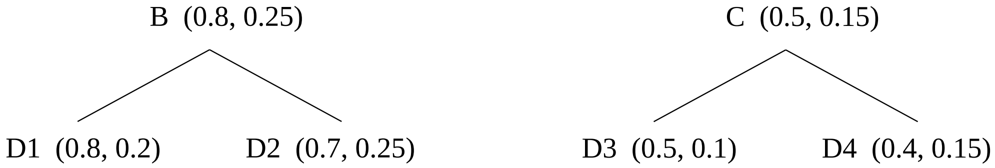

Figure 2: Illustrating Suitably Clustered Databases

Figure 3: Illustrating Not Suitably Clustered Databases

respect to a query q if for each term t of the query, the following condition holds. For any given m, where m is the number of most similar documents to the query to be retrieved, let D_i be a database containing at least one of the m most similar documents and C be a superdatabase that does not contain any of the m documents, then $mnw(t\ in\ D_i) > mnw(t\ in\ D)$ and $aw(t\ in\ D_i) \geq aw(t\ in\ D)$ for any database D which is contained in C.

We motivate this definition by the following example. There are two superdatabases in Figure 2. The statistics (a, b) of a term in two different databases and superdatabases are shown. For example, $D_1(0.8, 0.2)$ means the maximum normalized term weight and the average normalized term weight of the term in D_1 are respectively 0.8 and 0.2. Suppose in Figure 2, database D_1 contains all of the m most similar documents, but databases D_3 and D_4 do not contain any of them. The above definition says that if the databases are *suitably clustered* with respect to query q, then $mnw(t\ in\ D_1) > mnw(t\ in\ D_3)$, $aw(t\ in\ D_1) \geq aw(t\ in\ D_3)$, $mnw(t\ in\ D_1) > mnw(t\ in\ D_4)$ and $aw(t\ in\ D_1) \geq aw(t\ in\ D_4)$. The reason why these inequalities are reasonable is that databases with high values of the statistics of certain terms should be within a cluster and similarly databases with low values of the terms should also be within a cluster. In practice, this condition may not be true for all queries but are likely to hold for most queries if the databases are clustered properly. Figure 3 shows a violation of this definition as it is not true that $aw(D_1) \geq aw(D_2)$. Here, the high values of the statistics of the term in D_2 are mixed with the low values of the statistics of the term in D_4, which implies that the databases are not suitably clustered.

The statistics of a term t in a database representative R are the maximum normalized term weight and the average normalized term weight. For a super-representative, they are the maximum normalized weight and the maximum average normalized weight. In either case, they are denoted by $stat(t\ in\ R)$. We say the statistics of t in R_1 is larger than that of t in R_2, denoted by

$stat(t\ in\ R_1) > stat(t\ in\ R_2)$ if $mnw(t\ in\ R_1) \geq mnw(t\ in\ R_2)$ and $aw(t\ in\ R_1) \geq aw(t\ in\ R_2)$ (maximum average normalized weight of $t\ in\ R_2$ if R_2 is a super-representative) and at least one of the inequalities is strict.

A consequence that a set of databases are suitably clustered with respect to the query q is that the statistics of each term of q in a database representative R_1 containing at least one of the m most similar documents to q is larger than the statistics of the same term in any super-representative R_2 which does not contain any of the m most similar documents to q.

Lemma 4 *If a set of databases are suitably clustered with respect to a query q, then for any database representative D_1 containing at least one of the m most similar documents and any super-representative C which does not contain any of the m documents, $stat(t\ in\ D_1) > stat(t\ in\ C)$ for any term t in the query q.*

Proof: Let D_s be any database in C. By Definition 1, $mnw(t\ in\ D_1) > mnw(t\ in\ D_s)$ and $aw(t\ in\ D_1) \geq aw(t\ in\ D_s)$. Thus, $stat(t\ in\ D_1) > stat(t\ in\ D_s)$. Consider the representative, R, of the parent, P, of D_s. Recall that the two quantities of each term in R are obtained by taking the maximum of the corresponding quantities of the term in the databases contained in P. Thus, $stat(t\ in\ D_1) > stat(t\ in\ P)$. By repeating this process to the ancestors of P until C is reached, we obtain $stat(t\ in\ D_1) > stat(t\ in\ C)$ for every term t in the query q. ■

Proposition 3 *If the databases are suitably clustered with respect to the query q, then the number of similarity computations is bounded by $(s+1)*(h-1)*r$, where s is the number of databases containing the m most similar documents to q, h is the height of the hierarchy and r is the number of children per non-leaf node.*

Proof: Let t be the smallest positive integer such that the $(m+t)$-th most similarity document is in a database different from the s databases containing the m most similar documents. Let $D_1, ..., D_{s+1}$ denote these $(s+1)$ databases containing the $(m+t)$ most similar documents. We claim that for each database D containing one of the $(m+t)$ most similar documents and each superdatabase S which does not contain any of the $(m+t)$ most similar documents, $est(D) > est(S)$. By applying Lemma 4 to the $(m+t)$ most similar documents, we obtain $stat(D) > stat(S)$. This implies $est(D) > est(S)$. Let $g = \min\{est(D)\}$, where the minimum is over the $(s+1)$ databases. There are at most $(s+1)$ non-leaf nodes at each level below the root node which are ancestors of the $(s+1)$ databases $D_1, ..., D_{s+1}$. Let P be a non-root ancestor containing one of the $(s+1)$ databases. Then, $est(P) \geq g > est(S)$. This means that each such P will be expanded before any S is expanded. When all such P's are expanded, each database D with $est(D) \geq g$ will be searched. When the database D with $est(D) = g$ is searched, the $(m+t)$ most similar documents will be retrieved and the algorithm terminates. Thus, no S will be expanded. Since there are at most $(s+1)$ such P at each level, the total number of P's expanded at all levels is at most $(s+1)*(h-1)$. Each expansion takes r estimation computations. Thus, the total number of estimation computations is $(s+1)*(h-1)*r$. ■

3.5 Placement of Non-Leaf Nodes of the Hierarchy

It was assumed in the last subsection that the hierarchy of representatives are placed completely in the site containing the metasearch engine. This is clearly a reasonable choice. However, there is a potential problem with this approach. The amount of storage required to store the representatives can be rather large. The problem and its potential remedy can be illustrated by the following example.

Suppose the size of a database representative is 2% of the size of the local database. In the case of a non-root superdatabase S, let S be composed from a set of local databases or superdatabases which are one level below S in the hierarchy. Assume that the size of the representative of S is on the average twice the size of one of its component representatives. Recall that there are two quantities for each term in a database representative. Thus, the above assumption is based on the premise that the component databases or superdatabases of S have a lot of terms in common and therefore the number of distinct terms in S is only twice of the number of distinct terms in a component database or superdatabase. We now use the previous example involving 900 databases to illustrate the storage requirement. Let each database have the same size s. Then the total size needed to store the 900 database representatives is $900*2\%*s = 18*s$. The representatives of the 30 superdatabases take $30*4\%*s = 1.2*s$. Thus, the storage requirement is $19.2*s$. In general, if there are n local databases and each superdatabase is made up of r databases or superdatabases which are one level below it, with r reasonably large, say $r \geq 10$, then the amount of storage for all representatives and super-representatives are approximately $n*2\%*s$ and $t*4\%*s/r$, respectively, as the number of non-leaf nodes which are two or more levels above the leaf nodes are much less than the number of nodes at the bottom two levels. When t is large, say in the order of thousands, it will impose a rather heavy burden on the site S_1 containing the metasearch engine, in terms of storage and processing power, if all these database representatives are placed in that site. If S_1 is powerful enough, then searching the hierarchy of database representatives can be carried out as described; otherwise, we propose that all super-representatives be placed at the site S_1 and each database representative be placed at the site where the database resides. Thus, the storage requirement at S_1 is approximately $1.2*s$ for our example.

We now describe the process to execute a query q when local representatives are stored at their local database site. The query is submitted to the site of the metasearch engine. The global document frequency of each term in q is kept in the root representative, as before. These document frequencies together with the query q are used to estimate the similarity of the most similar document in each superdatabase and executes the best-first algorithm until the parent P of some local database representatives is reached. At that point, query q as well as the global document frequencies of the terms in q will be sent to each child (site) of P. Each of the these child sites then estimates the similarity of the most similar document in its own database and forwards the estimate to the global site S_1, which will continue to execute the best-first search algorithm $Search(m, q, Root)$. In other words, for a given query, the algorithm is executed distributively in multiple sites consisting of the sites S_1 and some of the sites containing the local databases. The trade-off between this way of storing the database representatives versus the traditional way in which all database representatives are stored in one site is that the former way will incur higher communication cost but less processing cost, since the estimations in local sites can be executed in parallel. Clearly, the former method imposes much less storage burden on the site S_1, since the representatives of the local databases (which take much more storage than the representatives of the superdatabases) are kept in their own sites.

4 Conclusion

We have shown that for a distributed environment involving reasonable number of databases a two-level hierarchy of database representatives together with an appropriate estimation algorithm can yield nearly the same effectiveness as if all documents were in one centralized site. When the number of databases is very large, we propose a hierarchy of representatives with the number of levels > 2. We provide an algorithm to search the hierarchy. It is shown that the search will produce the same effectiveness as the corresponding two-level hierarchy. We also show that the search of the hierarchy is efficient for single term queries which are submitted frequently in the Internet environment. Under the assumption that databases are clustered properly, we show that the search for multi-term queries is also efficient.

We plan to implement the hierarchy search process for number of levels > 2 in the future and perform experiments with it. We will also incorporate information (such as co-occurrence of terms and linkages between web pages) that affects relevance of documents into the solutions to the database selection problem and the collection fusion problem.

Acknowledgement: We are grateful to L. Gravano and H. Garcia-Molina for providing us with the collection of documents and queries used in our experiments.

References

[1] G. Abdulla, B. Liu, R. Saad, and E. Fox. *Characterizing World Wide Web Queries.* TR-97-04, Virginia Polytechnic Institute and State University, 1997.

[2] C. Baumgarten. *A Probabilistic Model for Distributed Information Retrieval.* ACM SIGIR Conference, Philadelphia, 1997.

[3] J. Callan, Z. Lu, and W. Bruce Croft. *Searching Distributed Collections with Inference Networks.* ACM SIGIR Conference, Seattle, 1995.

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

[5] Y. Fan, and S. Gauch. *Adaptive Agents for Information Gathering from Multiple, Distributed Information Sources.* 1999 AAAI Symposium on Intelligent Agents in Cyberspace, Stanford University, March 1999.

[6] S. Gauch, G. Wang, and M. Gomez. *ProFusion: Intelligent Fusion from Multiple, Distributed Search Engines.* Journal of Universal Computer Science, 2(9), pp. 637-649, 1996.

[7] L. Gravano, and H. Garcia-Molina. *Generalizing GlOSS to Vector-Space databases and Broker Hierarchies.* International Conferences on Very Large Data Bases, 1995.

[8] L. Gravano, and H. Garcia-Molina. *Merging Ranks from Heterogeneous Internet sources.* International Conferences on Very Large Data Bases, 1997.

[9] D. Harman. *Overview of the First Text Retrieval Conference.* Edited by D. Harman, Computer Systems Technology, U.S. Department of Commerce, NIST, 1993.

[10] A. Howe, and D. Dreilinger. *SavvySearch: A Meta-Search Engine that Learns Which Search Engines to Query.* AI Magazine, 18(2), 1997.

[11] B. Jansen, A. Spink, J. Bateman, and T. Saracevic. *Real Life Information Retrieval: A Study of User Queries on the Web.* ACM SIGIR Forum, 32:1, 1998.

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

[13] T. Kirk, A. Levy, Y. Sagiv, and D. Srivastava. *The Information Manifold.* AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments. 1995.

[14] M. Koster. *ALIWEB: Archie-Like Indexing in the Web.* Computer Networks and ISDN Systems, 27:2, 1994, pp. 175-182 (http://www.cs.indiana.edu/aliweb/form.html).

[15] G. Kowalski. *Information Retrieval Systems, Theory and Implementation.* Kluwer Academic Publishers, 1997.

[16] K. Kwok and M. Chan. *Improving Two-Stage Ad-Hoc Retrieval for Short Queries*, ACM SIGIR 1998.

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

[18] W. Meng, K-L. Liu, C. Yu, X. Wang, Y. Chang, and N. Rishe. *Determining Text Databases to Search in the Internet.* International Conferences on Very Large Data Bases, New York City, 1998.

[19] 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.

[20] W. Meng, C. Yu, and K. Liu. *Challenges and Solutions for Building an Efficient and Effective Metasearch Engine.* Technical Report, Dept. of CS, SUNY at Binghamton, 1999.

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

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

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

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

[25] G. Towell, E. Voorhees, N. Gupta, and B. Johnson-Laird. *Learning Collection Fusion Strategies for Information Retrieval.* 12th Int'l Conf. on Machine Learning, 1995.

[26] E. Voorhees, N. Gupta, and B. Johnson-Laird. *Learning Collection Fusion Strategies.* ACM SIGIR Conference, Seattle, 1995.

[27] J. Xu, and J. Callan. *Effective Retrieval with Distributed Collections.* ACM SIGIR Conference, 1998.

[28] T. W. Yan, and H. Garcia-Molina. *SIFT – A Tool for Wide-Area Information Dissemination.* USENIX 1995 Technical Conference, 1995.

[29] Y. Yang, and C. Chute. *An Example-based Mapping Method for Text Categorization and Retrieval.* ACM Transactions on Information Systems, 1994, pp. 252-277.

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

[31] C. Yu, K. Liu, W. Wu, W. Meng and N. Rishe. *A Methodology to Retrieve Text Documents from Multiple Databases.* Tech. report, U. of Illinois at Chicago, Chicago, Il. 60607-7053, 1999.

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

[33] C. Yu, W. Meng, K. Liu, W. Wu, and N. Rishe. *Efficient and Effective Metasearch for a Large Number of Text Databases.* Tech. report, U. of Illinois at Chicago, Chicago, Il. 60607-7053, 1999. (http://panda.cs.binghamton.edu/ ~meng/cikm99.ps)

[34] B. Yuwono, and D. Lee. *Server Ranking for Distributed Text Resource Systems on the Internet.* 5th International Conference On Database Systems For Advanced Applications (DASFAA'97), Melbourne, Australia, April 1997, pp. 391-400.