Generalizing *GlOSS* To Vector-Space Databases and Broker Hierarchies [*]

Luis Gravano Héctor García-Molina

Computer Science Department
Stanford University
Stanford, CA 94305-2140, USA

{gravano,hector}@cs.stanford.edu
FAX: +1-415-725-2588

Abstract

As large numbers of text databases have become available on the Internet, it is getting 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 [GGMT94a], 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.

Keywords: *resource discovery, database selection, vector-space retrieval model, information retrieval, text databases*

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 in search for information on a given topic: not only would such an exhaustive search take a long time to complete, but it could also cost the users lots of money, 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 is a specific instance of the more general resource-discovery problem [SEKN92, ODL93].

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 documents that are available anywhere. This approach clearly does not

[*] 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.

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 for the given queries 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 have a broker that users query first to obtain a rank of the *potentially* most useful databases for their query. This broker keeps only *partial information* on the contents of each database, so it scales with the growing number of available databases, but this information covers the *full-text* content of the documents, so that the useful sources are identified. In [GGMT94a] we presented *GlOSS* (for *Glossary-Of-Servers Server*) [1], a centralized tool 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.

Example 1.1 *Consider a boolean query "find documents with the word* computer *and the word* science *in them." The documents that match this query, according to the boolean model of document retrieval, are those containing both the word* computer *and the word* science*. Suppose that there are two databases available to us, db_1 and db_2. GlOSS knows that db_1 has, say, 10 documents with the word* computer *in them, and 20 documents with the word* science *in them. GlOSS also knows that there are 100 documents in database db_1. (The databases periodically collect and send this information to the GlOSS server.)*

However, GlOSS does not know the identities of the documents that contain each word, and therefore, it does not know how many documents in db_1 contain the two words in the query. GlOSS has to estimate this number. One way GlOSS does this estimation is by assuming that keywords appear in documents following uniform and independent probability distributions. (See [GGMT94a].) Using this assumption, db_1 has $\frac{10\times 20}{100} = 2$ documents matching the given query.

If db_2 has 1000 documents, 50 of which have the word computer*, and 100 of which have the word* science *in them, then GlOSS estimates the number of documents matching the query in db_2 as $\frac{50\times 100}{1000} = 5$. Therefore, GlOSS ranks database db_2 higher than database db_1, because db_2 has a higher estimate for the number of documents matching the given query.*

Although the boolean model of document retrieval is widely used, it is a rather primitive one. The information-retrieval community has worked for many years on developing more useful document-retrieval algorithms and models. One of the most popular models is the *vector-space retrieval model* [SM83, Sal89]. 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.

Example 1.2 *Consider a vector-space database db with N documents, and some word t. Suppose that this word appears in df documents of db, and tf times in a document doc. The importance, or* weight *w, of this word in document doc could be determined, for example, as [Sal89]:*

$$w = tf \times \log \frac{N}{df}$$

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

Therefore, the weight of the word in document doc is higher if the word appears in doc many times, and in very few other documents in database db.

*Now, consider the query q =*information retrieval. *We need to express this query as a weight vector too. For example, we could use the number of times that a word appears in the query as the weight of that word in the query vector. Therefore, both the word* information *and the word* retrieval *have weight one in the query vector, and all of the other words in the database vocabulary have weight zero.*

Suppose that a document doc_1 in db has weight 0.3 for word information *and weight 0.2 for word* retrieval. *We can compute $sim(q, doc_1)$, the similarity of doc_1 and q, as the inner product of the doc_1 and q weight vectors. Therefore, $sim(q, doc_1) = 0.3 \times 1 + 0.2 \times 1 = 0.5$. Similarly, if doc_2 in db has weight 0.7 for word* information *and weight 0 for word* retrieval*, $sim(q, doc_2) = 0.7 \times 1 + 0 \times 1 = 0.7$. Consequently, db ranks doc_2 as more promising than doc_1 for query q.*

The information-retrieval theory community has designed smart algorithms that, given a database and a query expressing an information need, try to find the documents that are relevant to the information need [SM83, Sal89]. In this paper we do not deal with this problem. Instead, we focus on choosing *among different* databases that already have local search engines. How good these search engines are at finding relevant documents is outside the scope of this paper [SM83].

As discussed earlier, the goal of *GlOSS* is to "guess" the correct ranking of databases (by whatever correctness metric we target) without actually accessing the databases and without keeping full indexes. However, since *GlOSS* only understands *boolean* queries, we would expect that its rankings do not approximate well actual database rankings based on document and query vector weights.

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 analyze several different options to decide the *goodness* of a database for a query, and to define 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: 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, the experimental results are 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 *hGlOSS* server is much smaller than the *gGlOSS* brokers, we can easily replicate the *hGlOSS* server so that it does not become a performance bottleneck, thus distributing the load of the whole searching system.

In Section 3 we analyze several possible "ideal" database ranks for a query. Section 4 presents different ways in which *gGlOSS* approximates the ideal database ranks using partial information, and Section 6 reports experimental results for *gGlOSS* using the methodology of Section 5. Finally, in Section 7 we show how to build the higher-level *hGlOSS* servers.

2 Related work

The text-database discovery problem and, more generally, the resource discovery problem [SEKN92, ODL93], have received considerable attention during the last few years.

One approach to solving the text-database discovery problem is to let users "browse" information about the different databases. Well-known examples include Gopher [SEKN92] and the World-Wide Web [BLCGP92]. The Prospero File System [Neu92] lets users organize information available on the Internet through the definition (and sharing) of customized views of the different objects and services available to them.

A different approach is to let users query a database of "meta-information" about the available databases. For example, WAIS [KM91] provides a "directory of servers." This "master" database contains a set of documents, each describing (in English) the contents of a database on the network. The users first query the master database, and once they have identified potential databases, direct their query to these databases. Many search facilities have been created for Gopher and the World-Wide Web, like the Veronica service [Fos92] for Gopher, and the Lycos service [2] for the World-Wide Web, for example. 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]).

[Sch90] follows a probabilistic approach to the resource-discovery problem that consists of two phases: a dissemination phase, during which information about the contents of the databases is replicated at randomly chosen sites, and a search phase, where several randomly chosen sites are searched in parallel. Also, sites are organized into "specialization subgraphs." If one node of such a graph is reached during the search process, the search proceeds "non-randomly" in this subgraph, if it corresponds to a specialization relevant to the query being executed. See also [Sch93].

In Indie (shorthand for "Distributed Indexing")[DLO92], every "broker" has an associated boolean query (called a "generator rule"). Each broker indexes (not necessarily local) documents that satisfy its generator rule. Whenever a document is added to an information source, the brokers whose generator rules match the new document are sent a descriptor of the new document. The generator objects associated with the brokers are gathered by a "directory of servers," that users query initially to obtain a list of the brokers whose generator rules match the given query. See also [DANO91]. [SA89], [BC92], and [OM92] are other examples of this type of approach in which users query "meta-information" databases.

The "content-based routing" system of [SDW$^+$94, DS94] keeps a "content label" for each information server (or collection of objects, more generally), with attributes describing the contents of the collection. Users assign values to the content-label attributes in their queries until a sufficiently small set of information servers is selected, and they can browse the possible values of each content-label attribute. See also [BDB$^+$92].

The Harvest system [BDH$^+$94] 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.

[2] Lycos is accessible at `http://fuzine.mt.cs.cmu.edu/mlm/lycos-home.html`.

[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`.

The WHOIS++ directory service [WS93] organizes the WHOIS++ servers into a distributed "directory mesh" that can be searched: each server automatically generates a "centroid" listing the words it contains (for the different attributes). Centroids are gathered by index servers, that in turn must generate a centroid describing their contents. The index-server centroids may be passed to other index servers, and so on. An index server forwards a given query to the (index) servers whose centroids match the query.

In [FY93], every site keeps statistics about the type of information it receives along each link connecting to other sites. When a query arrives in a site, the site forwards it through the most promising link according to these statistics.

In [GGMT94a] and [GGMT94b] we described *GlOSS* [5] (for *Glossary-Of-Servers Server*), a server that helps users select databases supporting the *boolean model* of document retrieval: given a boolean query (e.g., "*find documents with the word* computer *and the word* science *in them*"), these databases return a *set* of documents that match the query. We described a variety of ways in which *GlOSS* estimates the size of the answer set in each of the databases, using partial information on the databases. More specifically, for every database *db*, *GlOSS* only knows how many documents in *db* include each word, not the identity of these documents. In [GGMT94a] and [GGMT94b] we reported results on the performance of *GlOSS* for such scenario, and estimated the storage requirements of *GlOSS*: *GlOSS* required only around 2% as much storage as a full index of the databases, while choosing the "right" databases for most of the queries that we tried, for some definitions of what "right" means. The main differences between our initial *GlOSS* work and this paper are:

- We extend the approach to deal with vector-space databases: estimating vector-space searching is harder than estimating boolean searching.
- We present ways of ranking vector-space databases for a query, and we compare these rankings against the *gGlOSS* rankings.
- We evaluate *gGlOSS* more realistically by considering 53 databases, as opposed to the six we used in the original work.
- We define a hierarchy of *gGlOSS* servers that allows for wide replication of the (small) top levels.

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 explores how we can determine this *ideal database rank*. The next section explores how *gGlOSS* will try to rank the databases in a way that resembles the ideal rank as closely as possible.

One idea is to rank the databases based on the number of documents they contain that are *relevant* to the given query. However, we believe that this is not a useful scheme. To illustrate why, say a database *db* contains three relevant documents for some query *q*. (By relevant we mean

[5] *GlOSS* is accessible at `http://gloss.stanford.edu`.

that the user who submits q will judge these three documents to be of interest.) 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. Therefore, in this paper, we will only focus on the *answers* that each database gives to the user queries. We will not attempt to rank the databases based on their contents, or on relevance judgments about the documents, but rather on the search engine's answers to the users' queries.

Given a set of databases DB and a query q, we define four different ideal database ranks. Each of these ranks orders the databases according to their *goodness* for q, for different definitions of what goodness means. Thus, each rank has associated a different definition of *Goodness(q, db)*, the goodness of database *db* for query q.

The first two ranks, $All_W(l)$ and $All_D(l)$, consider only $Docs(l, q, db)$, the set of documents in *db* having similarity with q greater than a given *threshold l*. The underlying assumption is that these documents are the ones that are most likely to be relevant to the user: documents with lower similarity are unlikely to be useful, and therefore we ignore them. The *Goodness* metric associated with rank $\mathbf{All_W(l)}$ adds the similarities of the "acceptable" documents:

$$Goodness(q, db) \quad = \sum_{doc \in Docs(l,q,db)} sim(q, doc) \tag{1}$$

where $Docs(l, q, db) = \{doc \in db | sim(q, doc) > l\}$, for some given threshold l, and $sim(q, doc)$ is the similarity between query q and document *doc*.

Alternatively, the *Goodness* metric associated with rank $\mathbf{All_D(l)}$ counts how many "acceptable" documents there are:

$$Goodness(q, db) \quad = \quad |Docs(l, q, db)| \tag{2}$$

For our next two ranks, $Top_W(k)$ and $Top_D(k)$, we take the documents from all the databases together with their similarity with q, and generate a single document rank using these similarities. We then consider only the top k documents in this rank, for some k. For each database *db*, we consider $Top(k, DB, q, db)$, the *db* documents that appear among the overall top k documents for q. This set of documents determines the *Goodness* metrics associated with ranks $Top_W(k)$ and $Top_D(k)$. The *Goodness* metric associated with rank $\mathbf{Top_W(k)}$ adds the similarities of these documents:

$$Goodness(q, db) \quad = \sum_{doc \in Top(k,DB,q,db)} sim(q, doc) \tag{3}$$

where $Top(k, DB, q, db) = \{doc \in db | doc$ is one of the top-k documents for q in the DB databases$\}$.

Alternatively, the *Goodness* metric associated with rank $\mathbf{Top_D(k)}$ counts how many *db* documents appear among the overall top k documents for q:

$$Goodness(q, db) \quad = \quad |Top(k, DB, q, db)| \tag{4}$$

Example 3.1 *Consider three databases, db_1, db_2, and db_3, a query q, and the answers that the three databases give when presented with query q:*

$$\begin{aligned} db_1 &: (d_1^1, 0.9), (d_2^1, 0.9), (d_3^1, 0.1) \\ db_2 &: (d_1^2, 0.8), (d_2^2, 0.4), (d_3^2, 0.3), (d_4^2, 0.1) \\ db_3 &: (d_1^3, 0.4), (d_2^3, 0.1) \end{aligned}$$

$All_W(0.2)$		$All_D(0.2)$		$Top_W(5)$		$Top_D(5)$	
db	*Goodness*	*db*	*Goodness*	*db*	*Goodness*	*db*	*Goodness*
db_1	*1.8*	db_2	*3*	db_1	*1.8*	db_1	*2*
db_2	*1.5*	db_1	*2*	db_2	*1.2*	db_2	*2*
db_3	*0.4*	db_3	*1*	db_3	*0.4*	db_3	*1*

Table 1: The four database ranks for Example 3.1.

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 the definitions above. The first two columns of Table 1 correspond to rank $All_W(l)$ with threshold $l = 0.2$ (i.e., the user is willing to examine every document with similarity with q higher than 0.2). For example, the goodness of db_1 (Equation 1) is Goodness$(q, db_1) = 0.9 + 0.9 = 1.8$, because db_1 has two documents, d_1^1 and d_2^1, with similarity higher than 0.2. From Table 1, the $All_W(0.2)$ database rank is db_1, db_2, db_3.

Alternatively, the second two columns of Table 1 correspond to rank $All_D(0.2)$. For example, the goodness of db_1 (Equation 2) is Goodness$(q, db_1) = 1 + 1 = 2$, because there are two documents in db_1, d_1^1 and d_2^1, with similarity higher than 0.2. The $All_D(0.2)$ database rank is db_2, db_1, db_3, which is different from the $All_W(0.2)$ one: although db_1 contains documents whose vectors are closer to the query than those from db_2, db_2 contains more "acceptable" documents (i.e., documents with similarity greater than the given threshold).

To determine the $Top_W(k)$ and $Top_D(k)$ ranks, we sort the documents from all databases using their similarity with q:

$$(d_1^1, 0.9), (d_2^1, 0.9), (d_1^2, 0.8), (d_2^2, 0.4), (d_1^3, 0.4), (d_3^2, 0.3), (d_3^1, 0.1), (d_4^2, 0.1), (d_2^3, 0.1)$$

Suppose the user is willing to inspect the five documents that are closest to q ($k = 5$). Then db_1 contains two documents among the top five documents: $(d_1^1, 0.9)$ and $(d_2^1, 0.9)$. Using the $Top_W(5)$ Goodness metric (Equation 3), Goodness$(q, db_1) = 0.9 + 0.9 = 1.8$, and using the $Top_D(5)$ Goodness metric (Equation 4), Goodness$(q, db_1) = 2$. From Table 1, the $Top_W(5)$ and $Top_D(5)$ database ranks are db_1, db_2, db_3. (Actually, there is a tie between db_1 and db_2 in $Top_D(5)$.)

The definitions of a database's "goodness" try to quantify how useful the database is for the user that issued the query. The $All_W(l)$ and $All_D(l)$ ranks are for users who find useful any document that is close enough to the given query: they prefer to access first those databases containing either the highest number of useful documents ($All_D(l)$), or the highest "accumulated" similarity coming from useful documents ($All_W(l)$). Alternatively, the $Top_W(l)$ and $Top_D(l)$ ranks are for users who want to access the k documents that are closest to the given query: they prefer to access first those databases containing the highest number of such documents ($Top_D(l)$), or the highest "accumulated" similarity coming from top-k documents ($Top_W(l)$).

Of course, there are other meaningful ways in which we could define the "goodness" of a database for a query. However, in this paper we focus on the four definitions above, just to illustrate what the issues involved in choosing between databases are.

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.

The other extreme option would be for *gGlOSS* to not keep any information at all, and to rank databases randomly, for example: although the storage requirement of this approach is very low, it is not likely to be too useful, obviously.

In between these too extremes lie many reasonable solutions that 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 *doc* is a function of the number of times that t_j appears in *doc* and the number of documents in the database that contain t_j [Sal89]. Although the information that *gGlOSS* stores about each database is incomplete, it will prove useful to generate database ranks that resemble the ideal database ranks 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 [GGMT94a], 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 index of the database: although *gGlOSS* needs to keep weights associated with the words as well as document frequencies, so does a vector-space full-text index. Therefore, the *gGlOSS* information about a database remains at only an estimated 2% of the size of the corresponding full-text index.

To obtain the data that *gGlOSS* keeps about a database db_i, namely columns f_{i*} and w_{i*} of the F and W matrices above, database db_i will have to periodically extract this information from its local indexes and send it to the *gGlOSS* server. In the current implementation of the boolean *GlOSS* server, we provide the text databases that *GlOSS* indexes with a *collector* program. Each database runs this collector locally: the collector scans the local database's indexes, extracts the appropriate information, and sends it to the *GlOSS* server [6]. So far we provide a collector for WAIS [KM91] databases, and this collector already extracts the term-weight information that *gGlOSS* would need. (These weights are simply ignored by the current implementation of the boolean *GlOSS* server.)

[6] The *GlOSS* collector is described in `http://gloss.stanford.edu/collector.html`.

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:*

$$\text{computer} : (d_1, 0.8), (d_2, 0.7), (d_3, 0.9), (d_8, 0.9)$$

That is, document d_1 contains the word computer *with weight 0.8 (for some weight-computation algorithm [SM83]), 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 $Q = (q_1, \ldots, q_j, \ldots, q_t)$ [SM83], 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 vectors.

Since *gGlOSS* represents both the databases and the queries as vectors, *gGlOSS* can compute *similarities* between these vectors analogously to how *documents* and queries are compared. *gGlOSS* can 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 $w_{i*}.Q$, where $w_{i*} = (w_{i1}, \ldots, w_{it})$ is the column of W that corresponds to db_i. This estimate will be a special case of the more general estimates that we describe below. (See Section 4.2.) Also, note that the vectors with which *gGlOSS* represents each database can be viewed as *cluster centroids* [Sal89], where each database is considered as a single document cluster [7].

Because the information *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 two different sets of database ranks for a given query. These assumptions are unrealistic: 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 [GGMT94a, GGMT94b].

Although we will build the *gGlOSS* database ranks so they approximate the $All_W(l)$ and the $All_D(l)$ ideal ranks of Section 3, the *gGlOSS* ranks of the following sections could also be used to approximate the $Top_W(k)$ and $Top_D(k)$ ranks of Section 3.

4.1 High-correlation scenario

To derive $Max_W(l)$ and $Max_D(l)$, the first two database ranks with which *gGlOSS* tries to match the $All_W(l)$ and $All_D(l)$ database ranks 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.*

[7] 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*.

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 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. The goal of this (unrealistic) assumption is to simplify the computations that *gGlOSS* has to make to rank the databases.

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* with a given query, and what the added similarity of these documents is. These estimates determine the $Max_W(l)$ and $Max_D(l)$ database ranks.

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 with 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 with 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:

$$sim_p = \sum_{j=p,\ldots,n} q_j \times \frac{w_{ij}}{f_{ij}} > l, \text{ but} \quad (5)$$

$$sim_{p+1} = \sum_{j=p+1,\ldots,n} q_j \times \frac{w_{ij}}{f_{ij}} \leq l \quad (6)$$

Then, the f_{ip} documents having (at least) query words $t_p, \ldots, t_n$ have an estimated similarity with q greater than threshold l (Condition 5), 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 $\mathit{Estimate}(q, db_i)$, the estimated goodness of database db_i for query q, that determines the $\mathbf{Max_W(l)}$ database ranks:

$$\begin{aligned}\mathit{Estimate}(q, db_i) &= \sum_{j=1,\ldots,p} (f_{ij} - f_{i(j-1)}) \times sim_j \\ &= (\sum_{j=1,\ldots,p} q_j \times w_{ij}) + f_{ip} \times \sum_{j=p+1,\ldots,n} q_j \times \frac{w_{ij}}{f_{ij}} \end{aligned} \tag{7}$$

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.

Alternatively, for rank $\mathbf{Max_D(l)}$ we define:

$$\mathit{Estimate}(q, db_i) = f_{ip} \tag{8}$$

The first definition (Equation 7) computes the added similarity of the f_{ip} documents estimated to have similarity with q greater than threshold l (see Conditions 5 and 6, and Assumptions 4.1 and 4.2). The second definition (Equation 8) is how many such documents there are. The $Max_W(l)$ and $Max_D(l)$ database rank approximate the $All_W(l)$ and $All_D(l)$ database ranks of Section 3.

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 with 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 with 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 with 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 with q is even lower. Consequently, $p = 1$ (see Conditions 5 and 6).*

According to the $Max_W(0.2)$ definition of Estimate:

$$\begin{aligned}\mathit{Estimate}(q, db_i) &= f_{i1} \times (\frac{w_{i1}}{f_{i1}} + \frac{w_{i2}}{f_{i2}} + \frac{w_{i3}}{f_{i3}}) \\ &= 2 \times (\frac{0.45}{2} + \frac{0.2}{9} + \frac{0.9}{10}) \\ &= 0.674\end{aligned}$$

According to the $Max_D(0.2)$ definition of Estimate:

$$\mathit{Estimate}(q, db_i) = f_{i1} = 2$$

4.2 Disjoint scenario

To derive $Sum_W(l)$ and $Sum_D(l)$, two ranks that *gGlOSS* uses to approximate $All_W(l)$ and $All_D(l)$, *gGlOSS* assumes that if two words appear together in a user query, then these words do not appear together in any database document:

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 with q is exactly $q_j \times \frac{w_{ij}}{f_{ij}}$, if $f_{ij} > 0$ (from Assumption 4.2).

Therefore, for rank $\mathbf{Sum_W(l)}$ we define *Estimate*(q, db_i), the estimated goodness of database db_i for query q, as:

$$\begin{aligned} \mathit{Estimate}(q, db_i) &= \sum_{j=1,\ldots,n|(f_{ij}>0)\wedge(q_j \times \frac{w_{ij}}{f_{ij}}>l)} f_{ij} \times (q_j \times \frac{w_{ij}}{f_{ij}}) \\ &= \sum_{j=1,\ldots,n|(f_{ij}>0)\wedge(q_j \times \frac{w_{ij}}{f_{ij}}>l)} q_j \times w_{ij} \qquad (9) \end{aligned}$$

Alternatively, for rank $\mathbf{Sum_D(l)}$ we define *Estimate*(q, db_i) as:

$$\mathit{Estimate}(q, db_i) = \sum_{j=1,\ldots,n|(f_{ij}>0)\wedge(q_j \times \frac{w_{ij}}{f_{ij}}>l)} f_{ij} \qquad (10)$$

The first definition (Equation 9) computes the added similarity of the acceptable documents, while the second definition (Equation 10) computes the total number of such documents.

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* $Sum_W(0.2)$ *definition of Estimate:*

$$\mathit{Estimate}(q, db_i) = f_{i1} \times \frac{w_{i1}}{f_{i1}} = 0.45$$

According to the $Sum_D(0.2)$ *definition of Estimate:*

$$\mathit{Estimate}(q, db_i) = f_{i1} = 2$$

Notice the special case when the threshold l is zero. In this case, the $Max_W(0)$ and $Sum_W(0)$ definitions of *Estimate* (Equations 7 and 9) become:

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

assuming that if $f_{ij} = 0$, then $w_{ij} = 0$. Then, *Estimate*(q, db_i) becomes the inner product $Q.w_{i*}$. To compute the $Max_W(0)$ and $Sum_W(0)$ ranks, *gGlOSS* does not need the matrix F of document frequencies of the words; it only needs the matrix W of added weights. Similarly, when $l = 0$ the $Max_D(0)$ definition of *Estimate* (Equation 8) becomes:

$$Estimate(q, db_i) \quad = \quad f_{in}$$

and the $Sum_D(0)$ definition of *Estimate* (Equation 10) becomes:

$$Estimate(q, db_i) \quad = \quad \sum_{j=1,\ldots,n} f_{ij}$$

To compute the $Max_D(0)$ and $Sum_D(0)$ ranks, *gGlOSS* does not need the matrix W of added weights. Therefore, the storage requirements for *gGlOSS* to compute the database ranks are much lower if $l = 0$ (even when we are emulating ranks $All_W(l')$ and $All_D(l')$ with $l' > 0$, for example). We will then pay special attention to these ranks in our experiments of Section 6.2.

In the preceding discussion we just considered vector-space databases. However, *gGlOSS* can also deal with boolean databases by adapting the results of [GGMT94a, GGMT94b] to the framework presented in this paper.

In the following section we study how to compare the database ranks that *gGlOSS* generates against the ideal database ranks of Section 3.

5 Comparing database ranks

In this section we analyze how we can compare *gGlOSS*'s ranks (Section 4) to the ideal ones (Section 3) [8]. 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, according to one of the definitions of Section 3. 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:

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

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

[8] Our definition of the $\mathcal{R}_n$ metric in this section is partially based on work by Peter Schwarz and Laura Haas at IBM Almaden (personal communication, 1994).

I		*G*			*H*		
db	*Goodness*	*db*	*Estimate*	*Goodness*	*db*	*Estimate*	*Goodness*
db_1	0.9	db_2	0.8	0.4	db_2	0.9	0.4
db_2	0.4	db_1	0.6	0.9	db_1	0.8	0.9
db_3	0.3	db_3	0.3	0.3	db_3	0.4	0.3
db_4	0.2				db_5	0.2	0

Table 2: The ideal and *gGlOSS* database ranks for Example 5.1.

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 2 shows I, the ideal database rank, and G and H, two different gGlOSS database ranks for q, for some definition of these ranks (Sections 3 and 4). For example, db_1 is the top database in the ideal rank, with Goodness$(q, db_1) = 0.9$. Database db_5 does not appear in rank I, because Goodness$(q, db_5) = 0$. gGlOSS correctly predicted this for rank G (Estimate$(q, db_5) = 0$ for G), and so db_5 does not appear in G. However, db_5 does appear in H, because Estimate$(q, db_5) = 0.2$ for H.*

Let us focus on the G rank: db_2 is the top database in G, with Estimate$(q, db_2) = 0.8$. The real goodness of db_2 for q is Goodness$(q, db_2) = 0.4$. From the ranks of Table 2, $\mathcal{R}_1 = \frac{0.4}{0.9}$: if we access db_2, the top database from the G rank, we obtain Goodness$(q, db_2) = 0.4$, whereas the best database for q is db_1, with Goodness$(q, db_1) = 0.9$. 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$(q, db_4) = 0$), which is actually useful for q (Goodness$(q, db_4) = 0.2$).

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 a new metric, $\mathcal{P}_n$, that distinguishes 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 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:

$$\mathcal{P}_n = \frac{|\{db \in DB | db \text{ is a top-}n \text{ database in } G \text{ and } Goodness(q, db) > 0\}|}{|\{db \in DB | db \text{ is a top-}n \text{ database in } G\}|}$$

(Actually, $\mathcal{P}_n = 1$ if for all db, *Estimate*$(q, db) = 0$.) 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.*

We should notice that many other alternative metrics are possible for comparing database ranks. For example, we could consider the minimum number of swaps of pairs of databases that are required to obtain I from G (viewing I and G as permutations of DB). Alternatively, we could consider the top n databases in G, and compare them as a set against the top n databases in I, and report what fraction of the second set is in the first one (related to the *recall* metric of information-retrieval theory [SM83]), and what fraction of the first set is in the second set (related to the *precision* metric of information-retrieval theory) [GGMT94b]. In the following section, we only evaluate *gGlOSS* by using the $\mathcal{P}_n$ and $\mathcal{R}_n$ metrics, for different values of n, because we believe these metrics better reflect the users' expectations: if *gGlOSS* includes in its top-n database rank some database db that is not among the best n databases, *gGlOSS* should be given "partial credit" if db turns out to be useful for the query, even if db is not as useful as one of the best n databases. The metrics of [GGMT94a] and [GGMT94b] do not reflect this, unlike the $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics that we use in the following section.

6 Evaluating *gGlOSS*

In this section we evaluate *gGlOSS* experimentally, for different *gGlOSS* ranking algorithms (Section 4) and different ideal-ranking definitions (Section 3). We first describe the real-user queries and databases that we used in the experiments. Then, we report results for $Max_W(l)$, $Max_D(l)$, $Sum_W(l)$, and $Sum_D(l)$, the four *gGlOSS* ranking algorithms of Section 4, and for $All_W(l)$ and $All_D(l)$, two of the ideal ranking definitions of Section 3.

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 [YGM95] [9]. 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. Note that some of these newsgroups cover closely related topics, while others cover totally unrelated topics.

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 [Sal89]. 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.

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

For each query, ideal-rank definition, and *gGlOSS* ranking algorithm, we compared the ideal rank against the *gGlOSS* rank using the methodology of Section 5. For a fixed ideal-rank definition and a query, we generated the ideal rank of databases for that query: we evaluated the query at each of the 53 databases. For a fixed *gGlOSS* ranking definition and a query, we generated the rank of databases that *gGlOSS* would generate 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, what 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 $All_W(l)$, $All_D(l)$, $Sum_W(l)$, and $Sum_D(l)$ database ranks is better at predicting ideal ranks $All_W(l)$ and $All_D(l)$, 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 $All_W(0)$, $All_D(0)$, $Sum_W(0)$, and $Sum_D(0)$ (Section 4.2).

Figure 1 shows the values of the $\mathcal{R}_n$ metrics for ideal database rank $All_W(0)$, i.e., in this case any document with non-zero similarity with the query is considered useful. Ranks $Max_W(0)$ and $Sum_W(0)$ are identical to ideal rank $All_W(0)$, and so they have $\mathcal{R}_n = 1$ for all n, as the figure shows. Ranks $Max_D(0)$ and $Sum_D(0)$ also have high $\mathcal{R}_n$ values (0.91 and higher). All of the ranks have $\mathcal{P}_n = 1$, for all n, and we thus do not show the corresponding figure. (This is immediate for ranks $Max_W(0)$ and $Sum_W(0)$ because they coincide with the target rank $All_W(0)$. For ranks $Max_D(0)$ and $Sum_D(0)$, a given query q, and a database db, if $Estimate(q, db) > 0$, then $Goodness(q, db) > 0$. Consequently, $\mathcal{P}_n = 1$ for all n.) In summary, 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 ($All_W(0)$ rank), then *gGlOSS* should use the $Max_W(0)$ (or, identically, $Sum_W(0)$) ranks and get perfect results.

To study the impact of higher rank thresholds, Figures 2 and 3 show results for the $All_W(0.2)$ ideal rank. Surprisingly, in all of our experiments ranks $Max_W(l)$ and $Max_D(l)$ performed almost identically for $l > 0$. Although the *estimates* defining both ranks are different, they are based on the same assumptions, and so they produce similar database orderings. Analogously, ranks $Sum_W(l)$ and $Sum_D(l)$ performed almost identically for $l > 0$. Therefore, for clarity, we just present the data for $Max_W(l)$ and $Sum_W(l)$, labeled as $Max_W(l)/Max_D(l)$ and $Sum_W(l)/Sum_D(l)$, respectively, and we ignore $Max_D(l)$ and $Sum_D(l)$ in the rest of the discussion. Also, 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 $All_W(0.2)$. So, the results for high values of n are not that significant. Figure 3 shows that rank $Sum_W(0.2)$ has perfect $\mathcal{P}_n$ ($\mathcal{P}_n = 1$) for all n, because if a database db has *Estimate(q, db)*> 0 according to the $Sum_W(0.2)$ rank, then $Goodness(q, db) > 0$ according to $All_W(0.2)$. In other words, rank $Sum_W(0.2)$ only includes databases that are guaranteed to be useful. Rank $Max_W(0.2)$ may include databases not guaranteed to be useful, yielding higher $\mathcal{R}_n$ values (Figure 2), but lower $\mathcal{P}_n$ values (Figure 3).

To decide whether *gGlOSS* really needs to keep both matrices F and W (Section 4), we also use ranks $Max_W(0)$, $Max_D(0)$, $Sum_W(0)$, and $Sum_D(0)$ to approximate rank $All_W(0.2)$. *gGlOSS* needs only one of the two matrices to compute these four ranks (Section 4.2). Since ranks $Max_W(0)$ and $Sum_W(0)$ are always identical, and ranks $Max_D(0)$ and $Sum_D(0)$ performed very similarly in all of our experiments, we just present the data for $Max_W(0)$ and $Max_D(0)$, labeled $Max_W(0)/Sum_W(0)$ and $Max_D(0)/Sum_D(0)$, respectively. Figure 2 shows that the $Max_W(0)$ rank has the highest values of $\mathcal{R}_n$, followed by $Sum_W(0)$: these ranks assume a threshold $l = 0$, and thus they tend to include more databases in the ranks than their counterparts with threshold 0.2. This is also why $Max_W(0)$ and $Sum_W(0)$ have much lower $\mathcal{P}_n$ values (Figure 3) than $Max_W(0.2)$ and $Sum_W(0.2)$: they include more databases that have zero *Goodness* according to $All_W(0.2)$.

In summary, if the users are interested in not missing any useful database, but are willing to search some useless ones, then $Max_W(0)$ is the best choice for *gGlOSS*, and *gGlOSS* can do without matrix F. If the users wish to avoid searching useless databases, then $Sum_W(0.2)$ is the best choice. Unfortunately, $Sum_W(0.2)$ also has low $\mathcal{R}_n$ values, which means it can also miss some useful sources. As a compromise, a user can have $Max_W(0.2)$, which has much better $\mathcal{P}_n$ values than $Max_W(0)$ and generally better $\mathcal{R}_n$ values than $Sum_W(0.2)$. Also, note that in the special case where users are interested in accessing only one or two databases ($n = 1, 2$) then $Max_W(0.2)$ 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 4 and 5 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 $All_W(0.6)$ each query has on average only 0.29 useful databases, making the experimental results for such a threshold not that meaningful. 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 $Max_W(0)$ 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 $All_W(l)$ changes, and the static estimate provided by $Max_W(0)$ performs worse and worse. The $Max_W(l)$ and $Sum_W(l)$ ranks do take into account the target l values, and hence do substantially better. Also notice that our earlier conclusion still holds: strategy $Sum_W(l)$ is best at avoiding useless databases, while $Max_W(0)$ provides the best $\mathcal{R}_n$ values (at the cost of low $\mathcal{P}_n$ values).

Finally, for ideal rank $All_D(l)$, the *number of documents* in a database with similarity greater than l with the given query determines the goodness of the database. The performance of the *gGlOSS* ranks for $All_D(l)$ is very similar to that for $All_W(l)$. For space limitations, we only show how the $\mathcal{R}_n$ metric changes for different thresholds, for $n = 3$ (Figure 6). The $\mathcal{P}_n$ values are identical to those for ideal rank $All_W(l)$ (Figure 5): a database *db* has $Goodness(q, db) > 0$ for rank $All_W(l)$ if and only if *db* has $Goodness(q, db) > 0$ for rank $All_D(l)$, from the definition of these ranks (Section 4). However, unlike the $All_W(0)$ rank, the $All_D(0)$ rank is not matched exactly by any *gGlOSS* rank. (Recall that the $Max_W(0)$ and $Sum_W(0)$ ranks coincide with rank $All_W(0)$.) Thus, from the point of view of suggesting good sources, it does not matter if the users are looking for sites with the largest number of above-threshold documents ($All_D(l)$ rank), or if they are looking for sites with the largest total similarity with the given query of above-threshold documents ($All_W(l)$ rank).

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 interest, then *gGlOSS* should produce

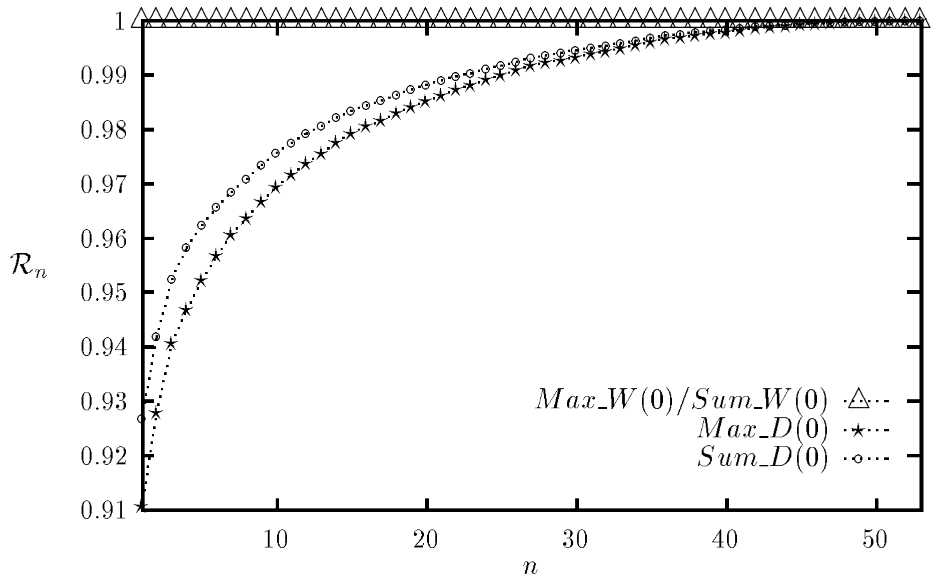

Figure 1: Parameter $\mathcal{R}_n$ as a function of n, the number of databases examined from the ranks, for the $All_W(0)$ ideal database ranking and the different *gGlOSS* rankings.

ranks based on the high-correlation assumption of Section 4.1: ranks $Max_W(l)$ and $Max_D(l)$ are the best candidates for ranks $All_W(l)$ and $All_D(l)$ with $l > 0$. If only high values of $\mathcal{R}_n$ are of interest, then *gGlOSS* can do without matrix F, and produce ranks $Max_W(0)$ or $Sum_W(0)$. 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: ranks $Sum_W(l)$ and $Sum_D(l)$ are the best candidates. For rank $All_W(0)$, ranks $Max_W(0)$ and $Sum_W(0)$ give perfect answers. Furthermore, *gGlOSS* does not need the F matrix to compute these ranks. For rank $All_D(0)$, ranks $Max_D(0)$ and $Sum_D(0)$ give the best approximation, and they do not require the W matrix.

7 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. The users will then query the *hGlOSS* 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, as Figure 7 illustrates.

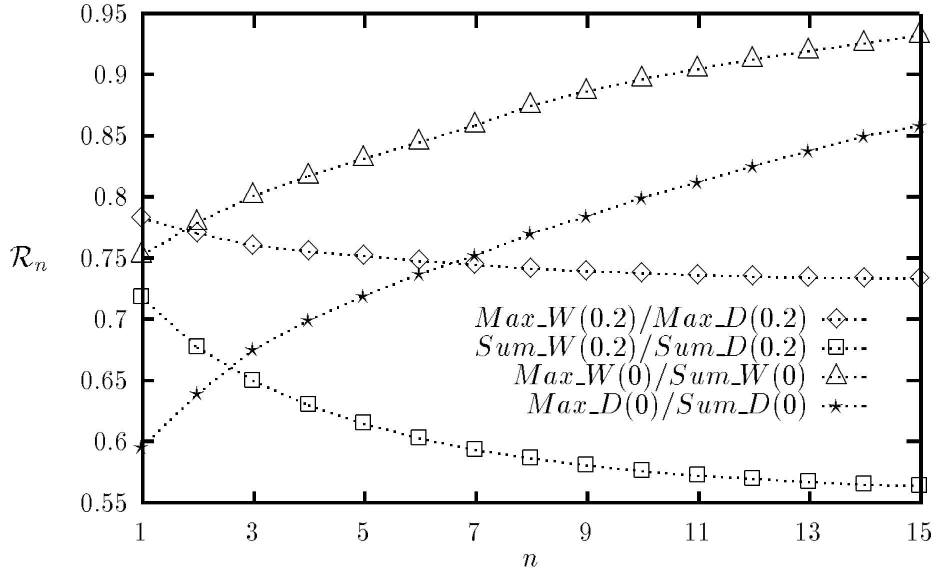

Figure 2: Parameter $\mathcal{R}_n$ as a function of n, the number of databases examined from the ranks, for the $All_W(0.2)$ ideal database ranking and the different *gGlOSS* rankings.

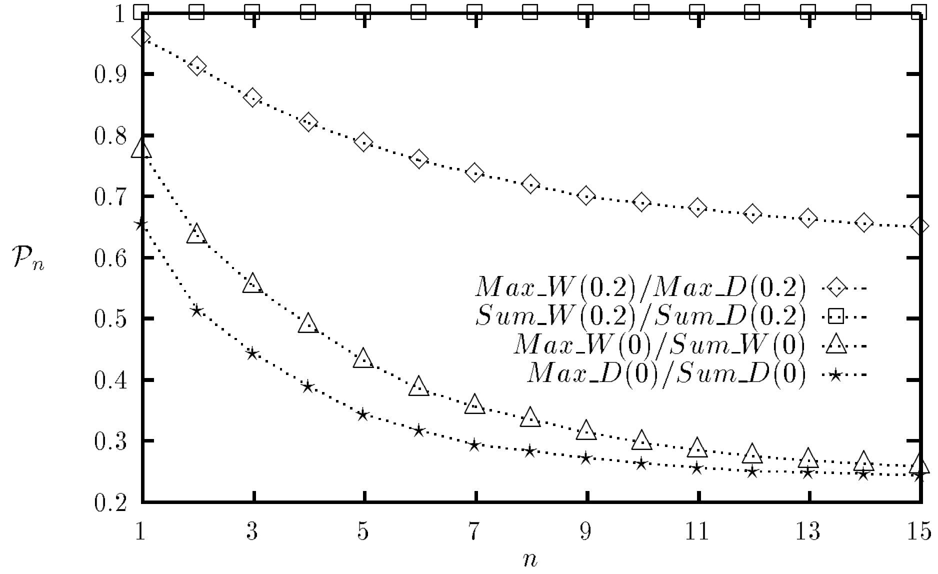

Figure 3: Parameter $\mathcal{P}_n$ as a function of n, the number of databases examined from the ranks, for the $All_W(0.2)$ ideal database ranking and the different *gGlOSS* rankings.

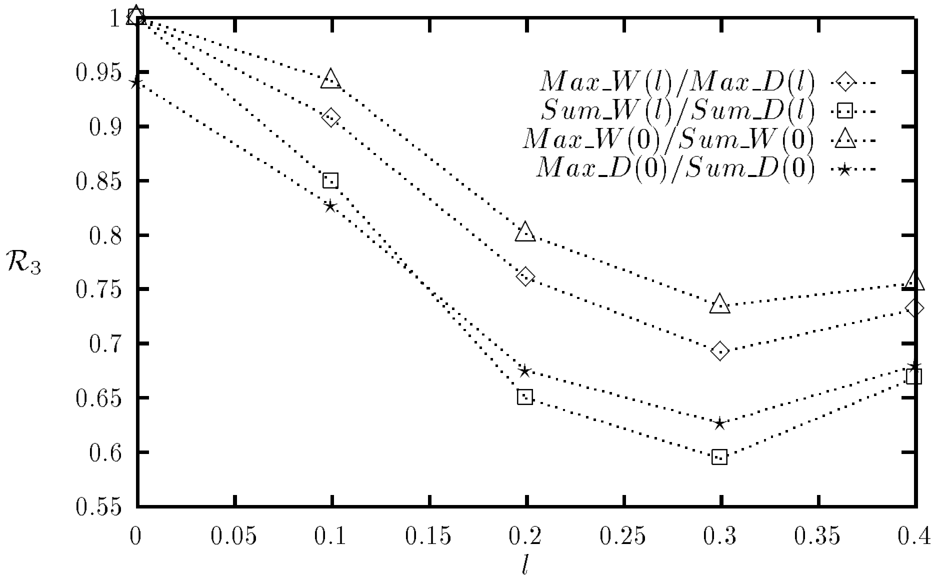

Figure 4: Parameter $\mathcal{R}_3$ as a function of the threshold l, for ideal rank $All_W(l)$.

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):

- H= (h_{rj}): h_{rj} is the number of *databases* in *gGlOSS* G_r that contain word t_j
- D= (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 7.1 *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:*

$$\text{computer} : (db_1, 5, 3.4), (db_2, 2, 2.1), (db_3, 1, 0.3)$$

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.*

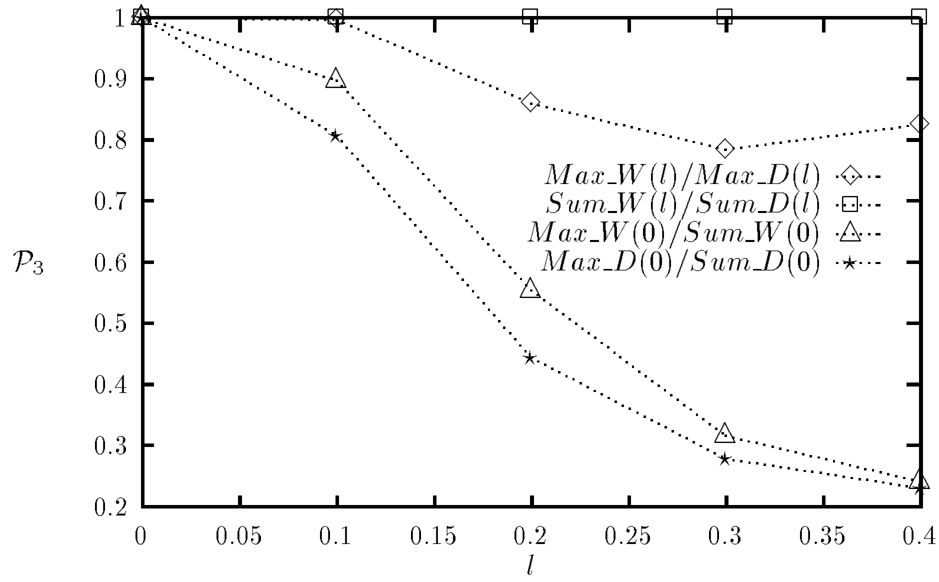

Figure 5: Parameter $\mathcal{P}_3$ as a function of the threshold l, for ideal rank $All_W(l)$.

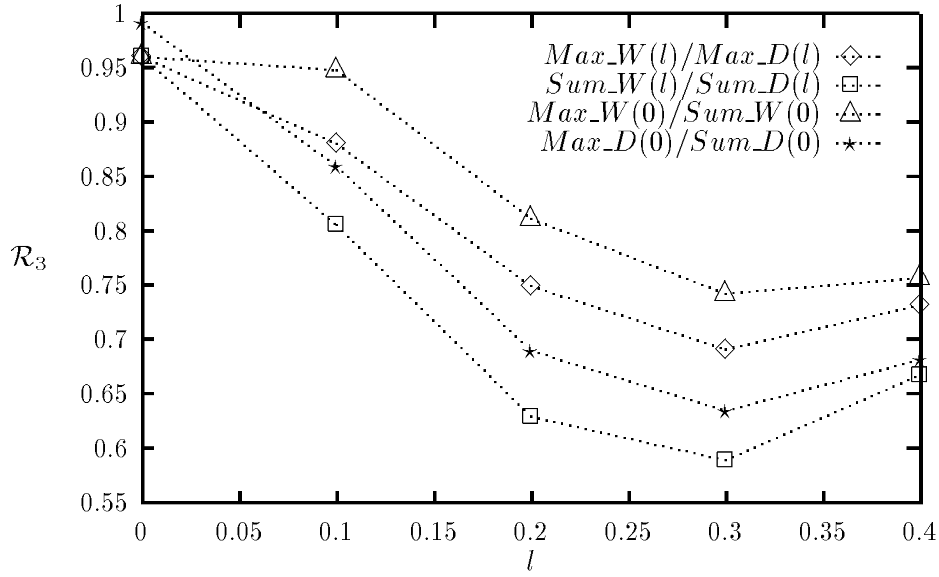

Figure 6: Parameter $\mathcal{R}_3$ as a function of the threshold l, for ideal rank $All_D(l)$.

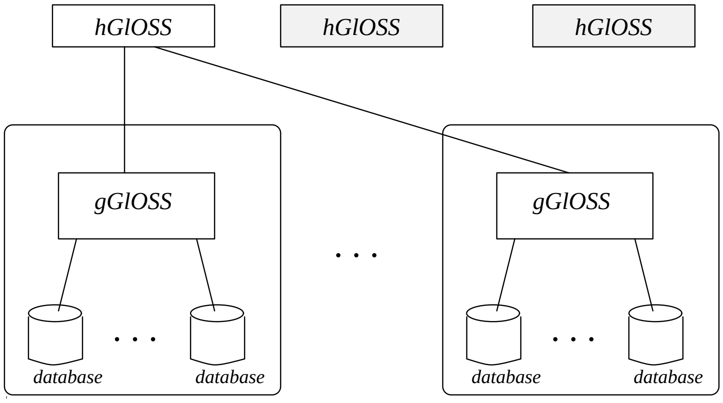

Figure 7: The *hGlOSS* server summarizes the contents of the traditional *gGlOSS* servers, who in turn summarize the contents of the underlying databases.

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 $All_D(0)$ with the $Max_D(0)$ database rank. Therefore, the *gGlOSS* servers only keep the document-count vector F (Section 4). Next, we define the goodness of a *gGlOSS* server for a query. This goodness determines the ideal rank of *gGlOSS* servers in a way analogous to how we defined the ideal database rank for a given query (Section 3). 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 is analogous to the goodness definition associated with ideal rank $All_D(0)$.) Finally, we define how *hGlOSS* approximates the ideal rank of *gGlOSS* servers. *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}. (This is analogous to the *Estimate* definition associated with rank $Max_D(0)$ in Section 4.2.) In other words, *hGlOSS* estimates that there are h_{rn} databases in G_r that have a non-zero goodness estimate for q.

Table 3 shows the different values of the (adapted) $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics for the 6,800 queries of

n	$\mathcal{R}_n$	$\mathcal{P}_n$
1	0.985217	1
2	0.990884	1
3	0.994409	1
4	0.997599	1
5	1.000000	1

Table 3: The $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics for *hGlOSS* and our sample experiment.

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

8 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.

References

[BC92] Daniel Barbará and Chris Clifton. Information Brokers: Sharing knowledge in a heterogeneous distributed system. Technical Report MITL-TR-31-92, Matsushita Information Technology Laboratory, October 1992.

[BDB+92] C. Mic Bowman, Chanda Dharap, Mrinal Baruah, Bill Camargo, and Sunil Potti. A file system for information management. In *Proceedings of the 1992 Conference on Intelligent Information Management Systems*, June 1992.

[BDH+94] 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.

[BLCGP92] 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.

[DANO91] Peter B. Danzig, Jongsuk Ahn, John Noll, and Katia Obraczka. Distributed indexing: a scalable mechanism for distributed information retrieval. In *Proceedings of the* 14^{th} *Annual SIGIR Conference*, October 1991.

[DLO92] Peter B. Danzig, Shih-Hao Li, and Katia Obraczka. Distributed indexing of autonomous internet services. *Computer Systems*, 5(4), 1992.

[DS94] Andrzej Duda and Mark A. Sheldon. Content routing in a network of WAIS servers. In *14th IEEE International Conference on Distributed Computing Systems*, 1994.

[Fos92] Steve Foster. About the Veronica service, November 1992. Message posted in `comp.infosystems.gopher`.

[FY93] David W. Flater and Yelena Yesha. An information retrieval system for network resources. In *Proceedings of the International Workshop on Next Generation Information Technologies and Systems*, June 1993.

[GGMT94a] 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. Also available by anonymous `ftp` from `db.stanford.edu` in `/pub/gravano/1994/stan.cs.tn.93.002.sigmod94.ps`.

[GGMT94b] 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. Also available by anonymous `ftp` from `db.stanford.edu` in `/pub/gravano/1994/stan.cs.tn.94.010.pdis94.ps`.

[KM91] Brewster Kahle and Art Medlar. An information system for corporate users: Wide Area Information Servers. Technical Report TMC199, Thinking Machines Corporation, April 1991.

[Neu92] B. Clifford Neuman. The Prospero File System: A global file system based on the Virtual System model. *Computer Systems*, 5(4), 1992.

[ODL93] Katia Obraczka, Peter B. Danzig, and Shih-Hao Li. Internet resource discovery services. *IEEE Computer*, September 1993.

[OM92] Joann J. Ordille and Barton P. Miller. Distributed active catalogs and meta-data caching in descriptive name services. Technical Report #1118, University of Wisconsin-Madison, November 1992.

[SA89] Patricia Simpson and Rafael Alonso. Querying a network of autonomous databases. Technical Report CS-TR-202-89, Dept. of Computer Science, Princeton University, January 1989.

[Sal89] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison Wesley, 1989.

[Sch90] Michael F. Schwartz. A scalable, non-hierarchical resource discovery mechanism based on probabilistic protocols. Technical Report CU-CS-474-90, Dept. of Computer Science, University of Colorado at Boulder, June 1990.

[Sch93] Michael F. Schwartz. Internet resource discovery at the University of Colorado. *IEEE Computer*, September 1993.

[SDW$^+$94] Mark A. Sheldon, Andrzej Duda, Ron Weiss, James W. O'Toole, and David K. Gifford. A content routing system for distributed information servers. In *Proc. Fourth International Conference on Extending Database Technology*, 1994.

[SEKN92] Michael F. Schwartz, Alan Emtage, Brewster Kahle, and B. Clifford Neuman. A comparison of internet resource discovery approaches. *Computer Systems*, 5(4), 1992.

[SM83] Gerard Salton and Michael J. McGill. *Introduction to modern information retrieval.* McGraw-Hill, 1983.

[WS93] Chris Weider and Simon Spero. Architecture of the WHOIS++ Index Service, October 1993. Working draft.

[YGM95] 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.