Generalizing *GlOSS* to Vector-Space Databases and Broker Hierarchies *

Luis Gravano Héctor García-Molina

Computer Science Department
Stanford University
Stanford, CA 94305-2140
{gravano,hector}@cs.stanford.edu

Abstract

As large numbers of text databases have become available on the Internet, it is harder to locate the right sources for given queries. In this paper we present *gGlOSS*, a generalized *Glossary-Of-Servers Server*, that keeps statistics on the available databases to estimate which databases are the potentially most useful for a given query. *gGlOSS* extends our previous work [1], which focused on databases using the boolean model of document retrieval, to cover databases using the more sophisticated *vector-space* retrieval model. We evaluate our new techniques using real-user queries and 53 databases. Finally, we further generalize our approach by showing how to build a hierarchy of *gGlOSS* brokers. The top level of the hierarchy is so small it could be widely replicated, even at end-user workstations.

*This research was sponsored by the Advanced Research Projects Agency (ARPA) of the Department of Defense under Grant No. MDA972-92-J-1029 with the Corporation for National Research Initiatives (CNRI). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of ARPA, the U.S. Government or CNRI. This work was supported by an equipment grant from IBM Corporation.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 21st VLDB Conference
Zürich, Switzerland, 1995

1 Introduction

The dramatic growth of the Internet over the past few years has created a new problem: finding the right text databases to evaluate a given query. There are thousands of sources available to the users on the Internet, and it is practically impossible to query all of them when searching for information on a given topic: not only would such an exhaustive search take a long time to complete, but it could also be expensive, since some of the text databases on the Internet may charge for their use. Consequently, users need a way to narrow their searches to a few useful text databases. This problem is a specific instance of the more general resource-discovery problem [2, 3].

Many tools have recently appeared on the Internet to help users select the (text) databases that might be more useful for their queries (see Section 2). However, many of these tools essentially keep a global index of the available documents. This approach clearly does not scale well with the growing number of sources and documents. Alternatively, many other tools index only a small part of each available document (e.g., its title). This approach fails to identify many useful sources because a significant part of each document is simply discarded. Similarly, other tools just keep succinct summaries of the contents of each database. These summaries are sometimes manually written, are often out of date, and fail to capture the whole content of the databases.

Our approach is to provide a broker that ranks the *potentially* most useful databases for a given query. This broker keeps only *partial information* on the contents of each database, so it scales with the growing number of available databases. However, this information covers the *full-text* content of the documents, so that the useful sources are identified. In [1] and [4] we described *GlOSS* (for *Glossary-Of-Servers Server*) [1], a centralized broker that keeps meta-information about databases supporting the *boolean* model of document retrieval. *GlOSS* maintains statistics on the databases, and uses these statistics to estimate the actual contents of the databases. When users query *GlOSS*, it uses these statistics to rank the databases according to their estimated usefulness for the given query. The users then access the databases themselves, following the order that *GlOSS* suggested.

Although the boolean model of document retrieval is widely used, it is a rather primitive one. One of the most popular alternative models is the *vector-space retrieval model* [5, 6]. This model represents both the documents in a database and the queries themselves as weight vectors. Given a query, the documents are ranked according to how "similar" their corresponding vectors are to the given query vector.

In this paper we present *gGlOSS*, a generalized and more powerful version of *GlOSS* that also deals well with vector-space databases and queries. Like *GlOSS*, *gGlOSS* periodically collects statistics on the underlying sources (this time including summary word-weight information). We first determine the *goodness* of a database for a query, and the *ideal database rank* for a query (i.e., the rank that *gGlOSS* should try to produce for the query). Then, given a query and a desired goodness metric, *gGlOSS* can rank the available sources.

Since *gGlOSS* produces estimates of the ideal database ranks, we need to compare these estimates against the ideal ranks. For this, we evaluate the performance of *gGlOSS* using real-user queries and 53 vector-space databases, in terms of how close the *gGlOSS* ranks are to the ideal ones. Although we can estimate the size of the *gGlOSS* information to be only around 2% of the size of a full index of the databases, its performance is good (Section 6), showing that *gGlOSS* can closely approximate the ideal database ranks for the given queries.

We also present facilities for building hierarchies of *gGlOSS* servers. In this case, *hGlOSS*, a high-level server, summarizes the contents of lower-level *gGlOSS* brokers, in much the same way as the *gGlOSS* brokers summarize the contents of the underlying databases. Given a query, the *hGlOSS* server suggests *gGlOSS* servers that might index useful databases for the query. Because the storage requirements of the *hGlOSS* server are much smaller than those of the *gGlOSS* brokers, we can easily replicate the *hGlOSS* server so that it does not become a performance bottleneck, thus distributing the load for searching the system.

In what follows, Section 3 defines one "ideal" database rank for a query. Section 4 shows how *gGlOSS* approximates the ideal database rank using partial information. Section 5 introduces the methodology for the experimental results of Section 6. Section 7 discusses alternative definitions of the ideal database rank. Finally, in Section 8 we show how to build the higher-level *hGlOSS* servers.

2 Related Work

One approach to solving the text-database discovery problem (see [2, 3] for surveys) is to let users "browse" the databases. Well-known examples include the Prospero file system [7], Gopher [2], and the World-Wide Web [8].

A different approach is to let users query a database of "meta-information" about the available databases. For example, WAIS [9] provides a "directory of servers." Many search facilities have been created for the World-Wide Web, like the Lycos service. [2] To scale with the growing number of available databases, some of these systems index only document titles or, more generally, just a small fraction of each document (e.g., the World-Wide Web Worm [3]). Other systems keep succinct, sometimes human-generated, summaries of the contents of each database (e.g., the ALIWEB system [4]).

Recently, [10] has applied inference networks (from information retrieval) to the text-database discovery problem. Their approach summarizes databases using document-frequency information for each term (the same type of information that *GlOSS* keeps about the databases), together with the "inverse collection frequency" of the different terms. An inference network then uses this information to rank the databases for a given query. The "content-based routing" system of [11, 12] keeps a "content label" for each collection of objects, with attributes describing the contents of the collection.

The Harvest system [13] provides a flexible architecture for accessing information on the Internet. "Gatherers" collect information about the data sources, and pass it to "brokers." The "Harvest Server Registry" is a special broker that keeps information about all other brokers, among other things. For flexibility, Harvest leaves the broker specification open, and many alternative designs are possible.

[1] We have implemented *GlOSS* and made it accessible at `http://gloss.stanford.edu`.

[2] Lycos is accessible at `http://lycos.cs.cmu.edu`.

[3] The World-Wide Web Worm is accessible at `http://www.cs.colorado.edu/home/mcbryan/WWWW.html`.

[4] ALIWEB is accessible at `http://web.nexor.co.uk/aliweb/doc/aliweb.html`.

3 Ranking Databases

Given a query, we would like to rank the available vector-space databases according to their usefulness. This ranking should capture the ideal order for searching the databases: we should first search the most useful database(s), then the second most useful database(s), and so on, until we either exhaust the rank, or become satisfied with whatever documents we got up to that point. This section presents one definition for the *ideal database rank.* The next section explores how *gGlOSS* will try to rank the databases as closely as possible to this ideal rank.

Determining the ideal database rank for a query is a hard problem. The definition of this section is based solely on the answers (i.e., the document ranks and their scores) that each database produces when presented with the query in question. This definition does not use the relevance [5] of the documents to the end user who submitted the query. Using relevance would be appropriate for evaluating the search engines at each database; instead, we are evaluating how well *gGlOSS* can predict the answers that the databases return. In Section 7 we discuss our choice further, and analyze some of the possible alternatives that we could have used.

To define the ideal database rank for a query q, we need to determine how good each database db is for q. In this paper we assume that all databases use the same algorithms to compute weights and similarities. We consider that the only documents in db that are useful for q are those with a similarity to q greater than a given threshold l, as determined by db. Documents with lower similarity are unlikely to be useful, and therefore we ignore them. Thus, we define:

$$Goodness(l, q, db) = \sum_{d \in Rank(l, q, db)} sim(q, d) \qquad (1)$$

where $sim(q, d)$ is the similarity between query q and document d, and $Rank(l, q, db) = \{d \in db | sim(q, d) > l\}$. The ideal rank of databases $Ideal(l)$ is then determined by sorting the databases according to their goodness for the query q.

Example 3.1 *Consider two databases, db_1 and db_2, a query q, and the answers that the two databases give when presented with query q:*

$$\begin{aligned} db_1 &: (d_1^1, 0.9), (d_2^1, 0.9), (d_3^1, 0.1) \\ db_2 &: (d_1^2, 0.8), (d_2^2, 0.4), (d_3^2, 0.3), (d_4^2, 0.1) \end{aligned}$$

In the example, db_1 returns documents d_1^1, d_2^1, and d_3^1 as its answer to q. Documents d_1^1 and d_2^1 are ranked the highest in the answer, because they are the "closest" to query q in database db_1 (similarity 0.9). To determine how good each of these databases is for q, we use Equation 1. If the threshold l is 0.2 (i.e., the user is willing to examine every document with similarity to q higher than 0.2), the goodness of db_1 is $Goodness(0.2, q, db_1) = 0.9+0.9 = 1.8$, because db_1 has two documents, d_1^1 and d_2^1, with similarity higher than 0.2. Similarly, $Goodness(0.2, q, db_2) = 0.8+0.4+0.3 = 1.5$. Therefore, $Ideal(0.2)$ is db_1, db_2.

The goodness of a database tries to quantify how useful the database is for the user that issued the query. It does so by examining the document-query similarities as computed by each local source. A problem with this definition is that these similarities can depend on the characteristics of the collection that contains the document. Therefore, these similarities are not "globally valid." For example, if a database db_1 specializes in computer science, the word *databases* might appear in many of its documents. Then, this word will tend to have a low associated weight in db_1 (e.g., if db_1 uses the *tf·idf* formula for computing weights [6]). The word *databases*, on the other hand, might have a high associated weight in a database db_2 that is totally unrelated to computer science and contains very few document with that word. Consequently, db_1 might assign its documents a low score for a query containing the word *databases*, while db_2 assigns a few documents a high score for that query. The *Goodness* definition of Equation 1 might then determine that db_2 is better than db_1, while db_1 is the best database for the query. In Section 7 we further discuss this problem, together with alternative ways of defining *Goodness*.

4 Choosing Databases

gGlOSS helps users determine what databases might be most helpful for a query. Users first query *gGlOSS* to obtain a rank of the databases according to their potential usefulness. To perform this task, *gGlOSS* keeps information on the available databases, to estimate their goodness for the query. One option would be for *gGlOSS* to keep complete information on each database: for each database db and word t, *gGlOSS* would know what documents in db contain t, what weight t has in each of them, and so on. Although *gGlOSS*'s answers would always be accurate (if this information is kept up to date), the storage requirements of such an approach would be too high: *gGlOSS* needs to index many databases, and keeping so much information on each of them does not scale.

More reasonable solutions keep incomplete yet useful information on the databases. In this paper we explore some options for *gGlOSS* that require one or both of the following matrices:

- $F = (f_{ij})$: f_{ij} is the number of documents in database db_i that contain word t_j
- $W = (w_{ij})$: w_{ij} is the sum of the weight of word t_j over all documents in database db_i

In other words, for each word t_j and each vector-space database db_i, *gGlOSS* needs (at most) two numbers. The second of these numbers is the sum of the weight of t_j over all documents in db_i, as determined by the vector-space retrieval algorithm that db_i uses. Typically, the weight of a word t_j in a document d is a function of the number of times that t_j appears in d and the number of documents in the database that contain t_j [6]. Although the information that *gGlOSS* stores about each database is incomplete, it will prove useful to generate database ranks that resemble the ideal database rank of Section 3, as we will see in Section 6.2. Furthermore, this information is orders of magnitude smaller than that required by a full-text index of the databases, for example. Adapting the boolean-database estimates of [1], we can estimate that the size of the *gGlOSS* information about a vector-space database is only around 2% of the size of a full-text vector-space index of the database.

To obtain the data that *gGlOSS* keeps about a database db_i, namely rows f_{i*} and w_{i*} of the F and W matrices above, database db_i will have to periodically run a *collector* program that extracts this information from the local indexes and sends it to the *gGlOSS* server.

Example 4.1 *Consider a database db and the word* computer. *Suppose that the following are the documents in db having the word* computer *in them, together with the associated weights:*

computer : $(d_1, 0.8), (d_2, 0.7), (d_3, 0.9), (d_8, 0.9)$

That is, document d_1 contains the word computer *with weight 0.8 (for some weight-computation algorithm [5]), document d_2, with weight 0.7, and so on. The gGlOSS collector will not send gGlOSS all this information: it will only tell gGlOSS that the word* computer *appears in four documents in database db, and that the sum of the weights with which the word appears in the documents is* $0.8 + 0.7 + 0.9 + 0.9 = 3.3$.

In our definitions below, we assume that a query q is expressed as a weight vector $Q = (q_1, \ldots, q_j, \ldots, q_t)$ [5], where q_j is the weight of word t_j in query q. For example, this weight can simply be the number of times that word t_j appears in the query. We also assume throughout this paper that the vector-space databases compute the similarity between a document and a query by taking the inner product of the corresponding document and query weight vectors.

Since *gGlOSS* represents both the databases and the queries as vectors, *gGlOSS* could compute similarities between these vectors analogously to how *documents* and queries are compared. *gGlOSS* could use these similarities to rank the databases for the given query. For example, *gGlOSS* could estimate the goodness of database db_i for query q as the inner product $w'_{i*} \cdot Q$, where $w'_{i*} = (w'_{i1}, \ldots, w'_{it})$ is the (normalized) row of W that corresponds to db_i. However, we are interested in finding the databases that contain useful documents for the queries, not those databases that are "similar" to the given queries. The definitions of the *gGlOSS* ranks below reflect this fact. Also, note that the vectors with which *gGlOSS* represents each database can be viewed as *cluster centroids* [6], where each database is considered as a single document cluster [5].

Because the information that *gGlOSS* keeps about each database is incomplete, it has to make assumptions regarding the distribution of query keywords and weights across the documents of each database. These assumptions allow *gGlOSS* to compute better estimates. The following sections present two sets of assumptions that *gGlOSS* will use to derive different database ranks for a given query. These assumptions are artificial: very rarely would a set of databases and queries conform to them. However, we use them because these type of assumptions proved themselves useful in the boolean-*GlOSS* case for choosing the "right" databases for a query [1, 4].

4.1 High-Correlation Scenario

To derive $Max(l)$, the first database rank with which *gGlOSS* tries to match the $Ideal(l)$ database rank of Section 3, *gGlOSS* assumes that if two words appear together in a user query, then these words will appear in the database documents with the highest possible correlation:

Assumption 4.1 *If query keywords t_1 and t_2 appear in f_{i1} and f_{i2} documents in database db_i, respectively, and $f_{i1} \leq f_{i2}$, then every db_i document that contains t_1 also contains t_2.*

Example 4.2 *Consider a database db_i and the query q=*computer science department. *For simplicity, let t_1=* computer, *t_2=* science, *and t_3=* department. *Suppose that $f_{i1} = 2$, $f_{i2} = 9$, and $f_{i3} = 10$: there are 2 documents in db_i with the word* computer, *9 with the word* science, *and 10 with the word* department.

gGlOSS assumes that the 2 documents with the word computer *also contain the words* science *and* department. *Furthermore, all of the $9 - 2 = 7$ documents*

[5] An interesting direction to explore is to represent each database db as a set of (very few) cluster centroids. Each of these centroids would summarize a set of closely related documents of db.

with word science *but not with word* computer *also contain the word* department. *Finally, there is exactly* $10 - 9 = 1$ *document with just the word* department.

gGlOSS also needs to make assumptions on the weight distribution of the words across the documents of a database:

Assumption 4.2 *The weight of a word is distributed uniformly over all documents that contain the word.*

Thus, word t_j has weight $\frac{w_{ij}}{f_{ij}}$ in every db_i document that contains t_j. This assumption simplifies the computations that *gGlOSS* has to make to rank the databases. We will see in Section 6 that this unrealistic assumption is surprisingly effective.

Example 4.2 (cont.) *Suppose that the total weights for the query words in database db_i are $w_{i1} = 0.45$, $w_{i2} = 0.2$, and $w_{i3} = 0.9$. According to Assumption 4.2, each of the two documents that contain word* computer *will do so with weight $\frac{0.45}{2} = 0.225$, each of the 9 documents that contain word* science *will do so with weight $\frac{0.2}{9} = 0.022$, and so on.*

gGlOSS uses the assumptions above to estimate how many documents in a database have similarity greater than some *threshold l* to a given query, and what the added similarity of these documents is. These estimates determine the $Max(l)$ database rank.

Consider database db_i with its two associated vectors f_{i*} and w_{i*}, and query q, with its associated vector Q. Suppose that the words in q are $t_1, \ldots, t_n$, with $f_{ia} \leq f_{ib}$ for all $1 \leq a \leq b \leq n$. Assume that $f_{i1} > 0$. From Assumption 4.1, the f_{i1} documents in db_i that contain word t_1 also contain all of the other $n-1$ query words. From Assumption 4.2, the similarity of any of these f_{i1} documents to the query q is:

$$sim_1 = \sum_{j=1,\ldots,n} q_j \times \frac{w_{ij}}{f_{ij}}$$

Furthermore, these f_{i1} documents have the highest similarity to q among the documents in db_i. Therefore, if $sim_1 \leq l$, then there are no documents in db_i with similarity greater than threshold l. If, on the other hand, $sim_1 > l$, then *gGlOSS* should explore the $f_{i2} - f_{i1}$ documents (Assumption 4.1) that contain words $t_2, \ldots, t_n$, but not word t_1. Thus, *gGlOSS* finds p such that:

$$sim_p = \sum_{j=p,\ldots,n} q_j \times \frac{w_{ij}}{f_{ij}} \quad > \quad l, \text{ but} \qquad (2)$$

$$sim_{p+1} = \sum_{j=p+1,\ldots,n} q_j \times \frac{w_{ij}}{f_{ij}} \quad \leq \quad l \qquad (3)$$

Then, the f_{ip} documents having (at least) query words $t_p, \ldots, t_n$ have an estimated similarity to q greater than threshold l (Condition 2), whereas the documents having only query words $t_{p+1}, \ldots, t_n$ do not.

Using this definition of p and the assumptions above, we give the first definition for $Estimate(l, q, db_i)$, the estimated goodness of database db_i for query q, that determines the $Max(l)$ database rank:

$$\begin{aligned} & Estimate(l, q, db_i) = \\ & = \sum_{j=1,\ldots,p} (f_{ij} - f_{i(j-1)}) \times sim_j \\ & = (\sum_{j=1,\ldots,p} q_j \times w_{ij}) + f_{ip} \times \sum_{j=p+1,\ldots,n} q_j \times \frac{w_{ij}}{f_{ij}} \qquad (4) \end{aligned}$$

where we define $f_{i0} = 0$, and sim_j is the similarity between q and any document having words $t_j, \ldots, t_n$, but not words $t_1, \ldots, t_{j-1}$. There are $f_{ij} - f_{i(j-1)}$ such documents in db_i. This definition computes the added similarity of the f_{ip} documents estimated to have similarity to q greater than threshold l. (See Conditions 2 and 3, and Assumptions 4.1 and 4.2.)

Example 4.2 (cont.) *Assume that query q has weight 1 for each of its three words. According to Assumption 4.1, the two documents with the word* computer *also have the words* science *and* department *in them. The similarity of any of these two documents to q is, using Assumption 4.2, $\frac{0.45}{2} + \frac{0.2}{9} + \frac{0.9}{10} = 0.337$. If our threshold l is 0.2, then all of these documents are acceptable, because their similarity to q is higher than 0.2. Also, there are $9 - 2 = 7$ documents with the words* science *and* department *but not* computer. *The similarity of any of these 7 documents to q is $\frac{0.2}{9} + \frac{0.9}{10} = 0.112$. Then these documents are not acceptable for threshold $l = 0.2$. There is $10 - 9 = 1$ document with only the word* department, *but this document's similarity to q is even lower. Consequently, $p = 1$. (See Conditions 2 and 3.) Then, according to the $Max(0.2)$ definition of Estimate, $Estimate(0.2, q, db_i) = f_{i1} \times (q_1 \times \frac{w_{i1}}{f_{i1}} + q_2 \times \frac{w_{i2}}{f_{i2}} + q_3 \times \frac{w_{i3}}{f_{i3}}) = 2 \times (1 \times \frac{0.45}{2} + 1 \times \frac{0.2}{9} + 1 \times \frac{0.9}{10}) = 0.674$.*

4.2 Disjoint Scenario

To derive $Sum(l)$, another rank that *gGlOSS* uses to approximate $Ideal(l)$, *gGlOSS* assumes that if two words appear together in a user query, then these words do not appear together in any database document (if possible):

Assumption 4.3 *The set of db_i documents with word t_1 is disjoint with the set of db_i documents with word t_2, for all t_1 and t_2, $t_1 \neq t_2$, that appear in query q.*

Therefore, the words that appear in a user query are assumed to be negatively correlated in the database documents. *gGlOSS* also needs to make Assumption 4.2, that is, the assumption that weights are uniformly distributed.

Consider database db_i with its two associated vectors f_{i*} and w_{i*}, and query q, with its associated vector Q. Suppose that the words in q are $t_1, \ldots, t_n$. For any query word t_j ($1 \leq j \leq n$), then the f_{ij} documents containing t_j do not contain query word t_p, for all $1 \leq p \leq n$, $p \neq j$ (Assumption 4.3). Furthermore, the similarity of each of these f_{ij} documents to q is exactly $q_j \times \frac{w_{ij}}{f_{ij}}$, if $f_{ij} > 0$ (from Assumption 4.2).

For rank $Sum(l)$ we then define $Estimate(l, q, db_i)$, the estimated goodness of database db_i for query q, as:

$$
\begin{aligned}
&Estimate(l, q, db_i) = \\
&= \sum_{j=1,\ldots,n|(f_{ij}>0)\wedge(q_j \times \frac{w_{ij}}{f_{ij}} > l)} f_{ij} \times (q_j \times \frac{w_{ij}}{f_{ij}}) \\
&= \sum_{j=1,\ldots,n|(f_{ij}>0)\wedge(q_j \times \frac{w_{ij}}{f_{ij}} > l)} q_j \times w_{ij} \qquad (5)
\end{aligned}
$$

Example 4.3 *Consider the data of Example 4.2. According to Assumption 4.3, there are 2 documents containing the word* computer *and none of the other query words, 9 documents containing the word* science *and none of the other query words, and 10 documents containing the word* department *and none of the other query words. The documents in the first group have similarity $\frac{0.45}{2} = 0.225$ (from Assumption 4.2), and are thus acceptable, because our threshold l is 0.2. The documents in the second and third groups have similarity $\frac{0.2}{9} = 0.022$ and $\frac{0.9}{10} = 0.09$, respectively, and are thus not acceptable for our threshold. So, the only documents close enough to query q are the two documents that contain word* computer. *Then, according to the $Sum(0.2)$ definition of Estimate, $Estimate(0.2, q, db_i) = f_{i1} \times \frac{w_{i1}}{f_{i1}} = 0.45$.*

Notice the special case when the threshold l is zero. In this case, the $Max(0)$ and $Sum(0)$ definitions of *Estimate* (Equations 4 and 5) become:

$$
Estimate(0, q, db_i) = \sum_{j=1,\ldots,n} q_j \times w_{ij}
$$

assuming that if $f_{ij} = 0$, then $w_{ij} = 0$. Then, $Estimate(0, q, db_i)$ becomes the inner product $Q \cdot w_{i*}$. To compute the $Max(0)$ and $Sum(0)$ ranks, *gGlOSS* does not need the matrix F of document frequencies of the words; it only needs the matrix W of added weights. [6] Therefore, the storage requirements for *gGlOSS* to compute the database ranks may be much lower if $l = 0$. We pay special attention to these ranks in our experiments of Section 6.2.

[6] We might need F, though, to compute the weight vector for the queries, depending on the algorithm used for this.

5 Comparing Database Ranks

In this section we analyze how we can compare *gGlOSS*'s ranks (Section 4) to the ideal one (Section 3) [7]. In the following section we report experimental results using the comparison methodology of this section.

Let q be a query, and $DB = \{db_1, \ldots, db_s\}$ be the set of available databases. Let $G = (db_{g_1}, \ldots, db_{g_{s'}})$ be the database rank that *gGlOSS* generated for q, using one of the schemes of Section 4. We only include in G those databases with estimated goodness greater than zero: we assume that users ignore databases with zero estimated goodness. Thus, in general, $s' \leq s$. Finally, let $I = (db_{i_1}, \ldots, db_{i_{s''}})$ be the ideal database rank. We only include in I those databases with actual goodness greater than zero. Our goal is to compare G against I, and quantify how close the two ranks are.

One way to compare the G and I ranks is by using the *Goodness* metric that we used to build I. We consider the top n databases in rank I, and compute i_n, the accumulated goodness of these n databases for query q. Because rank I was generated using this metric, the top n databases in rank I have the maximum accumulated goodness for q that any subset of n databases of DB can have. We then consider the top n databases in rank G, and compute g_n, the accumulated goodness of these n databases for q. Because *gGlOSS* generated rank G using only partial information about the databases, in general $g_n \leq i_n$. (If $n > s'$ (resp. $n > s''$), we compute g_n (i_n) by just taking the s' (s'') databases in G (I).) We then compute:

$$
\mathcal{R}_n = \begin{cases} \frac{g_n}{i_n} & \text{if } i_n > 0 \\ 1 & \text{otherwise} \end{cases}
$$

This number gives us the fraction of the optimum goodness (i_n) that *gGlOSS* captured in the top n databases in G, and models what the user that searches the top n databases that *gGlOSS* suggests would get, compared to what the user would have gotten by searching the top n databases in the ideal rank.

Example 5.1 *Consider a query q, and five databases db_i, $1 \leq i \leq 5$. Table 1 shows I, the ideal database rank, and G and H, two different gGlOSS database ranks for q, for some definition of these ranks. For example, db_1 is the top database in the ideal rank, with $Goodness(l, q, db_1) = 0.9$. Database db_5 does not appear in rank I, because $Goodness(l, q, db_5) =$*

[7] Our definition of the $\mathcal{R}_n$ metric in this section is partially based on the normalized cumulative recall metric of [14].

I		G		H	
db	*Goodness*	db	*Estimate*	db	*Estimate*
db_1	0.9	db_2	0.8	db_2	0.9
db_2	0.4	db_1	0.6	db_1	0.8
db_3	0.3	db_3	0.3	db_3	0.4
db_4	0.2			db_5	0.2

Table 1: The ideal and *gGlOSS* database ranks for Example 5.1.

0. *gGlOSS correctly predicted this for rank G (Estimate*$(l, q, db_5) = 0$ *for G), and so* db_5 *does not appear in G. However,* db_5 *does appear in H, because Estimate*$(l, q, db_5) = 0.2$ *for H.*

Let us focus on the G rank: db_2 *is the top database in G, with Estimate*$(l, q, db_2) = 0.8$. *The real goodness of* db_2 *for q is Goodness*$(l, q, db_2) = 0.4$. *From the ranks of Table 1,* $\mathcal{R}_1 = \frac{0.4}{0.9}$: *if we access* db_2, *the top database from the G rank, we obtain Goodness*$(l, q, db_2) = 0.4$, *whereas the best database for q is* db_1, *with Goodness*$(l, q, db_1) = 0.9$. *Similarly,* $\mathcal{R}_3 = \frac{0.4+0.9+0.3}{0.9+0.4+0.3} = 1$. *In this case, by accessing the top three databases in the G rank we access exactly the top three databases in the ideal rank, and thus* $\mathcal{R}_3 = 1$. *However,* $\mathcal{R}_4 = \frac{0.4+0.9+0.3}{0.9+0.4+0.3+0.2} = 0.89$, *since the G rank does not include* db_4 *(Estimate*$(l, q, db_4) = 0$*), which is actually useful for q (Goodness*$(l, q, db_4) = 0.2$*).*

Now consider the H rank. H includes all the databases that have Goodness> 0 in exactly the same order as G. Therefore, the $\mathcal{R}_n$ *metric for H coincides with that for G, for all n. However, rank G is in some sense better than rank H, since it predicted that* db_5 *has zero goodness, as we mentioned above. H failed to predict this. The* $\mathcal{R}_n$ *metric does not distinguish between the two ranks. This is why we introduce our following metric.*

As the previous example motivated, we need another metric, $\mathcal{P}_n$, to distinguish between *gGlOSS* ranks that include useless databases and those that do not. Given a *gGlOSS* rank G for query q, $\mathcal{P}_n$ is the fraction of $Top_n(G)$, the top n databases of G (which have a non-zero *Estimate* for being in G), that actually have non-zero goodness for query q:

$$\mathcal{P}_n = \frac{|\{db \in Top_n(G) |\ Goodness(l, q, db) > 0\}|}{|Top_n(G)|}$$

(Actually, $\mathcal{P}_n = 1$ if for all db, $Estimate(l, q, db) = 0$.) Note that $\mathcal{P}_n$ is independent of the ideal database rank I: it just depends on how many databases that *gGlOSS* estimated as potentially useful turned out to actually be useful for the query. From the point of view of the end users, a ranking with higher $\mathcal{P}_n$ is better because it leads them to fewer fruitless database searches.

Example 5.1 (cont.) *In the previous example,* $\mathcal{P}_4 = \frac{3}{3} = 1$ *for G, because all of the databases in G have actual non-zero goodness. However,* $\mathcal{P}_4 = \frac{3}{4} = 0.75$ *for H: of the four databases in H, only three have non-zero goodness.*

6 Evaluating *gGlOSS*

In this section we evaluate different *gGlOSS* ranking algorithms experimentally. We first describe the real-user queries and databases that we used in the experiments. Then, we report results for $Max(l)$ and $Sum(l)$, the two *gGlOSS* ranks of Section 4.

6.1 Queries and Databases

To evaluate *gGlOSS* experimentally, we used real-user queries and databases. The queries that we used where profiles that real users submitted to the SIFT Netnews server developed at Stanford [15] [8]. Users send profiles in the form of boolean or vector-space queries to the SIFT server, which in turn filters Netnews articles every day and sends the articles matching the profiles to the corresponding users. We used the 6800 vector-space profiles that were active on the server in December 1994.

To evaluate the *gGlOSS* performance using these 6800 queries, we used 53 newsgroups as 53 databases: we took a snapshot of the articles that were active at the Stanford Computer-Science-Department news host on one arbitrary day, and used these articles to populate the 53 databases. We selected all the newsgroups in the **`comp.databases`**, **`comp.graphics`**, **`comp.infosystems`**, **`comp.security`**, **`rec.arts.books`**, **`rec.arts.cinema`**, **`rec.arts.comics`**, and **`rec.arts.theatre`** hierarchies that had active documents in them when we took the snapshot.

We indexed the 53 databases and evaluated the 6800 queries on them using the SMART system (version 11.0) developed at Cornell University. To keep our experiments simple, we chose the same weighting algorithms for the queries and the documents across all of the databases. We indexed the documents using the SMART *ntc* formula, which generates document weight vectors using the cosine-normalized *tf-idf* product [6]. We indexed the queries using the SMART *nnn* formula, which generates query weight vectors using the word frequencies in the queries. The similarity coefficient between a document vector and a query vector is computed by taking the inner product of the two vectors.

For each query and *gGlOSS* ranking algorithm we compared the ideal rank against the *gGlOSS* rank using the methodology of Section 5. We evaluated each

[8] SIFT is accessible at `http://sift.stanford.edu`.

query at each of the 53 databases to generate its ideal database rank. For a fixed *gGlOSS* ranking definition and a query, we computed the rank of databases that *gGlOSS* would produce for that query: we extracted the (partial) information that *gGlOSS* needs from each of the 53 databases. For each query word, *gGlOSS* needs the number of documents in each database that include the word, and the sum of the weight of the word in each of these documents. To extract all this information, we queried the 53 databases using each query word individually, which totaled an extra 18,213 queries. We should stress that this is just the way we performed the experiments, not the way a *gGlOSS* server will obtain the information it needs about each database: in a real system, each database will periodically scan its indexes, generate the information that *gGlOSS* needs, and send it to the *gGlOSS* server. (See Section 4.)

6.2 Experimental Results

In this section we experimentally compare the *gGlOSS* database ranks against the ideal ranks in terms of the $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics. We study which of the $Max(l)$ and $Sum(l)$ database ranks is better at predicting ideal rank $Ideal(l)$, and what impact the threshold l has on the performance of *gGlOSS*. We also investigate whether keeping both the F and W matrices of Section 4 is really necessary, since *gGlOSS* needs only one of these matrices to compute ranks $Max(0)$ and $Sum(0)$ (Section 4.2).

Ideal database rank $Ideal(0)$ considers any document with a non-zero similarity to the query as useful. Ranks $Max(0)$ and $Sum(0)$ are identical to $Ideal(0)$, and so they have $\mathcal{R}_n = \mathcal{P}_n = 1$ for all n. Consequently, if a user wishes to locate databases where the overall similarity between documents and the given query is highest and any document with non-zero similarity is interesting, *gGlOSS* should use the $Max(0)$ (or, identically, $Sum(0)$) ranks and get perfect results.

To study the impact of higher rank thresholds, Figures 1 and 2 show results for the $Ideal(0.2)$ ideal rank. We show $\mathcal{R}_n$ and $\mathcal{P}_n$ for values of n ranging from 1 to 15. We do not report data for higher n's because most of the queries have fewer than 15 useful databases according to $Ideal(0.2)$ and hence, the results for high values of n are not that significant. Figure 2 shows that rank $Sum(0.2)$ has perfect $\mathcal{P}_n$ ($\mathcal{P}_n = 1$) for all n, because if a database db has $Estimate(0.2, q, db) > 0$ according to the $Sum(0.2)$ rank, then $Goodness(0.2, q, db) > 0$ according to $Ideal(0.2)$. In other words, rank $Sum(0.2)$ only includes databases that are guaranteed to be useful. Rank $Max(0.2)$ may include databases not guaranteed to be useful, yielding higher $\mathcal{R}_n$ values (Figure 1), but lower $\mathcal{P}_n$ values (Figure 2).

To decide whether *gGlOSS* really needs to keep both matrices F and W (Section 4), we also use ranks $Max(0)$ and $Sum(0)$ to approximate rank $Ideal(0.2)$. *gGlOSS* needs only one of the two matrices to compute these ranks (Section 4.2). Since ranks $Max(0)$ and $Sum(0)$ are always identical, we just present their data once labeled $Max(0)/Sum(0)$. Figure 1 shows that the $Max(0)$ rank has the highest values of $\mathcal{R}_n$. This rank assumes a threshold $l = 0$, and thus it tends to include more databases than its counterparts with threshold 0.2. This is also why $Max(0)$ has much lower $\mathcal{P}_n$ values (Figure 2) than $Max(0.2)$ and $Sum(0.2)$: it includes more databases that have zero goodness according to $Ideal(0.2)$.

In summary, if the users are interested in not missing any useful database, but are willing to search some useless ones, then $Max(0)$ is the best choice for *gGlOSS*, and *gGlOSS* can do without matrix F. If the users wish to avoid searching useless databases, then $Sum(0.2)$ is the best choice. Unfortunately, $Sum(0.2)$ also has low $\mathcal{R}_n$ values, which means it can also miss some useful sources. As a compromise, a user can have $Max(0.2)$, which has much better $\mathcal{P}_n$ values than $Max(0)$ and generally better $\mathcal{R}_n$ values than $Sum(0.2)$. Also, note that in the special case where users are interested in accessing only one or two databases ($n = 1, 2$) then $Max(0.2)$ is the best choice for the $\mathcal{R}_n$ metric. In this case, it is worthwhile for *gGlOSS* to keep both matrices F and W.

To show the impact of the rank thresholds, Figures 3 and 4 show the $\mathcal{R}_n$ and $\mathcal{P}_n$ values for the different ranks and a fixed $n = 3$, and for values of the threshold l from 0 to 0.4. For larger values of l, most of the queries have no database with goodness greater than zero. For example, for ideal rank $Ideal(0.6)$ each query has on average only 0.29 useful databases. Therefore, we only show the data for threshold 0.4 and lower. At first glance one might expect the $\mathcal{R}_n$ and $\mathcal{P}_n$ performance of $Max(0)$ not to change as the threshold l varies, since the ranking it computes is independent of the desired l. However, as l increases, the ideal rank $Ideal(l)$ changes, and the static estimate provided by $Max(0)$ performs worse and worse for $\mathcal{P}_n$. The $Max(l)$ and $Sum(l)$ ranks do take into account the target l values, and hence do substantially better. Our earlier conclusion still holds: strategy $Sum(l)$ is best at avoiding useless databases, while $Max(0)$ provides the best $\mathcal{R}_n$ values (at the cost of low $\mathcal{P}_n$ values).

In summary, *gGlOSS* generally predicts fairly well the best databases for a given query. Actually, the more *gGlOSS* knows about the users' expectations, the better *gGlOSS* can rank the databases for the query. If high values of both $\mathcal{R}_n$ and $\mathcal{P}_n$ are of in-

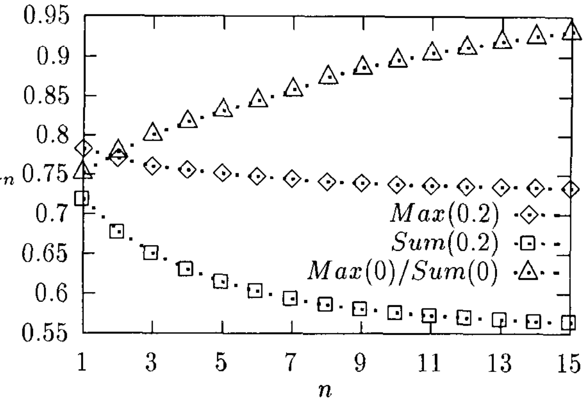

Figure 1: Parameter $\mathcal{R}_n$ as a function of n, the number of databases examined from the ranks, for the *Ideal*(0.2) ideal database ranking and the different *gGlOSS* rankings.

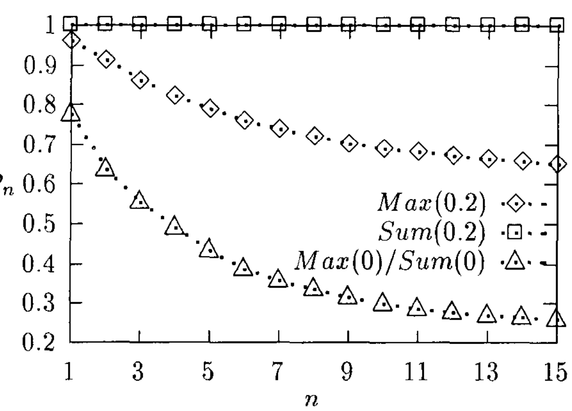

Figure 2: Parameter $\mathcal{P}_n$ as a function of n, the number of databases examined from the ranks, for the *Ideal*(0.2) ideal database ranking and the different *gGlOSS* rankings.

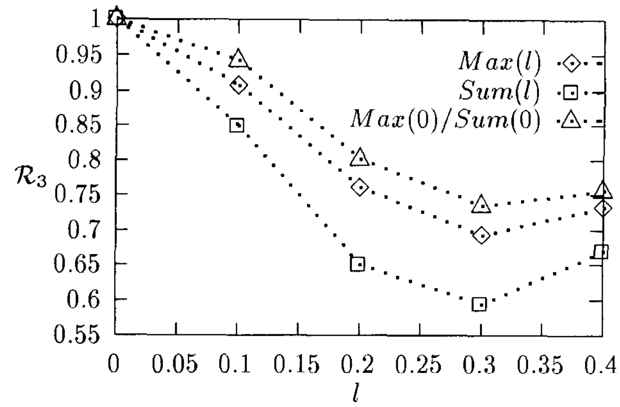

Figure 3: Parameter $\mathcal{R}_3$ as a function of the threshold l, for ideal rank *Ideal*(l).

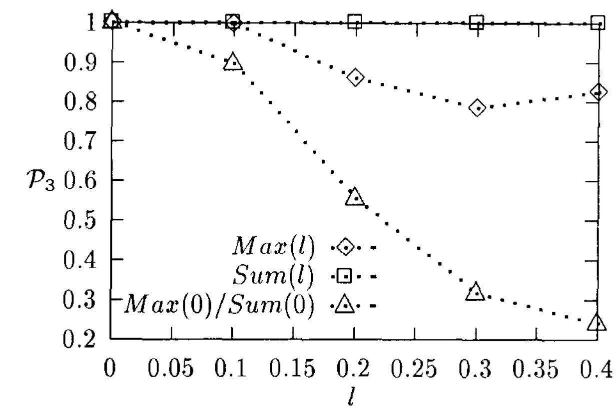

Figure 4: Parameter $\mathcal{P}_3$ as a function of the threshold l, for ideal rank *Ideal*(l).

terest, then *gGlOSS* should produce ranks based on the high-correlation assumption of Section 4.1: rank *Max*(l) is the best candidate for rank *Ideal*(l) with $l > 0$. If only high values of $\mathcal{R}_n$ are of interest, then *gGlOSS* can do without matrix F, and produce ranks *Max*(0) or *Sum*(0). If only high values of $\mathcal{P}_n$ are of interest, then *gGlOSS* should produce ranks based on the disjoint-scenario assumption of Section 4.2: rank *Sum*(l) is the best candidate. For rank *Ideal*(0), ranks *Max*(0) and *Sum*(0) give perfect answers.

7 Alternative Ideal Ranks

Section 3 presented a way of defining the goodness of a database for a query, and also showed a problem with its associated ideal database rank. In this section we explore alternative ideal database ranks for a query. (Even other possibilities are discussed in [16].)

We can organize the different database ranks for a query into two classes, according to whether the ranks depend on the number of relevant documents for the query in each database or not [17]. The first two alternative ranks belong to the first class.

The first rank, *Rel_All*, simply orders the databases based on the number of relevant documents they contain for the given query. By relevant we mean that the user who submits q will judge these documents to be of interest. To see a problem with this rank, consider a database *db* that contains, say, three relevant documents for some query q. Unfortunately, it turns out that the search engine at *db* does not include any of these documents in the answer to q. So, the user will not benefit from these three relevant documents. Thus, we believe it is best to evaluate the ideal goodness of a database by what its search engine might retrieve, not by what potentially relevant documents it might contain. Notice that a user might eventually obtain these relevant documents by successively modifying the query. Our model would treat each of these queries separately, and decide which databases are the best for each individual query.

Our second rank, *Rel_Rank*(l), improves on *Rel_All* by considering only the relevant documents in each database that have a similarity to q greater than a threshold l, as computed by the individual databases. The underlying assumption is that users will not examine documents with lower similarity in the answers to the queries, because these documents are unlikely to be useful. This definition does not suffer from the problem of the *Rel_All* rank: we simply ignore relevant documents that db does not include in the answer to q with sufficiently high similarity. However, in general we believe that ranks based on end-user relevance are not appropriate for evaluating schemes like *gGlOSS*. That is, the best we can hope for any tool like *gGlOSS* is that it predicts the answers that the databases will give when presented with a query. If the databases cannot rank the relevant documents high and the non-relevant ones low with complete index information, it is asking too much that *gGlOSS* derive relevance judgments with only partial information. Consequently, the database rankings that are not based on document relevance seem a more useful frame of reference to evaluate the effectiveness of *gGlOSS*. Hence, the remaining ranks that we consider do not use relevance information.

The *Global*(l) rank is based on considering the contents of all the databases as a single collection. The documents are then ranked according to their "global" similarity to query q. We consider only those documents having similarity to q greater than a threshold l. The *Goodness* metric associated with rank *Global*(l) would add the similarities of the acceptable documents. The problem with this rank is related to the problem with the *Rel_All* rank: a database db may get high goodness values for documents that do not appear (high) in the answer that the database produces for q. Therefore, db is not as useful to q as the *Goodness* metric predicted. To avoid this problem, the goodness of a database for a query should be based on the document rank that the database generates for the given query.

The definition of *Goodness* of Section 3 does not rely on relevance judgments, and is based on the document ranks that the databases produce for the queries. Therefore, that definition does not suffer from the problems of the alternative ranks that we considered so far in this section. However, as we mentioned in Section 3, a problem is that the similarities computed at the local databases can depend on the characteristics of the collections, and thus they might not be valid globally. The next definition attempts to compensate for this collection-dependent computations.

The next rank, *Local*(l), considers only the set of documents in db having *scaled* similarity to q greater than a threshold l. We scale the similarities coming from different databases differently, to compensate for the collection-dependent way in which these similarities are computed. Also, we should base the goodness of each database on its answer to the query, to avoid the anomalies we mentioned above for the *Rel_All* and the *Global* ranks. One way to achieve these two goals is to multiply the similarities computed by database db by a positive constant *scale*(q, db):

$$Goodness(l, q, db) = scale(q, db) \times \sum_{d \in Scaled\_Rank(l,q,db)} sim(q, d)$$

where *scale*(q, db) is the scaling factor associated with query q and database db, and $Scaled\_Rank(l, q, db) = \{d \in db | sim(q, d) \times scale(q, db) > l\}$.

The problem of how to modify the locally computed similarities to compensate for collection-dependent factors in their computation has received attention recently in the context of the *collection-fusion* problem. The collection-fusion problem [18, 10, 19] studies how to merge document rankings for a query from different sources into a single document ranking. (See [10] for a way to use *GlOSS*-like information to scale the similarities computed at each source.) In general, determining what scaling factor to use to define the *Local*(l) ideal database rank is an interesting problem that we will explore in the near future. Also, if we incorporate scaling into the *Goodness* definition, we should modify *gGlOSS*'s ranks to imitate this scaling.

In summary, none of the database ranking schemes that we have discussed is perfect, including the ones we used for our experiments. Each scheme has its limitations, and hence, should be used with care. However, we believe that the ranking that we used (Section 3) is a good starting point for now, until more work on scaling is done.

8 Decentralizing *gGlOSS*

So far, we described *gGlOSS* as a centralized server that users query to select the most promising sources for their queries. In this section we show how we can build a more distributed version of *gGlOSS* using essentially the same methodology that we developed in the previous sections.

Suppose that we have a number of *gGlOSS* servers $G_1, \ldots, G_s$, indexing each a set of databases as we described in the previous sections. (Each of these servers can index the databases at one university or company, for example.) We will now build a *higher-level* *gGlOSS* server, *hGlOSS*, that summarizes the contents of the *gGlOSS* servers in much the same way as the *gGlOSS* servers summarize the contents of the underlying databases. [9] The users will then query the *hGlOSS*

[9] Although our discussion focuses on a 2-level hierarchy of servers, we can use the same principles to construct deeper hierarchies.

server first, and obtain a rank of the *gGlOSS* servers according to how likely they are to have indexed useful databases. Later, the *gGlOSS* servers will produce the final database ranks. Although the *hGlOSS* server is still a single entry point for users to search for documents, the size of this server will be so small that it will be inexpensive to massively replicate it, distributing the access load among the replicas. In this way, organizations will be able to manage their own "traditional" *gGlOSS* servers, and will let replicas of a logically unique higher-level *gGlOSS*, *hGlOSS*, concisely summarize the contents of their *gGlOSS* servers.

The key point is to notice that *hGlOSS* can treat the information about a database at a traditional *gGlOSS* server in the same way as the traditional *gGlOSS* servers treat the information about a document at the underlying databases. The "documents" for *hGlOSS* will be the *database summaries* at the *gGlOSS* servers.

To keep the size of the *hGlOSS* server small, the information that the *hGlOSS* server keeps about a *gGlOSS* server G_r is limited. For example, *hGlOSS* keeps one or both of the following matrices (see Section 4):

- $\mathsf{H} = (\mathsf{h}_{rj})$: h_{rj} is the number of *databases* in *gGlOSS* G_r that contain word t_j
- $\mathsf{D} = (\mathsf{d}_{rj})$: d_{rj} is the sum of the number of documents that contain word t_j in each database in *gGlOSS* G_r

In other words, for each word t_j and each *gGlOSS* server G_r, *hGlOSS* needs (at most) two numbers, in much the same way as the *gGlOSS* servers summarize the contents of the document databases (Section 4).

Example 8.1 *Consider a gGlOSS server G_r and the word* computer. *Suppose that the following are the databases in G_r having documents with the word* computer *in them, together with their corresponding gGlOSS weights and frequencies:*

$$\text{computer} : (db_1, 5, 3.4), (db_2, 2, 1.8), (db_3, 1, 0.3)$$

That is, database db_1 has five documents with the word computer *in them, and their added weight is 3.4 for that word, database db_2 has two documents with the word* computer *in them, and so on. hGlOSS will only know that the word* computer *appears in three databases in G_r, and that the sum of the number of documents for the word and the databases is $5 + 2 + 1 = 8$. hGlOSS will not know the identities of these databases, or the individual document counts associated with the word and each database.*

We can now use the same methodology we used for *gGlOSS* in the previous sections: given a query q, we define the goodness of each *gGlOSS* server G_r for the query: for example, we can take the database rank that G_r produces for q, together with the goodness estimate for each of these databases according to G_r, and define the goodness of G_r for q as a function of this rank. This computation is analogous to how we computed the goodness of the *databases* in Section 3.

After defining what the goodness of each *gGlOSS* server is for query q, we define how *hGlOSS* is going to estimate this goodness given only partial information about each *gGlOSS* server. *hGlOSS* will determine the *Estimate* for a *gGlOSS* server G_r using the vectors h_{r*} and d_{r*} for G_r in a way analogous to how the *gGlOSS* servers determine the *Estimate* for a database db_i using the f_{i*} and w_{i*} vectors. After defining the *Estimate* for each *gGlOSS* server, *hGlOSS* ranks the *gGlOSS* servers so that the users can access the most promising servers first, i.e., those most likely to index useful databases.

Due to space limitations, we are unable to present detailed results for *hGlOSS*. However, simply to illustrate its potential, here we briefly describe one experiment. For this, we divide the 53 databases of Section 6 into five randomly-chosen groups of around ten databases each. Each of these groups corresponds to a different *gGlOSS* server.

We assume that the *gGlOSS* servers approximate ideal rank *Ideal*(0) with the *Max*(0) database rank. Next, we define the goodness of a *gGlOSS* server G_r for a query q as the number of databases indexed by G_r having a goodness *Estimate* for q greater than zero. This definition determines the ideal rank of *gGlOSS* servers. To approximate this ideal rank, *hGlOSS* periodically receives the H matrix defined above from the underlying *gGlOSS* servers. For query q with words $t_1, \ldots, t_n$ and *gGlOSS* server G_r, $\mathsf{h}_{r1}, \ldots, \mathsf{h}_{rn}$ are the database counts for G_r associated with the query words. (Word t_1 appears in h_{r1} *databases* in *gGlOSS* server G_r, and so on.) Assume that $\mathsf{h}_{r1} \leq \ldots \leq \mathsf{h}_{rn}$. Then, *hGlOSS* estimates the goodness of G_r for q as h_{rn}. In other words, *hGlOSS* estimates that there are h_{rn} databases in G_r that have a non-zero goodness estimate for q.

Table 2 shows the different values of the (adapted) $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics for the 6,800 queries of Section 6. Note that $\mathcal{P}_n = 1$ for all n, because every time *hGlOSS* chooses a *gGlOSS* server using the ranking described above, this server actually has databases with non-zero estimates. From the high values for $\mathcal{R}_n$ it follows that *hGlOSS* is extremely good at ranking "useful" *gGlOSS* servers.

Our single experiment used a particular ideal ranking and evaluation strategy. We can also use the other rankings and strategies we have presented adapted to the *hGlOSS* level, and tuned to the actual user requirements. Also, the *hGlOSS* server will be very small in

n	$\mathcal{R}_n$	$\mathcal{P}_n$
1	0.985	1
3	0.994	1
5	1	1

Table 2: The $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics for *hGlOSS* and our sample experiment.

size and easily replicated, thus eliminating the potential bottleneck that the centralized *gGlOSS* architecture can suffer.

9 Conclusion

We have shown how to construct an information broker for both vector-space text databases and hierarchies of brokers. Based on compact collected statistics, the broker can provide very good hints for finding the relevant databases, or finding relevant lower-level brokers with more information for a given query. An important feature of our approach is that the same machinery can be used for both types of brokers, either the lower-level or the higher-level ones. Our experimental results show that the *gGlOSS* and the *hGlOSS* brokers are quite promising and could provide useful services in large, distributed information systems.

Acknowledgments

We thank Anthony Tomasic, Tak Yan, and the anonymous referees for their useful comments on the paper.

References

[1] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. The effectiveness of *GlOSS* for the text-database discovery problem. In *Proceedings of the 1994 ACM SIGMOD Conference*, May 1994.

[2] Michael F. Schwartz, Alan Emtage, Brewster Kahle, and B. Clifford Neuman. A comparison of Internet resource discovery approaches. *Computer Systems*, 5(4), 1992.

[3] Katia Obraczka, Peter B. Danzig, and Shih-Hao Li. Internet resource discovery services. *IEEE Computer*, September 1993.

[4] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. Precision and recall of *GlOSS* estimators for database discovery. In *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS'94)*, September 1994.

[5] Gerard Salton and Michael J. McGill. *Introduction to modern information retrieval.* McGraw-Hill, 1983.

[6] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison Wesley, 1989.

[7] B. Clifford Neuman. The Prospero File System: A global file system based on the Virtual System model. *Computer Systems*, 5(4), 1992.

[8] Tim Berners-Lee, Robert Cailliau, Jean-F. Groff, and Bernd Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1(2), 1992.

[9] Brewster Kahle and Art Medlar. An information system for corporate users: Wide Area Information Servers. Technical Report TMC199, Thinking Machines Corporation, April 1991.

[10] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the* 18^{th} *Annual SIGIR Conference*, 1995.

[11] Mark A. Sheldon, Andrzej Duda, Ron Weiss, James W. O'Toole, and David K. Gifford. A content routing system for distributed information servers. In *Proceedings of the* 4^{th} *International Conference on Extending Database Technology*, 1994.

[12] Andrzej Duda and Mark A. Sheldon. Content routing in a network of WAIS servers. In *14th IEEE International Conference on Distributed Computing Systems*, 1994.

[13] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado-Boulder, August 1994.

[14] Anthony Tomasic, Luis Gravano, Calvin Lue, Peter Schwarz, and Laura Haas. Data structures for efficient broker implementation. Technical report, IBM Almaden Research Center, June 1995.

[15] Tak W. Yan and Héctor García-Molina. SIFT–a tool for wide-area information dissemination. In *Proceedings of the USENIX 1995 Technical Conference*, pages 177–86, 1995.

[16] Luis Gravano and Héctor García-Molina. Generalizing *GlOSS* to vector-space databases and broker hierarchies. Technical Report STAN-CS-TN-95-21, Stanford University, May 1995. Available as `ftp://db.stanford.edu/pub/gravano/1995/stan.cs.tn.95.21.ps`.

[17] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. Precision and recall of *GlOSS* estimators for database discovery. Technical Report STAN-CS-TN-94-010, Stanford University, July 1994. Available as `ftp://db.stanford.edu/pub/gravano/1994/stan.cs.tn.94.010.ps`.

[18] Ellen M. Voorhees, Narendra K. Gupta, and Ben Johnson-Laird. The collection fusion problem. In *Proceedings of the* 3^{rd} *Text Retrieval Conference (TREC-3)*, 1995.

[19] Alistair Moffat and Justin Zobel. Information retrieval systems for large document collections. In *Proceedings of the* 3^{rd} *Text Retrieval Conference (TREC-3)*, 1995.