GlOSS: Text-Source Discovery over the Internet

Luis Gravano
Columbia University
and
Héctor García-Molina
Stanford University
and
Anthony Tomasic
INRIA Rocquencourt

The dramatic growth of the Internet has created a new problem for users: the location of relevant sources of documents. This article presents a framework for (and experimentally analyzes a solution to) this problem, which we call the *text-source discovery problem*. Our approach consists of two phases. First, each text source exports its contents to a centralized service. Then, users present queries to the service, which returns an ordered list of promising text sources. This article describes *GlOSS* – Glossary of Servers Server –, with two versions: *bGlOSS*, which provides a Boolean query retrieval model, and *vGlOSS*, which provides a vector-space retrieval model. We also present *hGlOSS*, which provides a decentralized version of the system. We extensively describe the methodology for measuring the retrieval effectiveness of these systems and provide experimental evidence, based on actual data, that all three systems are highly effective at determining promising text sources for a given query.

Categories and Subject Descriptors: H.3 [**Information Systems**]: Information Storage and Retrieval

General Terms: Performance, Measurement

Additional Key Words and Phrases: Internet search and retrieval, digital libraries, text databases, distributed information retrieval

Name: Luis Gravano
Address: Computer Science Department, Columbia University, 1214 Amsterdam Avenue, New York, NY 10027, USA; email: gravano@cs.columbia.edu
Name: Héctor García-Molina
Affiliation: Computer Science Department, Stanford University, USA
Address: hector@cs.stanford.edu
Name: Anthony Tomasic
Affiliation: INRIA Rocquencourt, France
Address: Anthony.Tomasic@inria.fr

1. INTRODUCTION

The Internet has grown dramatically over the past few years. Document sources are available everywhere, both within the internal networks of organizations and on the Internet. This growth represents an incredible wealth of information. Our goal is to help an end user find documents of interest across potential sources on the Internet.

There are a number of options for searching over a large and distributed collection of documents, each with its own strengths and weaknesses. Solutions fall into two broad categories: single versus distributed search engines. A single search engine builds a full index of the entire collection, by scanning all documents. Some systems (e.g., Web search engines) discard the documents and only retain the index with pointers to the original documents; other systems warehouse the documents themselves, providing users with access to both the index and the documents (e.g., Dialog, Mead Data). The index may be partitioned by topic or sub-collection, but is managed by a single search engine.

The second option is to index documents through multiple engines, each run by the organization owning each source of documents. A global search is managed by a *meta-searcher* that interacts with the individual source engines. One alternative for meta-searching is to send a user query to all engines, and collect the results (e.g., MetaCrawler [Selberg and Etzioni 1995]). The user can then be directed to sites that have matching documents or to particular documents at those sites.

Another option for the multiple source scenario, one we explore in depth in this paper, is to obtain from the engines in advance metadata that can guide queries to sources that have many matching documents. This requires the cooperation of the engines, i.e., they must export metadata describing their collection. When the meta-searcher receives a user query, it consults its collected metadata and suggests to the user sources to try. This solution may not be as accurate as submitting the query to all sources, since the suggestions are only based on collection metadata. However, the query overhead is much less since queries are not executed everywhere. We call the problem of identifying document sources based on exported metadata the *text-source discovery problem.*

In this paper we focus on the multiple-engine scenario, and study solutions to the text-source discovery problem. We call our family of solutions *GlOSS*, for *Glossary-of-Servers Server.* In particular *GlOSS* meta-searchers use statistical metadata, i.e., how many times each term occurs at each source. As we will show, these "summaries" are small relative to the collection, and because they only contain statistics, will be much easier to export by a source. Statistical summaries can be obtained mechanically, and hence are superior to manually produced summaries that are often out of date. Similarly, since they summarize the entire collection, they are better than summaries based on a single field (such as titles). As we will see, *GlOSS* works best with a large collection of *heterogeneous* data sources. That is, the subject areas covered by the different data sources are very distinct from each other. In this case, the statistical summaries used by *GlOSS* strongly discriminate each source from the others.

It is important to note that in this paper we do not compare the single and multiple engine scenarios. First, in many cases one is not given a choice. For example, the documents may be owned by competing organizations that do not wish to export their full collections. On the Web, for instance, growing numbers of documents are only available through search interfaces, and hence unavailable to the crawlers that feed search engines. Second, if we do have a choice, the factors to consider are very diverse: copyright issues regarding the indexing or warehousing of documents, the cost and scalability (storage, operations) of maintaining a single index, the frequency at which new documents are indexed, and the accuracy of the results obtained. Instead, we only consider a multiple-engine scenario, and study *GlOSS* solutions to the text-discovery problem. We compare the "accuracy" of these solutions to what could be obtained by sending a query to all underlying search engines.

Also note that in this paper we do not study how a user submits queries to the individual sources. That is, once *GlOSS* suggests sources, the user must submit the query there. The user or some translation service must express the query using the particular syntax and operators used by a source. Similarly, the user may wish to combine and rank the results obtained at different sources. These are hard problems that are addressed in other papers [Chang et al. 1996; Gravano et al. 1997; Gravano and García-Molina 1997].

In summary, the contributions of this paper are as follows:

—We present a version of *GlOSS* (*vGlOSS*) that works with vector-space search engines [Salton 1989; Salton and McGill 1983]. (These engines treat both the documents and the queries themselves as weight vectors.)

—We describe a text-source discovery service for Boolean engines, *bGlOSS*. These engines, while not as sophisticated, are still widely used.

—We define metrics for evaluating text-source discovery services.

—We experimentally evaluate *vGlOSS* and *bGlOSS*, using real document databases. We note that even though discovery schemes for Internet sources have been proposed and implemented by others, it is rare to find an experimental evaluation like ours that carefully compares the various options.

—We analyze the *GlOSS* storage requirements, showing that a *GlOSS* index is significantly smaller than a full conventional index. We also discuss ways to further reduce storage needs.

—We briefly describe how *GlOSS* services can form a hierarchy. In such a case, services that only index a fraction of the sources can be accessed by a higher level *GlOSS* service.

We start in Sections 2 and 3 by presenting and evaluating our *vGlOSS* and *bGlOSS* services. In Section 4 we discuss storage requirements, hierarchical discovery schemes, and other issues. Finally, in Section 5 we briefly survey related techniques, some of which could work in conjunction with *GlOSS*.

2. CHOOSING VECTOR-SPACE DATABASES

In this section we present *vGlOSS*, a text-source discovery service that deals with vector-space databases and queries [Gravano and García-Molina 1995a].

2.1 Overview of the Vector-Space Retrieval Model

Under the vector-space model, documents and queries are conceptually represented as vectors [Salton 1989]. If m distinct words are available for content identification, a document d is represented as a normalized m-dimensional vector, $D = \langle w_1, \ldots, w_m \rangle$, where w_j is the "weight" assigned to the j-th word t_j. If t_j is not present in d, then w_j is 0. For example, the document with vector $D_1 = \langle 0.5, 0, 0.3, \ldots, \rangle$ contains the first word in the vocabulary (say, by alphabetical order) with weight 0.5, does not contain the second word, and so on.

The weight for a document word indicates how statistically important it is. One common way to compute D is to first obtain an un-normalized vector $D' = \langle w'_1, \ldots, w'_m \rangle$, where each w'_i is the product of a word frequency (*tf*) factor and an inverse document frequency (*idf*) factor. The *tf* factor is equal (or proportional) to the frequency of the i^{th} word within the document. The *idf* factor corresponds to the content discriminating power of the i^{th} word: a word that appears rarely in documents has a high *idf*, while a word that occurs in a large number of documents has a low *idf*. Typically, *idf* is computed by $\log \frac{n}{d_i}$, where n is the total number of documents in the collection, and d_i is the number of documents having the i^{th} word. (If a word appears in every document, its discriminating power is 0. If a word appears in a single document, its discriminating power is as large as possible.) Once D' is computed, the normalized vector D is typically obtained by dividing each w'_i term by $\sqrt{\sum_{i=1}^{m}(w'_i)^2}$.

Queries in the vector-space model are also represented as normalized vectors over the word space, $Q = \langle q_1, \ldots, q_m \rangle$, where each entry indicates the importance of the word in the search. Often queries are written by a user in natural language. In this case, q_j is typically a function of the number of times word t_j appears in the query string times the *idf* factor for the word. The similarity between a query q and a document d, $sim(q, d)$, is defined as the inner product of the query vector Q and the document vector D. That is,

$$sim(q, d) = Q \cdot D = \sum_{j=1}^{m} q_j \cdot w_j$$

Notice that similarity values range between zero and one, inclusive, because Q and D are normalized.

Ideally, a user would like to find documents with the highest similarity to some query. It is important to notice that similarity is always relative to some collection. That is, the same document may be given different vectors by two different search engines, due to the different *idf* factors used. Thus, one engine may judge the document relevant to a query, while the second one may not.

2.2 Evaluating Databases

Given a query, we would like to rank the available vector-space databases according to their "usefulness," or *goodness* for the query. In this section we present one possible definition of goodness, with its associated *ideal database rank*. (The next section explores how *vGlOSS* will try to rank the databases as closely as possible to this ideal rank.) The goodness of a database depends on the number of documents in the database that are reasonably similar to the given query, and on their actual similarity to the query. The best databases are those with many documents that are highly similar to the query in hand. However, a database might also have a high goodness value if it holds a few documents with very high similarity, or many documents with intermediate similarity to the query.

Our goodness definition 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* of the documents to the end user who submitted the query. (The effectiveness of information retrieval searching is based on subjective relevance assessments [Salton and McGill 1983].) Using relevance would be appropriate for evaluating the search engines at each database; instead, we are evaluating how well *vGlOSS* can predict the answers that the databases return. In Section 2.6 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 section 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 user-provided threshold l. Documents with lower similarity are unlikely to be useful, and therefore we ignore them. Thus, we define:

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

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 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:

$$db_1 \ : \ (d_1^1, 0.9), (d_2^1, 0.9), (d_3^1, 0.1)$$
$$db_2 \ : \ (d_1^2, 0.8), (d_2^2, 0.4), (d_3^2, 0.3), (d_4^2, 0.1)$$

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 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. As mentioned earlier, these similarities can depend on the characteristics of the collection that contains the document, and may not be "globally valid." For example, if a database db_1 specializes in computer science, the word *databases* might appear in many of its documents, and its *idf* factor will be low. The word *databases*, on the other hand, may have a high *idf* factor in a database db_2 that is totally unrelated to computer science and contains very few documents 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 2.6 we further discuss this problem, together with alternative ways of defining *Goodness*.

2.3 Ranking Databases

vGlOSS ranks the databases according to their potential usefulness for a given query. The goal is to approximate the $Ideal(l)$ database rank as closely as possible, for which *vGlOSS* should know the number of documents in each database with similarity to the query greater than l, and add their similarities (Section 2.2). To perform this task, *vGlOSS* keeps information about the available databases. One option would be for *vGlOSS* to keep complete information on each database: for each database db and word t, *vGlOSS* would know what documents in db contain t, what weight t has in each of them, and so on. Although *vGlOSS*'s answers would always match the $Ideal(l)$ ranks (if this information is kept up to date), the storage requirements of such an approach would be too high: *vGlOSS* needs to index many databases, and keeping so much information on each of them does not scale. Furthermore, this information might not be available for commercial databases, for example.

More reasonable solutions keep incomplete yet useful information on the databases. In this section we explore some options for *vGlOSS* 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, *vGlOSS* needs (at most) two numbers. This partial information will prove useful for estimating database ranks that resemble the ideal one, as we will see in Section 2.5.2. Furthermore, this information is orders of magnitude smaller than that required by a full-text index of the databases (Section 4.1).

To obtain the f_{i*} and w_{i*} values for database db_i, *vGlOSS* may have to periodically run a *collector* program that extracts this information from the local indexes and sends it to the *vGlOSS* server. An alternative architecture uses the *STARTS* protocol [Gravano et al. 1997] to export summaries from the source to the server. *STARTS* is an emerging protocol proposal for Internet searching coordinated by Stanford, and that involved over 10 companies and organizations. *STARTS* specifies that sources should export content summaries that closely resemble the *vGlOSS* summaries.

EXAMPLE 2.**:** 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, document d_2, with weight 0.7, and so on. Database *db* will not export all this information to *vGlOSS*: it will only tell *vGlOSS* 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 $0.8 + 0.7 + 0.9 + 0.9 = 3.3$. ∎

vGlOSS could compare a query q and a database db_i analogously to how queries and documents are compared. That is, it could treat db_i as a "document" with vector $D = \langle w_{i1}, \ldots, w_{im} \rangle$, normalize the vector, and then compute $sim(q, db_i)$. 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 *vGlOSS* ranks below reflect this fact.

Also, note that the vectors with which *vGlOSS* represents each database can be viewed as *cluster centroids* [Salton 1989] used in information retrieval techniques. In these techniques, a cluster centroid is a vector that represents a collection of documents that are "near" each other according to some clustering algorithm. When an information retrieval engine processes a query, it compares the query against the cluster centroids using a similarity function, and retrieves the documents in the clusters with matching centroids. Thus, *GlOSS* can be viewed as one such system where each database is considered as a single document cluster represented by a centroid.

As mentioned above, *vGlOSS* estimates the number of documents with similarity to a given query greater than a threshold l, and their added similarity. Because the information that *vGlOSS* 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 *vGlOSS* to compute database ranks that approximate the $Ideal(l)$ rank. The following sections present $Max(l)$ and $Sum(l)$, two such database ranks based on different underlying keyword distribution assumptions. $Max(l)$ assumes that query keywords occur together in the database documents, while $Sum(l)$ is at the other end of the spectrum, and assumes that query keywords do not occur together in the database documents.

2.3.1 *High-Correlation Scenario.* To derive $Max(l)$, the first database rank with which *vGlOSS* estimates the $Ideal(l)$ database rank of Section 2.2, *vGlOSS* 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 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.*

Because this assumption is unrealistic, in Section 2.3.2 we introduce an alternative assumption that can be regarded as the opposite of Assumption 1. In Section 2.5 we compare experimentally these two computationally tractable extreme cases, and we analyze the circumstances under which one outperforms the other.

EXAMPLE 3.: 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*.

vGlOSS 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*. ∎

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

ASSUMPTION 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 *vGlOSS* has to make to rank the databases.

EXAMPLE 3.: **(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 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. ∎

vGlOSS uses the assumptions above to estimate how many documents in a database have similarity greater than some threshold l to a given query, and their added similarity. 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 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 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 *vGlOSS* should explore the $f_{i2} - f_{i1}$ documents (Assumption 1) that contain words $t_2, \ldots, t_n$, but not word t_1. Thus, *vGlOSS* finds p such that:

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

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

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:

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

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 1 and 2.)

Example 3.: **(cont.)** Assume that query q has weight 1 for each of its three words. According to Assumption 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 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*$(0.2, q, db_i) = f_{i1} \times (q_1 \times \frac{w_{i1}}{f_{i1}} + q_2 \times \frac{w_{i2}}{f_{i2}} + q_3 \times \frac{w_{i3}}{f_{i3}}) = 2 \times (1 \times \frac{0.45}{2} + 1 \times \frac{0.2}{9} + 1 \times \frac{0.9}{10}) = 0.674$. ∎

2.3.2 *Disjoint Scenario.* The *Max*(l) rank that *vGlOSS* uses to approximate *Ideal*(l) assumes that query keywords tend to appear together in database documents. We now present *Sum*(l), a new database rank built upon the "opposite" assumption, namely that if two words appear together in a user query, then these words do not appear together in any database document (if possible):

Assumption 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. *vGlOSS* also needs to make Assumption 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 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 2).

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

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

Example 4.: Consider the data of Example 3. According to Assumption 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 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*, $Estimate(0.2, q, db_i) = f_{i1} \times \frac{w_{i1}}{f_{i1}} = 0.45$. ∎

In general, the $Max(l)$ estimate for a database and a query is always greater than or equal to the corresponding $Sum(l)$ estimate. ($Sum(l)$ makes "pessimistic" assumptions on the distribution of the query keywords across the database documents.) However, in the special case when the threshold l is zero, the $Max(0)$ and $Sum(0)$ definitions of *Estimate* (Equations 4 and 5) become the same:

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

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, *vGlOSS* does not need the matrix F of document frequencies of the words; it only needs the matrix W of added weights. [1] Therefore, the storage requirements for *vGlOSS* to compute the database ranks may be much lower if $l = 0$. We pay special attention to these ranks in our experiments of Section 2.5.2.

2.4 Comparing Database Ranks

In this section we analyze how we can compare *vGlOSS*'s ranks (Section 2.3) to the ideal one (Section 2.2). 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 *vGlOSS* generated for q, using one of the schemes of Section 2.3. 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 $I = (db_{i_1}, \ldots, db_{i_{s''}})$

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

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. The database ranks produced by *vGlOSS* are incremental "plans" for evaluating a query. In effect, we first contact the top database in the rank. If we are not satisfied with the answers retrieved, we contact the second database, and so on. Thus, we consider the top n databases in rank I, and compute i_n, the accumulated goodness (in rank I) of these n databases for query q. We then consider the top n databases in rank G, and compute g_n, the accumulated goodness of these n databases for q. The computation of both i_n and g_n implicitly assumes that databases are disjoint, so that the goodness contribution of a database does not depend on what databases appear higher in the rank.

Because rank I was generated using the actual goodness 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. Because *vGlOSS* 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 metric is thus related to the *recall* metric used in the information retrieval community [Salton 1989]: $\mathcal{R}_n$ is a measure of how much of the available goodness in the n best databases (as determined by I) is accumulated in the first n databases in the *vGlOSS* rank G. In other words, $\mathcal{R}_n$ models what the user that searches the top n databases that *vGlOSS* suggests would get, compared to what the user would have gotten by searching the top n databases in the ideal rank.

Example 5.**:** Consider a query q, and five databases db_i, $1 \leq i \leq 5$. Figure 1 shows I, the ideal database rank, and G and H, two different *vGlOSS* database ranks for q, for some definition of these ranks. For example, db_1 is the top database in the ideal rank, with $\mathit{Goodness}(l, q, db_1) = 0.9$. Database db_5 does not appear in rank I, because $\mathit{Goodness}(l, q, db_5) = 0$. *vGlOSS* correctly predicted this for rank G ($\mathit{Estimate}(l, q, db_5) = 0$ for G), and so db_5 does not appear in G. However, db_5 does appear in H, because $\mathit{Estimate}(l, q, db_5) = 0.2$ for H.

Let us focus on the G rank: db_2 is the top database in G, with $\mathit{Estimate}(l, q, db_2) = 0.8$. The real goodness of db_2 for q is $\mathit{Goodness}(l, q, db_2) = 0.4$. From the ranks of Figure 1, $\mathcal{R}_1 = \frac{0.4}{0.9}$: if we access db_2, the top database from the G rank, we obtain $\mathit{Goodness}(l, q, db_2) = 0.4$, whereas the best database for q is db_1, with $\mathit{Goodness}(l, q, db_1) = 0.9$. 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 ($\mathit{Estimate}(l, q, db_4) = 0$), which is actually useful for q ($\mathit{Goodness}(l, q, db_4) = 0.2$).

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

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

Fig. 1. The ideal and *vGlOSS* database ranks for Example 5.

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 *vGlOSS* ranks that include useless databases and those that do not. Given a *vGlOSS* 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, $Estimate(l, q, db) = 0$.) $\mathcal{P}_n$ is related to the *precision* metric used in the information retrieval community, and measures the fraction of the first n databases in the *vGlOSS* rank G with non-zero goodness. Note that $\mathcal{P}_n$ is independent of the ideal database rank I: it just depends on how many databases that *vGlOSS* estimated as potentially useful turned out to actually be useful for the query. A ranking with higher $\mathcal{P}_n$ is better because it leads to fewer fruitless database searches.

Example 5.: **(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. ∎

The metrics that we introduced in this section focus on the goodness of the databases, and do not examine if the *same* databases are present both in the ideal database ranks and in the *vGlOSS* ranks. In [Gravano et al. 1994a; Gravano et al. 1994b] we use different metrics that focus on how well a Boolean version of *GlOSS* (Section 3) identifies the actual *best* databases for a query.

2.5 Evaluating *vGlOSS*

In this section we evaluate different *vGlOSS* 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 *vGlOSS* ranks of Section 2.3.

2.5.1 *Queries and Databases.* To evaluate *vGlOSS* experimentally, we used real-user queries and databases. The queries that we used were profiles that real users submitted to the SIFT Netnews server developed at Stanford [Yan and García-Molina 1995]. 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 6,800 vector-space profiles that were active on the server in December 1994. These queries have an average of 2.75 words each, for a total of 7,627 unique words.

To evaluate the *vGlOSS* performance using these 6,800 queries, we used 53 newsgroups as 53 databases: we took a snapshot of the articles that were active at the Stanford Computer Science Department Netnews 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 6,800 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 [Salton 1989]. 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 *vGlOSS* ranking algorithm we compared the ideal rank against the *vGlOSS* rank using the methodology of Section 2.4. We evaluated each query at each of the 53 databases to generate its ideal database rank. For a fixed *vGlOSS* ranking definition and a query, we computed the rank of databases that *vGlOSS* would produce for that query: we extracted the (partial) information that *vGlOSS* needs from each of the 53 databases. For each query word, *vGlOSS* 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 *vGlOSS* 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 *vGlOSS* needs, and export it to the *vGlOSS* server. (See Section 2.3.)

2.5.2 *Experimental Results.* In this section we experimentally compare the *vGlOSS* 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 *vGlOSS*. We also investigate whether keeping both the F and W matrices of Section 2.3 is really necessary, since *vGlOSS* needs only one of these matrices to compute ranks $Max(0)$ and $Sum(0)$ (Section 2.3.2).

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, *vGlOSS* should use the $Max(0)$ (or, identically, $Sum(0)$) ranks and get perfect results.

To study the impact of higher rank thresholds, Figures 2 and 3 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 3 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 2), but lower $\mathcal{P}_n$ values (Figure 3).

To decide whether *vGlOSS* really needs to keep both matrices F and W (Section 2.3), we also use ranks *Max*(0) and *Sum*(0) to approximate rank *Ideal*(0.2). *vGlOSS* needs only one of the two matrices to compute these ranks (Section 2.3.2). Since ranks *Max*(0) and *Sum*(0) are always identical, we just present their data once labeled *Max*(0)/*Sum*(0). Figure 2 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 3) 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 *vGlOSS*, and *vGlOSS* 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 *vGlOSS* 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 *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, *vGlOSS* generally predicts fairly well the best databases for a given query. Actually, the more *vGlOSS* knows about the users' expectations, the better *vGlOSS* can rank the databases for the query. If high values of both $\mathcal{R}_n$ and $\mathcal{P}_n$ are of interest, then *vGlOSS* should produce ranks based on the high-correlation assumption of Section 2.3.1: 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 *vGlOSS* can do without matrix

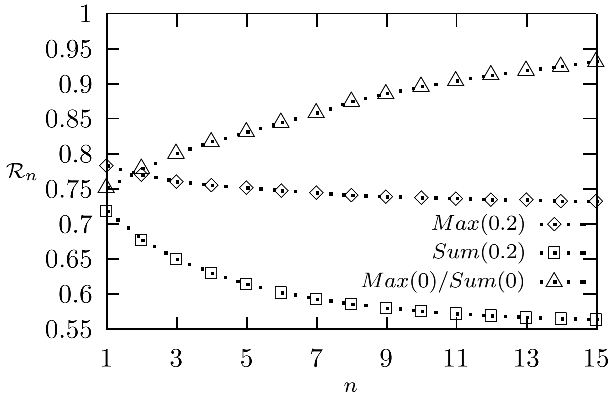

Fig. 2. Parameter $\mathcal{R}_n$ as a function of n, the number of databases examined from the ranks, for the *Ideal*(0.2) ideal database ranking and the different *vGlOSS* rankings.

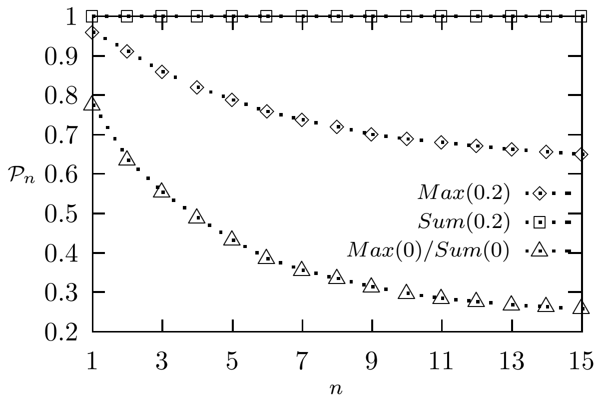

Fig. 3. Parameter $\mathcal{P}_n$ as a function of n, the number of databases examined from the ranks, for the *Ideal*(0.2) ideal database ranking and the different *vGlOSS* rankings.

F, and produce ranks *Max*(0) or *Sum*(0). If only high values of $\mathcal{P}_n$ are of interest, then *vGlOSS* should produce ranks based on the disjoint-scenario assumption of Section 2.3.2: rank *Sum*(l) is the best candidate. For rank *Ideal*(0), ranks *Max*(0) and *Sum*(0) give perfect answers.

2.6 Alternative Ideal Ranks

Section 2.2 presented a way of defining the goodness of a database for a query, and the associated ideal database rank. It also showed a problem with this definition, namely that the document similarities that contribute to a database's goodness may not be "globally valid," since they incorporate database-dependent *idf* factors. In this section we explore alternative ideal database ranks for a query. (Even other possibilities are discussed in [Gravano and García-Molina 1995b].) The first new ranks use the number of relevant documents for the query in each database. However, as we will discuss, we believe that ranks based on relevance are not

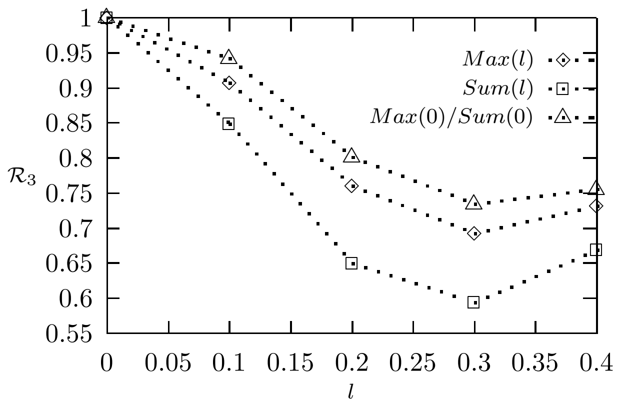

Fig. 4. Parameter $\mathcal{R}_3$ as a function of the threshold l, for ideal rank $Ideal(l)$.

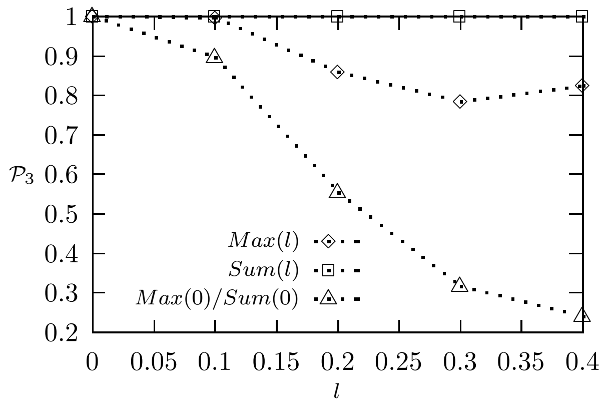

Fig. 5. Parameter $\mathcal{P}_3$ as a function of the threshold l, for ideal rank $Ideal(l)$.

appropriate for evaluating schemes like *vGlOSS*. Thus, the remaining ranks that we describe do not depend on end-user relevance judgments.

The first rank, *Rel_All*, simply orders the databases based on the number of relevant documents they contain for the given query. (See [French et al. 1998] for an experimental evaluation of *vGlOSS* using this ideal rank.) 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 end-user relevance is not appropriate for evaluating schemes like *vGlOSS*. That is, the best we can hope for any tool like *vGlOSS* 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 *vGlOSS* 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 *vGlOSS*. 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 2.2 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 2.2, 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 [Voorhees et al. 1995]. In general, determining what scaling factor to use to define the *Local*(l) ideal database rank is an interesting problem. If we incorporated scaling into the *Goodness* definition, we should modify *vGlOSS*'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.

3. CHOOSING BOOLEAN DATABASES

So far, we have discussed databases supporting the vector-space model of document retrieval. The *Boolean* model is more primitive than the vector-space model, but it is important because many sources still use it for answering queries. In this model, documents are represented as words with position information. Queries are expressions composed of words, and connectives such as "and," "or," "not," and proximity operations such as "within k words of." The answer to a query is the set of all the documents that satisfy the Boolean expression. Many other features are available with these systems, such as thesauri and regular expression matching. In this section, we present *bGlOSS*, a version of *GlOSS* for databases supporting the Boolean model of document retrieval. (*bGlOSS* is described in more detail in [Gravano et al. 1994a; Gravano et al. 1994b].)

Like *vGlOSS*, *bGlOSS* gives a hint of what databases might be useful for the user's query, based on word-frequency information for each database. Essentially, the Boolean model, as compared to the vector space model, impacts the statistics used by *bGlOSS* and the estimation functions used. Because in the Boolean model there are no document-query similarities (i.e., a document does or does not satisfy a query), *bGlOSS* only needs the $F = (f_{ij})$ matrix of Section 2.3, where f_{ij} is the number of documents in db_i that contain word t_j.

Example 6.: Consider three databases, A, B, and C, and suppose that *bGlOSS* has collected the statistics of Figure 6. Suppose that *bGlOSS* receives a query q=*retrieval* $\wedge$ *discovery* (this query searches for documents that contain both words, *retrieval* and *discovery*). Using the information in the figure, *bGlOSS* then estimates

Database	A	B	C
Number of documents	100	1000	200
Number of documents with the word *retrieval*	40	500	10
Number of documents with the word *discovery*	5	40	0

Fig. 6. A portion of the database frequency information that *bGlOSS* keeps for three databases.

the number of matching documents in each of the databases.

It is easy to see that no documents in C match q because C does not contain any documents with the word *discovery*. For the other two databases, *bGlOSS* has to "guess" the number of matching documents. There are different *estimators* that can be used to make this guess. One of the estimators that we study, *Ind* (for "independence"), estimates the result size as follows. Database A contains 100 documents, 40 of which contain the word *retrieval*. Therefore, the probability that a document in A contains the word *retrieval* is $\frac{40}{100}$. Similarly, the probability that an A document contains the word *discovery* is $\frac{5}{100}$. Under the assumption that words appear independently in documents, the probability that an A document has both words is $\frac{40}{100} \times \frac{5}{100}$. Consequently, we can estimate the result size of query q in database A as $Estimate(q, A) = \frac{40}{100} \times \frac{5}{100} \times 100 = 2$ documents. Similarly, $Estimate(q, B) = \frac{500}{1000} \times \frac{40}{1000} \times 1000 = 20$, and $Estimate(q, C) = \frac{10}{200} \times \frac{0}{200} \times 200 = 0$. Therefore, the best database for q according to *Ind* is B, followed by database A. Database C is not included in the database rank, since it cannot have any matching document. Unfortunately, as in the vector-space case, the database rank computed by *bGlOSS* might be wrong. For example, it could be the case that database B did not contain any matching document for q, while *Ind* predicted there would be 20 such documents in B. Furthermore, if database A did contain matching documents, then *Ind* would fail to conclude that database A is more promising than database B. ∎

3.1 Ranking Databases

Consider a *Boolean "and"* query q that we want to evaluate over a set of databases DB. (We consider other kinds of queries in [Gravano et al. 1994a].) *bGlOSS* ranks the databases in DB according to their estimated number of matches for q, and using an *estimator*. Different estimators are possible, and we studied several of them in [Gravano et al. 1994a; Gravano et al. 1994b]. In this article, we focus on the *Ind* (for "independence") estimator that we define below. For a database db_i, *bGlOSS* keeps:

—$|db_i|$, the total number of documents in database db_i, and

—f_{ij}, the number of documents in db_i that contain t_j, for all keyword field-designation pairs t_j. Note that, unlike *vGlOSS*, *bGlOSS* keeps different frequencies for a word appearing in different fields (e.g., author, title). The reason is that most Boolean sources support fields in their query language. (For example, a user can ask for documents having "Ullman" as one of the authors.)

As with *vGlOSS*, a real implementation of *bGlOSS* requires that each database cooperate and periodically export these frequencies to the *bGlOSS* server following some predefined protocol, like *STARTS* (see Section 2.3).

Given the frequencies and sizes for a set of databases DB, *bGlOSS* uses the *Ind* estimator to rank the databases in DB. This estimator is built upon the unrealistic assumption that keywords appear in the different documents of a database following independent and uniform probability distributions. Under this assumption, given a database db_i, any n keyword field-designation pairs $t_1, \ldots, t_n$, and any document $d \in db_i$, the probability that d contains all of $t_1, \ldots, t_n$ is:

$$\frac{f_{i1}}{|db_i|} \times \ldots \times \frac{f_{in}}{|db_i|}$$

So, according to *Ind*, the estimated number of documents in db_i that will satisfy the query $t_1 \wedge \ldots \wedge t_n$ is [Salton et al. 1983]:

$$Estimate_{Ind}(\ t_1 \wedge \ldots \wedge t_n, db_i) \ = \ \frac{\prod_{j=1}^{n} f_{ij}}{|db_i|^{n-1}} \tag{6}$$

As our previous example illustrated, this estimate may be incorrect, unless one of the frequencies is zero. In the later case, we know for sure that no document in db_i matches. (See [Gravano et al. 1994a; Gravano et al. 1994b] for a comparison of *Ind* against alternative estimators.)

3.2 Evaluating *bGlOSS*

This section uses the metrics of Section 2.4 to demonstrate that *bGlOSS* can select relevant databases effectively from among a large set of candidates [Tomasic et al. 1997]. The key difference from the evaluation in Section 2.4 is that the *goodness* of a database db for a Boolean query q is simply the number of documents in db that match q.

For our *bGlOSS* experiments, we used as data the complete set of United States patents for 1991. Each patent issued is described by an entry that includes various attributes (e.g., names of the patent owners, issuing date) as well as a text description of the patent. The total size of the patent data is 3.4 gigabytes. We divided the patents into 500 databases by first partitioning them into fifty groups based on date of issue, and then dividing each of these groups into ten subgroups, based on the high order digit of a subject-related patent classification code. This partitioning scheme gave databases that ranged in size by an order of magnitude, and were at least somewhat differentiated by subject. Both properties are ones we would expect to see in a real distributed environment. (See [Gravano et al. 1994a] for an evaluation of *bGlOSS* over a smaller number of independent, pre-existing collections.)

For test queries, we used a trace of 8,392 real-user queries that were issued at Stanford University to the INSPEC database from 4/12 to 4/25 in 1993. (INSPEC is a database of Physics, Electrical Engineering, and Computer Science bibliographic records.) We only considered correctly formed "and" queries. Also, we did not consider the so-called "phrase" queries (e.g., *titlephrase knowledge bases*). The final set of queries, $TRACE_{INSPEC}$, has 6,897 queries. Finally, we eliminated all queries with field designators not applicable to the patent data. Although IN-

n	$\mathcal{R}_n$
1	0.712
2	0.725
3	0.730
4	0.736
5	0.744
6	0.750
7	0.755
8	0.758
9	0.764
10	0.769

Fig. 7. The average $\mathcal{R}_n$ metric for 500 text databases and the $TRACE_{INSPEC}$ queries of Section 3.

SPEC is not a patent database, it covers a similar range of technical subjects, so we expected a fair number of hits against our patent data. Each of the remaining 3,719 queries is a Boolean conjunction of one or more words, e.g., *microwave* $\wedge$ *interferometer*.

To test *bGlOSS*, for each query we found the exact number of matching documents in each database, and computed the ideal database rank accordingly. We compared this ranking with the ranking suggested by *bGlOSS* by calculating, for various values of n, the $\mathcal{R}_n$ metric of Section 2.4.

Figure 7 shows the results of this experiment. Compared to an omniscient selector, *bGlOSS* does a reasonable job of selecting relevant databases, on average finding over seventy percent of the documents that could be found by examining an equal number of databases under ideal circumstances, with gradual improvement as the number of databases examined increases.

4. VARIATIONS AND DISCUSSION

In this section we discuss several issues that impact both *vGlOSS* and *bGlOSS*. In Section 4.1 we study the storage requirements of the *GlOSS* scheme, using *bGlOSS* for concreteness. We then discuss (Section 4.2) how a *GlOSS* server would deal with *both* vector-space and Boolean databases simultaneously. Finally, in Section 4.3 we study how collections of *GlOSS* servers could cooperate, and we show some experimental results, using *vGlOSS* for concreteness.

4.1 *bGlOSS* Storage Requirements

In this section we study the space requirements of *bGlOSS* and compare them with those of a full index of the databases. Our evaluation is for the 6 database scenario shown in Figure 8. Although the number of databases is small, we do have very detailed information about these databases, and we feel that our storage results are representative. Our storage estimates will be approximate, i.e., should be taken just as an indication of the relative order of magnitude of the corresponding requirements. An evaluation of the *vGlOSS* space requirements would be analogous, but is not covered here.

We start our analysis with the INSPEC database, and then consider the remaining bibliography databases in Figure 8. Figure 9 shows information on the INSPEC database that will be useful for computing the size of the *bGlOSS* information. This

Database	*Number of documents*	*Area*
INSPEC	1,416,823	Physics, Elect. Eng., Computer Sc.
COMPENDEX	1,086,289	Engineering
ABI	454,251	Business Periodical Literature
GEOREF	1,748,996	Geology and Geophysics
ERIC	803,022	Educational Materials
PSYCINFO	323,952	Psychology

Fig. 8. The six databases of the *bGlOSS* experimental study.

Field Designator	*Full Index*	*bGlOSS (threshold=0)*
	# of postings	*# of entries*
Author	4108027	311632
Title	10292321	171537
Publication	6794557	18411
Abstract	74477422	487247
Thesaurus	11382655	3695
Conference	7246145	11934
Organization	9374199	62051
Class	4211136	2962
Numbers (ISBN, ...)	2445828	12637
Report Numbers	7833	7508
Totals	130,340,123	1,089,614

Fig. 9. The *bGlOSS* summaries vs. a full text index for the INSPEC database.

information was generated using data from Stanford's FOLIO library information retrieval system. The "# of entries" column reports the number of entries required for each of the INSPEC indexes appearing in the $TRACE_{INSPEC}$ queries of Section 3.2. For example, there are $311,632$ different author last names appearing in INSPEC (field designation "author"), and each will have an associated entry in the INSPEC frequency information. A total of $1,089,614$ entries will be required for the INSPEC database. Each of these entries will correspond to a keyword field-designation pair and its associated frequency (e.g., <*author Knuth, 47*>, meaning that there are 47 documents in INSPEC with *Knuth* as the *author*). In contrast, if we were to keep the complete inverted lists associated with the different indexes we considered, $130,340,123$ postings would have to be stored in the full index.

Each of the *postings* of a full index will typically contain a field designation and a document identifier. If we dedicate one byte for the field designation and three bytes for the document identifier, we end up with four bytes per posting. Let us assume that, after compression, two bytes suffice per posting (compression of 50% is typical for inverted lists).

Each of the *frequencies* kept by *bGlOSS* will typically contain a field designation, a database identifier, and the frequency itself. Regarding the size of the frequencies themselves, only 1417 keyword field-designation pairs in INSPEC have more than 2^{16} documents containing them. Therefore, in the vast majority of the cases, two bytes suffice to store these frequencies, according to the INSPEC data we have available. We will thus assume that we dedicate two bytes per frequency. So, using one byte for the field designation and two bytes for the database identifier, we end

Size of	*Full Index*	*bGlOSS/threshold=0*
Vocabulary	3.13 MBytes	3.13 MBytes
Index	248.60 MBytes	2.60 MBytes
Total	251.73 MBytes	5.73 MBytes
% of Full Index	100	2.28

Fig. 10. Storage estimates for *bGlOSS* and a full text index for the INSPEC database.

up with five bytes per frequency. Again, after compression we will assume that 2.5 bytes are required per frequency. Using the data from Figure 9 and our estimates for the size of each posting and frequency information entry, we obtain the index sizes shown in Figure 10 ("Index" row).

The vocabulary for INSPEC [2], including only indexes that appear in $TRACE_{INSPEC}$ queries, consists of 819,437 words. If we dedicate four bytes to store each keyword [Gravano et al. 1993], around $4 \times 819,437$ bytes, or 3.13 MBytes are needed to store the INSPEC vocabulary. This statistic is shown in the "Vocabulary" row of Figure 10.

After adding the vocabulary and index sizes ("Total" row of Figure 10), the size of the frequency information that *bGlOSS* needs is only around 2.28% the size of the corresponding full index, for the INSPEC database.

So far, we have only focused on the space requirements of a single database, namely INSPEC. We will base the space requirement estimates for the six databases on the figures for the INSPEC database, for which we have reliable index information. To do this, we multiply the different values we calculated for INSPEC by a growth factor G (see Figure 8):

$$G = \frac{\sum_{db \in DB} |db|}{|\text{INSPEC}|} \approx 4.12$$

where $DB = \{\text{INSPEC, COMPENDEX, ABI, GEOREF, ERIC, PSYCINFO}\}$. Therefore, the number of postings required by a full index of the six databases is estimated as $G \times$ INSPEC number of postings $= 537,001,307$ postings, or around 1024.25 MBytes. The number of frequencies required by *bGlOSS* for the six databases is estimated as $G \times$ INSPEC number of frequencies $= 4,489,210$ frequencies, or around 10.70 MBytes (see the "Index" row of Figure 11).

The space occupied by the index keywords of the six databases considered will be proportional to the size of their *merged* vocabularies. Using index information from Stanford's FOLIO system, we can determine that the size of the merged vocabulary of the six databases we considered is approximately 90% of the sum of the six individual vocabulary sizes. Therefore, we estimate the size of the merged vocabulary for the six databases as $G \times 0.90 \times$ INSPEC vocabulary size = 3,038,472 words, or around 11.59 MBytes (see the "Vocabulary" row of Figure 11).

Figure 11 summarizes the storage estimates for *bGlOSS* and a full index. Note that the *bGlOSS* frequency information is only 2.15% the size of the full index. This percentage is even less than the corresponding figure we obtained above just for the INSPEC database (2.28%). The reason for this is the fact that the merged vocabulary size is only 90% of the sum of the individual vocabulary sizes. Although

[2]The field designators are stored with each posting and frequency, as described above.

Size of	*Full index*	*bGlOSS/threshold=0*
Vocabulary	11.59 MBytes	11.59 MBytes
Index	1024.25 MBytes	10.70 MBytes
Total	1035.84 MBytes	22.29 MBytes
% of Full index	100	2.15

Fig. 11. Storage estimates for *bGlOSS* and a full index for the six databases.

this 10% reduction "benefits" both *bGlOSS* and the full index case, the impact on *bGlOSS* is higher, since the vocabulary size is a much larger fraction of the total storage needed by *bGlOSS* than it is for the full index.

We have obtained the numbers of Figure 11 using some very rough estimates and approximations, so they should be taken cautiously. However, we think they are useful to illustrate the low space requirements of *bGlOSS*, which are two orders of magnitude lower than those of a full-text index of the databases. This is an important property, since *bGlOSS* should scale to large numbers of databases. Furthermore, this drastic space reduction makes the *bGlOSS* indexes less expensive to update and maintain, as well as decreasing the communication cost (for statistics) between the *bGlOSS* server and the distributed collections. Note, however, that the overall response time for a user query might be slower using *bGlOSS* than it would be if we could maintain a centralized full-text index of the collections. In effect, after obtaining an answer from *bGlOSS*, the suggested databases need to be contacted to obtain the documents that match the query. In contrast, a full-text index of the databases would produce a document set directly.

Pruning the bGlOSS Summaries. The statistical information kept by both *bGlOSS* and *vGlOSS* can be "compressed" in a variety of ways, for additional space savings. Here we illustrate one possible technique, again using *bGlOSS* for concreteness. The technique is based on a frequency *threshold*. If a database db_i has no more than *threshold* documents with a given keyword-field pair t_j, then *bGlOSS* will not keep this information. Therefore, *bGlOSS* will assume that f_{ij} is zero whenever this data is needed.

As a result of the introduction of *threshold*, the estimator may now conclude that some database db_i does not contain any documents matching a query of the form $t_1 \wedge \ldots \wedge t_n$ if f_{ij} is missing, for some j, while in fact db_i does contain documents matching the query. This situation was not possible before: if f_{ij} was missing from the information set of the estimator, then $f_{ij} = 0$, and so, there could be no documents in db_i satisfying such a query.

Introducing *thresholds* reduces the storage cost of the estimator. Figure 12 reports the number of entries that would be left, for different field designators, in the frequency information for the INSPEC database. Some field designators (e.g., "thesaurus") are not affected much by this pruning of the smallest entries, whereas the space requirements for some others (e.g., "author," "title," and "abstract") are reduced drastically. Adding together all of the indexes, the number of entries in the INSPEC frequency information kept by *bGlOSS* decreases very fast as *threshold* increases: for *threshold*=1, for instance, $508,978$ entries, or 46.71% of the original number of entries, are eliminated. In [Gravano et al. 1994a] we report experimental results that show that the performance of *bGlOSS* is only slightly sensitive to small

Field Designator	*threshold* 0	1	2	3	4	5
Author	311632	194769	150968	125220	107432	94248
Title	171537	85448	62759	51664	44687	40007
Publication	18411	11666	10042	9281	8832	8535
Abstract	487247	227526	163644	133323	115237	102761
Thesaurus	3695	3682	3666	3653	3641	3637
Conference	11934	10138	9887	9789	9702	9653
Organization	62051	34153	26518	22612	20121	18382
Class	2962	2953	2946	2937	2931	2926
Numbers (ISBN, ...)	12637	10199	10067	9946	9865	9779
Report Numbers	7508	102	37	22	14	12
Totals	1089614	580636	440534	368447	322462	289940
%	100	53.29	40.43	33.81	29.59	26.61

Fig. 12. Number of entries left for the different thresholds and field designators in the INSPEC database.

increases of *threshold*. Therefore, the size of the *bGlOSS* frequency information can be substantially reduced beyond the already small size estimated in Figure 11.

4.2 *GlOSS* Over Both Vector-Space and Boolean Databases

It is possible for a single *GlOSS* server to keep statistics on both Boolean and vector-space sources. For the Boolean sources, it would collect the statistics *bGlOSS* needs, while for the vector-space sources, it would keep the *vGlOSS* information. This combined *GlOSS* server could easily treat user queries separately. That is, a Boolean query could be processed against the information on Boolean sources, while vector-space queries could be directed to the remaining statistics.

It would of course be better for *GlOSS* to try to suggest sources, regardless of the query type. In such a case, the user could be warned if a good source uses a query model that is different from the one the query was posed in (since the user would have to reformulate the query for that source). Processing queries in this integrated fashion introduces two challenging problems. Here we mention these problems and briefly sketch possible solutions:

—*Query specification:* Vector-space queries are usually just lists of words, while Boolean queries are structured through connectives like "and," "or," and "not." Thus, *GlOSS* must somehow translate a query from one form into the other. For example, if *GlOSS* receives a vector-space query q (a list of words), then it can choose to interpret q as the Boolean "and" of these words for the Boolean sources. For a Boolean query with only "and" and "or" operators, it may remove all operators and consider the plain words as the vector-space query. The "not" Boolean operator complicates the translation further. A possibility is for *GlOSS* to eliminate the negated terms from the query for an initial goodness estimate. *GlOSS* can then use the negated terms to adjust the initial estimates, so that a database containing a negated term many times might see its goodness estimate for the query decreased. These mappings are clearly not precise, but could still give the user reasonable database suggestions.

—*Database ranking:* Both *vGlOSS* and *bGlOSS* rank databases for a query q based on the numeric *goodness* of the databases for q. Therefore, to rank both vector-space and Boolean sources together, a simple solution is for *GlOSS* to simply "normalize" the *vGlOSS* goodness scores and the *bGlOSS* goodness scores so their relative magnitudes are comparable, and then compute the database ranks in the usual way. Alternatively, *GlOSS* could produce two different database ranks: one including the vector-space databases, and the other including the Boolean databases.

4.3 Decentralizing *GlOSS*

In this section we show how we can build a more distributed version of *GlOSS* using essentially the same methodology that we developed in the previous sections. Suppose that we have a number of *GlOSS* 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.) For simplicity, assume all servers are of the same type, either *bGlOSS* or *vGlOSS*. We will now build a *higher-level GlOSS* server, *hGlOSS*, that summarizes the contents of the *GlOSS* servers in much the same way as the *GlOSS* servers summarize the contents of the underlying databases. [3] A user first queries the *hGlOSS* server, obtaining a rank of the *GlOSS* servers according to how likely they are to have indexed useful databases. Then the user visits the suggested *GlOSS* servers, submitting the query there to obtain suggested databases to visit.

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" *GlOSS* servers, and will let replicas of a logically unique higher-level *GlOSS*, *hGlOSS*, concisely summarize the contents of their *GlOSS* servers.

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

To keep the size of the *hGlOSS* server small, the information that the *hGlOSS* server keeps about a *GlOSS* server G_r is limited. For brevity, we focus our discussion now on the *vGlOSS* version of *GlOSS*, but we can proceed analogously for *bGlOSS*. *hGlOSS* keeps one or both of the following matrices (see Section 2.3):

—$H = (h_{rj})$: h_{rj} is the number of *databases* in *vGlOSS* 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 *vGlOSS* G_r

In other words, for each word t_j and each *vGlOSS* server G_r, *hGlOSS* needs (at most) two numbers, in much the same way as the *vGlOSS* servers summarize the contents of the document databases (Section 2.3). (An alternative would be for *hGlOSS* to (also) maintain a matrix $S = (s_{rj})$, where s_{rj} is the sum of the weight of word t_j over all documents in databases in *vGlOSS* G_r.)

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

EXAMPLE 7.: Consider a *vGlOSS* 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 *vGlOSS* weights and frequencies:

$$computer : (db_1, 5, 3.4), (db_2, 2, 1.8), (db_3, 1, 0.3)$$

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 *vGlOSS* methodology: given a query q, we define the goodness of each *vGlOSS* 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 2.2.

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

To illustrate *hGlOSS*'s potential, we briefly describe one experiment. For this, we divide the 53 databases of Section 2.5 into five randomly-chosen groups of around ten databases each. Each of these groups corresponds to a different *vGlOSS* server.

We assume that the *vGlOSS* servers approximate ideal rank $Ideal(0)$ with the $Max(0)$ database rank. Next, we define the goodness of a *vGlOSS* 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 *vGlOSS* servers. To approximate this ideal rank, *hGlOSS* periodically receives the H matrix defined above from the underlying *vGlOSS* servers. For query q with words $t_1, \ldots, t_n$ and *vGlOSS* server G_r, $h_{r1}, \ldots, h_{rn}$ are the database counts for G_r associated with the query words. (Word t_1 appears in h_{r1} *databases* in *vGlOSS* server G_r, and so on.) Assume that $h_{r1} \leq \ldots \leq h_{rn}$. 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.

Figure 13 shows the different values of the (adapted) $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics for the 6,800 queries of Section 2.5. Note that $\mathcal{P}_n = 1$ for all n, because every time *hGlOSS* chooses a *vGlOSS* 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" *vGlOSS* 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

n	$\mathcal{R}_n$	$\mathcal{P}_n$
1	0.985	1
2	0.991	1
3	0.994	1
4	0.998	1
5	1	1

Fig. 13. The $\mathcal{R}_n$ and $\mathcal{P}_n$ metrics for *hGlOSS* and our sample experiment.

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 of the centralized *GlOSS* architecture.

5. RELATED WORK

Many solutions have been presented recently for the text-source discovery problem, or, more generally, for the resource-discovery problem: the text-source discovery problem is a subcase of the resource-discovery problem, since the latter generally deals with a larger variety of types of information [Obraczka et al. 1993; Schwartz et al. 1992].

One solution to the text-source discovery problem is to let the database selection be driven by the user. Thus, the user will be aware of and an active participant in this selection process. Different systems follow different approaches to this: one such approach is to let users "browse" through information about the different resources. A typical example of this paradigm is Yahoo! (`http://www.yahoo.-com`). As another example, the Prospero File System [Neuman 1992] lets users organize information available in the Internet through the definition (and sharing) of customized views of the different objects and services available to them.

A different approach is to keep a database of "meta-information" about the available databases and have users query this database to obtain the set of databases to search. For example, WAIS [Kahle and Medlar 1991] 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. One disadvantage is that the master-database documents have to be written by hand to cover the relevant topics, and have to be manually kept up to date as the underlying database changes. However, freeWAIS [Fullton et al. 1993] automatically adds the most frequently occurring words in an information server to the associated description in the directory of servers. Another drawback is that in general, databases containing relevant documents might be missed if they are not chosen during the database-selection phase. [Duda and Sheldon 1994] shows sample queries for which very few of the existing relevant servers are found by querying the WAIS directory of servers (e.g., only 6 out of 223 relevant WAIS servers).

Reference [Schwartz 1990] follows a probabilistic approach to the resource-discovery problem, and presents a resource-discovery protocol 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 [Schwartz 1993].

In Indie (shorthand for "Distributed Indexing") [Danzig et al. 1992], information is indexed by "Indie brokers," each of which has associated, among other administrative data, a 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," which is queried initially by the users to obtain a list of the brokers whose generator rules match the given query. See also [Danzig et al. 1991]. [Barbará and Clifton 1992], [Ordille and Miller 1992], and [Simpson and Alonso 1989] are other examples of this type of approach in which users query "meta-information" databases.

A "content-based routing" system is used in [Sheldon et al. 1994] to address the resource-discovery problem. The "content routing system" 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. Also, users can browse the possible values of each content-label attribute.

The WHOIS++ directory service (`http://www.ucdavis.edu/whoisplus`) 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. A query that is presented to an index server is forwarded to the (index) servers whose centroids match the query.

In [Flater and Yesha 1993], 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, it is forwarded through the most promising link according to these statistics. References [Morris et al. 1993], [Zahir and Chang 1992], and [Morris et al. 1992] follow an expert-systems approach to solving the related problem of selecting online business databases.

A complementary approach to *GlOSS* is taken by Chamis [Chamis 1988]. Briefly, the approach this article takes is to expand a user query with thesaurus terms. The expanded query is compared with a set of databases, and the query terms with exact matches, thesauri matches, and "associative" matches are counted for each database. Each database is then ranked as a function of these counts. We believe that this approach is complementary in its emphasis on thesauri to expand the meaning of a user query.

Reference [Callan et al. 1995] has applied inference networks (from information retrieval) to the text-source 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 Harvest system [Bowman et al. 1994] 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.

An interesting alternative approach is the Pharos system [Dolin et al. 1996], which combines browsing and searching for resource discovery. This system keeps information on the number of objects that each source has for each category of a subject hierarchy like the Library of Congress's LC Classification System.

6. CONCLUSIONS

We have shown how to construct source-discovery servers for vector-space and Boolean text databases, and for hierarchies of source-discovery servers. Based on compact collected statistics, these servers can provide very good hints for finding the relevant databases, or finding relevant lower-level servers with more information for a given query. An important feature of our approach is that the same machinery can be used for both lower-level or higher-level servers. Our experimental results show that *bGlOSS*, *vGlOSS*, and *hGlOSS* are quite promising and could provide useful services in large, distributed information systems. The storage cost of *GlOSS* is relatively low: for our case study, the size of the *GlOSS* index was about 2% of the size of a full index. A small index means it is easier to replicate the discovery service, for improved load balancing and availability.

Our approach to solving the text-source discovery problem could also deal with information servers that charge for their use. Since we are selecting what databases to search according to a quantitative measure of their "goodness" for a query, we could easily incorporate this cost factor so that, for example, given two equally promising databases, a higher value would be assigned to the less expensive of the two.

A *bGlOSS* server has been implemented and is available for testing. This server keeps information on 40+ collections of computer science technical reports that are part of the NCSTRL project (`http://www.ncstrl.org`). The *bGlOSS* server is available on the World-Wide Web at `http://gloss.stanford.edu`.

Acknowledgments

Section 3.2 is joint work with Laura Haas, Calvin Lue, and Peter Schwarz at IBM Almaden [Tomasic et al. 1997]. We also thank Helena Galhardas for her useful comments on an earlier version of the paper.

REFERENCES

Barbará, D. and Clifton, C. 1992. Information Brokers: Sharing knowledge in a heterogeneous distributed system. Technical Report MITL-TR-31-92 (Oct.), Matsushita Information Technology Laboratory.

Bowman, C. M., Danzig, P. B., Hardy, D. R., Manber, U., and Schwartz, M. F. 1994. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94 (Aug.), Department of Computer Science, University of Colorado-Boulder.

Callan, J. P., Lu, Z., and Croft, W. B. 1995. Searching distributed collections with inference networks. In *Proceedings of the Eighteenth ACM International Conference on Research and Development in Information Retrieval (SIGIR'95)* (July 1995).

Chamis, A. Y. 1988. Selection of online databases using switching vocabularies. *Journal of the American Society for Information Science 39*, 3, 217–218.

Chang, C.-C. K., García-Molina, H., and Paepcke, A. 1996. Boolean query mapping across heterogeneous information sources. *IEEE Transactions on Knowledge and Data Engineering 8*, 4 (Aug.), 515–521.

Danzig, P. B., Ahn, J., Noll, J., and Obraczka, K. 1991. Distributed indexing: a scalable mechanism for distributed information retrieval. In *Proceedings of the Fourteenth ACM International Conference on Research and Development in Information Retrieval (SIGIR'91)* (Oct. 1991).

Danzig, P. B., Li, S.-H., and Obraczka, K. 1992. Distributed indexing of autonomous Internet services. *Computing Systems 5*, 4, 433–459.

Dolin, R., Agrawal, D., Dillon, L., and Abbadi, A. E. 1996. Pharos: A scalable distributed architecture for locating heterogeneous information sources. Technical Report TRCS96-05 (July), Computer Science Department, University of California at Santa Barbara.

Duda, A. and Sheldon, M. A. 1994. Content routing in a network of WAIS servers. In *Proceedings of the Fourteenth IEEE International Conference on Distributed Computing Systems* (June 1994).

Flater, D. W. and Yesha, Y. 1993. An information retrieval system for network resources. In *Proceedings of the International Workshop on Next Generation Information Technologies and Systems* (June 1993).

French, J. C., Powell, A. L., Viles, C. L., Emmitt, T., and Prey, K. J. 1998. Evaluating database selection techniques: A testbed and experiment. In *Proceedings of the Twentyfirst ACM International Conference on Research and Development in Information Retrieval (SIGIR'98)* (Aug. 1998).

Fullton, J., Warnock, A., et al. 1993. Release notes for freeWAIS 0.2.

Gravano, L., Chang, C.-C. K., García-Molina, H., and Paepcke, A. 1997. *STARTS*: Stanford proposal for Internet meta-searching. In *Proceedings of the 1997 ACM International Conference on Management of Data (SIGMOD'97)* (May 1997).

Gravano, L. and García-Molina, H. 1995a. Generalizing *GlOSS* for vector-space databases and broker hierarchies. In *Proceedings of the Twenty-first International Conference on Very Large Databases (VLDB'95)* (Sept. 1995), pp. 78–89.

Gravano, L. and García-Molina, H. 1995b. Generalizing *GlOSS* to vector-space databases and broker hierarchies. Technical Report STAN-CS-TN-95-21 (May), Computer Science Department, Stanford University.

Gravano, L. and García-Molina, H. 1997. Merging ranks from heterogeneous Internet sources. In *Proceedings of the Twenty-third International Conference on Very Large Databases (VLDB'97)* (Aug. 1997).

Gravano, L., García-Molina, H., and Tomasic, A. 1993. The efficacy of *GlOSS* for the text-database discovery problem. Technical Report STAN-CS-TN-93-002 (Nov.), Computer Science Department, Stanford University.

Gravano, L., García-Molina, H., and Tomasic, A. 1994a. The effectiveness of *GlOSS* for the text-database discovery problem. In *Proceedings of the 1994 ACM International Conference on Management of Data (SIGMOD'94)* (May 1994).

Gravano, L., García-Molina, H., and Tomasic, A. 1994b. Precision and recall of *GlOSS* estimators for database discovery. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems (PDIS'94)* (Sept. 1994).

Kahle, B. and Medlar, A. 1991. An information system for corporate users: Wide Area Information Servers. Technical Report TMC199 (April), Thinking Machines Corporation.

Morris, A., Drenth, H., and Tseng, G. 1993. The development of an expert system for online company database selection. *Expert Systems 10*, 2 (May), 47–60.

Morris, A., Tseng, G., and Drenth, H. 1992. Expert systems for online business database selection. *Library Hi Tech 10*, 1-2, 65–68.

Neuman, B. C. 1992. The Prospero File System: A global file system based on the Virtual System model. *Computing Systems 5*, 4, 407–432.

Obraczka, K., Danzig, P. B., and Li, S.-H. 1993. Internet resource discovery services. *IEEE Computer 26*, 9 (Sept.), 8–22.

Ordille, J. J. and Miller, B. P. 1992. Distributed active catalogs and meta-data caching in descriptive name services. Technical Report #1118 (November), University of Wisconsin-Madison.

Salton, G. 1989. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley.

Salton, G., Fox, E. A., and Voorhees, E. M. 1983. A comparison of two methods for Boolean query relevance feedback. Technical Report TR 83-564 (July), Computer Science Department, Cornell University.

Salton, G. and McGill, M. J. 1983. *Introduction to modern information retrieval*. McGraw-Hill.

Schwartz, M. F. 1990. A scalable, non-hierarchical resource discovery mechanism based on probabilistic protocols. Technical Report CU-CS-474-90 (June), Department of Computer Science, University of Colorado at Boulder.

Schwartz, M. F. 1993. Internet resource discovery at the University of Colorado. *IEEE Computer 26*, 9 (Sept.), 25–35.

Schwartz, M. F., Emtage, A., Kahle, B., and Neuman, B. C. 1992. A comparison of Internet resource discovery approaches. *Computing Systems 5*, 4, 461–493.

Selberg, E. and Etzioni, O. 1995. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the Fourth International World-Wide Web Conference* (Dec. 1995).

Sheldon, M. A., Duda, A., Weiss, R., O'Toole, J. W., and Gifford, D. K. 1994. A content routing system for distributed information servers. In *Proceedings of the Fourth International Conference on Extending Database Technology* (1994).

Simpson, P. and Alonso, R. 1989. Querying a network of autonomous databases. Technical Report CS-TR-202-89 (January), Department of Computer Science, Princeton University.

Tomasic, A., Gravano, L., Lue, C., Schwarz, P., and Haas, L. 1997. Data structures for efficient broker implementation. *ACM Transactions on Information Systems 15*, 3 (July), 223–253.

Voorhees, E. M., Gupta, N. K., and Johnson-Laird, B. 1995. The collection fusion problem. In *Proceedings of the Third Text Retrieval Conference (TREC-3)* (March 1995).

Yan, T. W. and García-Molina, H. 1995. SIFT–a tool for wide-area information dissemination. In *Proceedings of the 1995 USENIX Technical Conference* (Jan. 1995), pp. 177–86.

Zahir, S. and Chang, C. L. 1992. Online-Expert: An expert system for online database selection. *Journal of the American Society for Information Science 43*, 5 (June), 340–357.