# The Effectiveness of *GlOSS* for the Text Database Discovery Problem

Luis Gravano* Héctor García-Molina† Anthony Tomasic‡

## Abstract

The popularity of on-line document databases has led to a new problem: finding which text databases (out of many candidate choices) are the most relevant to a user. Identifying the relevant databases for a given query is the *text database discovery problem.* The first part of this paper presents a practical solution based on estimating the result size of a query and a database. The method is termed *GlOSS–Glossary of Servers Server.* The second part of this paper evaluates the effectiveness of *GlOSS* based on a trace of real user queries. In addition, we analyze the storage cost of our approach.

## 1 Introduction

Information vendors such as Dialog and Mead Data Central provide content-indexed access to multiple databases. Dialog for instance has over four hundred databases. In addition, the advent of archie, WAIS, World Wide Web, and other INTERNET tools has provided easy, distributed access to many more hundreds of text document databases. Thus, users are faced with finding the databases that are relevant to their information need (the user query). This paper presents a framework for (and analyzes a solution to) this problem, which we call the *text database discovery problem.*

The difficulty of our problem stems from the large number of databases available to the users and their worldwide distribution: any solution to this problem should scale with the increasing number of data sources. For example, forwarding a user's query to all known databases and merging the results obtained is not a feasible solution, due to the enormous amount of traffic that would be generated and the load that would be received by the information sources. Also, if the databases charge for their use, this approach would be exceedingly expensive to the users, since most likely lots of useless databases would be accessed when processing each query. The other obvious solution, that of building a central full index for *all* of the documents, does not scale well either. (As we will see, full indexes tend to be as large as the document collection itself.)

Our solution to the text database discovery problem is to build a service that can suggest potentially good databases to search. Then, a user's query will go through two steps: first, the query is presented to our server (dubbed *GlOSS*, for **Gl**ossary **O**f **S**ervers **S**erver) to select a set of promising databases to search. During the second step, the query is actually evaluated at the chosen databases[1]. *GlOSS* gives a hint of what databases might be useful for the user's query, based on word-frequency information for each database. This information indicates how many documents at that database actually contain each word in the database's vocabulary. For example, a Computer Science Library could report that the word *Knuth* occurs in 180 documents, the word *computer*, in 25,548 documents, and so on. This information is orders of magnitude smaller than a full index (see Section 6.3 and Figure 11), since for each word we only need to keep its frequency, as opposed to the identities of the documents that contain it.

**Example 1.1** Consider four databases, $A$, $B$, $C$, and $D$, and suppose that *GlOSS* has collected the statistics of Figure 1. If *GlOSS* receives a query $q$="*find Knuth*

*Stanford University, Computer Science Dept., Margaret Jacks Hall, Stanford, CA 94305. E-mail: gravano@cs.stanford.edu

†Stanford University, Computer Science Dept., Margaret Jacks Hall, Stanford, CA 94305. E-mail: hector@cs.stanford.edu

‡Princeton University, Department of Computer Science. Current Address: Stanford University, Computer Science Dept., Margaret Jacks Hall, Stanford, CA 94305. E-mail: tomasic@cs.stanford.edu

[1] As an intermediate step, *GlOSS* could show the chosen databases to the user, who would in turn select which ones to actually search.

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| Knuth 100 | Knuth 10 | Knuth 4 | Knuth 10 |
| computer 100 | computer 10 | computer 100 | computer 0 |
| $d = 1000$ | $d = 100$ | $d = 200$ | $d = 20$ |

Figure 1: Portion of the database frequency information kept by *GlOSS* for four databases. Parameter $d$ is the database size in documents.

$\wedge$ *computer*" (this query searches for documents that contain both words, "*Knuth*" and "*computer*"), *GlOSS* has to estimate the number of matching documents in each of the four databases. Figure 1 shows that database $D$ does not contain any documents with the word "*computer*," and so, there cannot be any documents in $D$ matching query $q$. For the other three databases, *GlOSS* has to "guess" what the number of documents matching query $q$ is: an *estimator* for *GlOSS* uses the *GlOSS* information to make this guess. There are different ways in which this can be done. In this paper, we study *Ind*, an estimator for *GlOSS* that estimates the result size of the given query in each of the databases in the following way. Database $A$ contains 1000 documents, 100 of which contain the word "*Knuth.*" Therefore, the probability that a document in $A$ contains the word "*Knuth*" is $\frac{100}{1000}$. Similarly, the probability that a document in $A$ contains the word "*computer*" is $\frac{100}{1000}$. Under the assumption that words appear independently in documents [2] the probability that a document in database $A$ has both the words "*Knuth*" and "*computer*" is $\frac{100}{1000} \times \frac{100}{1000}$. Consequently, we can estimate the result size of query $q$ in database $A$ as $f(q, A) = \frac{100}{1000} \times \frac{100}{1000} \times 1000 = 10$ documents. Similarly, $f(q, B) = \frac{10}{100} \times \frac{10}{100} \times 100 = 1$, $f(q, C) = \frac{4}{200} \times \frac{100}{200} \times 200 = 2$, and $f(q, D) = \frac{10}{20} \times \frac{0}{20} \times 20 = 0$ (as we explained above).

The *Ind* estimator chooses those databases with the highest estimates as the answer to the given query. So, *Ind* will return $\{A\}$ as the answer to $q$. This may or may not be a "correct" answer, depending on different factors. Firstly, it is possible that some of the result size estimates given by *Ind* are wrong. For example, it could be the case that database $A$ did not contain any document matching $q$, while *Ind* predicted there would be 10 such documents in $A$. Furthermore, if databases $B$ and $C$ did contain matching documents, then *Ind* would fail to pick any database with matching documents (since its answer was $\{A\}$).

Secondly, even if the estimates given by *Ind* are accurate, the correctness of the produced answer depends on the *semantics* of the query the user that issued the query is interested in. Assume in what follows that the result size estimates given above are correct (i.e., there actually are 10 documents matching query $q$ in database $A$, one in database $B$, two in database $C$, and none in database $D$). Given a query and a set of databases, the user may be interested in (at least) four different *semantics* for the query:

- *Exhaustive Search.* The user is interested in the set of all databases that contain any *matching* documents. Since databases $A$, $B$, and $C$ contain documents that match the query above, the set of databases to search should be $\{A, B, C\}$. The answer given by *Ind*, $\{A\}$, is thus not a correct answer according to this semantics. In [1] we study an estimator targeted at this semantics.
- *All-Best Search.* The user wants all the *best* databases for the query. The best databases are the ones that contain more matching documents than any other database. In this case, the user is willing to miss some databases, as long as they are not the best ones. That is, the user recognizes that there are more databases that could be examined, but wants to ensure that at least those databases having the highest payoff (i.e., the largest number of documents) are searched. Since $A$ is the database that contains the most documents matching $q$ (10), the answer should be $\{A\}$. This is exactly the answer given by *Ind*.
- *Only-Best Search.* The user only wants to examine (some) best databases. Because of limited resources (e.g., time, money) the user only wants to submit the query at databases that will yield the highest payoff. Since $A$ is the best database for $q$, the answer should be $\{A\}$. This is exactly the answer given by *Ind*.
- *Sample Search.* In this case, the user is in "browsing" mode, and simply wants to obtain some matching documents. Therefore, since $A$, $B$, and $C$ contain matching documents, any non-empty subset of $\{A,\ B,\ C\}$ will be a correct answer. The answer produced by *Ind*, $\{A\}$, is one such subset. □

As a result of space limitations, in this paper we focus on the All-Best and Only-Best semantics. In [1] we explore the other query semantics.

As we mentioned above, the answers given by an estimator may be wrong. Therefore, given an estimator, we need to evaluate its effectiveness with respect to the different query semantics. For this, we have performed experiments using query traces from the FOLIO library information retrieval system at Stanford University, and involving six databases available through FOLIO. As we will see, the results obtained for *GlOSS* and several

[2] Although this independence assumption is questionable, we obtained good experimental results with it. We examine this assumption further in [1].

estimators are very promising. Even though *GlOSS* keeps a small amount of information about the contents of the databases, this information proved to be sufficient to produce very useful hints on where to search.

Another advantage of *GlOSS* is that its frequency information can be updated mechanically, that is, sources can periodically extract word counts and send them to *GlOSS*. Other approaches (see Section 2) require human-generated summaries of the contents of a database, and are prone to errors or very out-of-date information. Also, since *GlOSS*' storage requirements are so low, it is straightforward to replicate the service at many sites. Thus, a user may be able to consult *GlOSS* at the local machine or cluster, and immediately determine the candidate databases for a given query.

The contributions of this paper are:

- a formal framework for the text database discovery problem,
- the concept of a Glossary Of Servers Server (*GlOSS*) that routes queries to appropriate information sources, based on previously collected frequency statistics about the sources,
- an estimator that may be used by *GlOSS* for making decisions, and
- an experimental evaluation of *GlOSS* according to different semantics for the queries, using real users' queries.

Of course, *GlOSS* is not the only solution to the text database discovery problem, and in practice we may wish to combine it with other complementary strategies. Section 2 describes these strategies. We note, incidentally, that, to the best of our knowledge, experimental evaluations of these other strategies *for the text database discovery problem* are rare: in most cases, strategies are presented with no statistical evidence as to how good they are at locating sites with documents of interest *for actual user queries*. Thus, we view the experimental methodology and results of this paper (even though they still have limitations) as an important contribution to this emerging research area.

Section 3 introduces *GlOSS* and the concept of an *estimator*. In particular, Section 3.3 describes *Ind*, the estimator for *GlOSS* that we will evaluate in the rest of the paper, using the evaluation criteria defined in Section 3.4. Section 4 describes the experiments performed to assess the effectiveness of *GlOSS*. Section 5 reports the experimental results. Section 6 examines *GlOSS*' space requirements and introduces enhancements to further reduce them.

## 2 Related work

Many solutions have been presented recently for the text database discovery problem, or, more generally, for the resource discovery problem: the text database discovery problem is a subcase of the resource discovery problem, since the latter generally deals with a larger variety of types of information [2, 3].

One solution to the text database 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 databases. Examples include Gopher [2] and World Wide Web [4]. Various search facilities are being created for these systems, like the Veronica Service [5] for Gopher, for example. The Prospero File System [6] 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 [7] 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. In addition to this, free-WAIS [8] automatically adds the 50 most frequently occurring words in an information server to the associated description in the directory of servers. The users first query the master database, and once they have identified potential databases, direct their query to these databases.

Schwartz [9, 10] presents a probabilistic resource discovery protocol that conceptually 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.

In Indie (shorthand for "Distributed Indexing") [11, 12], information is indexed by "Indie brokers," each of which has associated a boolean query (called a "generator rule"). Each broker indexes (not necessarily local) documents that satisfy its generator rule. The generator objects associated with the brokers are gathered by a "directory of servers." [13], [14], [15], and [16] are other examples of this type of approach in which users query "meta-information" databases. The master database idea can be enhanced if we automatically extract the semantics of queries and databases [17].

A "content based routing" system is used in [18] to address the database discovery problem. The "content routing system" keeps a "content label" for each information server, with attributes describing the contents of the collection.

Chamis [19] takes a complementary approach to *GlOSS*. Each user query is expanded 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.

# 3 *GlOSS*: Glossary Of Servers Server

Consider a query $q$ (permissible queries are defined in Section 3.1) that we want to evaluate over a set of databases $DB$. *GlOSS* selects a subset of $DB$ consisting of "good candidate" databases for actually submitting $q$. To make this selection, *GlOSS* uses an *estimator* (Section 3.3), that assesses how "good" each database in $DB$ is with respect to the given query, based on the word-frequency information on each database (Section 3.2).

## 3.1 Query representation

In this paper, we will only consider *boolean "and"* queries, that is, queries that consist of positive atomic subqueries connected by the boolean "and" operator (denoted as "$\wedge$" in what follows). An atomic subquery is a *keyword field-designation* pair. An example of a query is "*find author Knuth* $\wedge$ *subject computer.*" This query has two atomic subqueries: "*author Knuth*" and "*subject computer.*" In "*author Knuth,*" *author* is the field designation, and *Knuth* the corresponding keyword[3].

The reason why we are considering only boolean queries so far is because this model is used by library systems and information vendors worldwide. Also, the system we had available to perform our experiments uses only boolean queries (see Section 4.1). Nevertheless, it should be stressed that we can generalize the approach we take in this paper to the vector space retrieval model [20]. The reason why we restrict our study to "and" queries is that we want to understand a simple case first. Also, most of the queries in the trace we studied (see Section 4.1) are "and" queries. However, a limited form of "or" queries is implicit whenever the *subject* field designation is used (see Section 6.1).

## 3.2 Database word-frequency information

*GlOSS* keeps the following information on the databases:

- $DBSize(db)$, the total number of documents in database $db$, $\forall\ db \in DB$, and

- $freq(t, db)$, the number of documents in $db$ that contain $t$, $\forall\ db \in DB$, and for all keyword field-designation pairs $t$. Note that *GlOSS* does not have available the actual "inverted lists" corresponding to each keyword-field pair and each database, but just the length of these inverted lists. The value $freq(t, db)$ is the size of the result of query "*find t*" in database $db$.

  If $freq(t, db) = 0$, *GlOSS* does not need to store this explicitly, of course. Therefore, if *GlOSS* finds no information about $freq(t, db)$, then $freq(t, db)$ will be assumed to be zero (see Section 6.4).

A real implementation of *GlOSS* would require that each database cooperate and periodically submit these frequencies to the *GlOSS* server following some predefined protocol.

Section 6.4 modifies the frequency information kept by *GlOSS* on each database so as to reduce its size.

## 3.3 The *Ind* estimator

This section describes *Ind*, the estimator that we will use in our experiments. (In [1] we study the effectiveness of other estimators for *GlOSS*.) An estimator consists of a function ($\mathsf{ESize}_{Ind}$ below) that estimates the result size of a query in each of the databases, and a "matching" function (the max function for *Ind*), that uses these estimates to select the set of databases ($Chosen_{Ind}$ below) where to submit the given query. *Ind* (for "independence") is an estimator built upon the (possibly unrealistic) assumption that keywords appear in the different documents of a database following independent and uniform probability distributions. Under this assumption, given a database $db$, any $n$ keyword field-designation pairs $t_1, \ldots, t_n$, and any document $d \in db$, the probability that $d$ contains all of $t_1, \ldots, t_n$ is:

$$\frac{freq(t_1, db)}{DBSize(db)} \times \ldots \times \frac{freq(t_n, db)}{DBSize(db)}$$

So, the estimated number of documents in $db$ that will satisfy the query "*find* $t_1 \wedge \ldots \wedge t_n$" is given, according to *Ind*, by:

$$\mathsf{ESize}_{Ind}(find\ t_1 \wedge \ldots \wedge t_n, db) = \frac{\prod_{i=1}^{n} freq(t_i, db)}{DBSize(db)^{n-1}} \quad (1)$$

*Ind* chooses those databases with the *highest* estimates (as given by $\mathsf{ESize}_{Ind}$)[4]. The $Chosen_{Ind}$ set of

[3] Uniform field designators for all the databases we considered (see Section 4.1) were available for our experiments. However, *GlOSS* does not rely completely on this, and could be adapted to the case where the field designators are not uniform across the databases, for example.

[4] In [1] we present a variation to *Ind* that arises from making its "matching" function more flexible.

| | INSPEC | PSYCINFO |
|---|---|---|
| $DBSize(\_)$ | 1,416,823 | 323,952 |
| $freq(author\ D.\ Knuth, \_)$ | 13 | 0 |
| $freq(title\ computer, \_)$ | 24,086 | 2704 |

Figure 2: Information needed by *Ind* for $DB$ = {INSPEC, PSYCINFO} and $q$= *find author D. Knuth* $\wedge$ *title computer.*

selected databases to evaluate $q$ is then computed in the following way:

$$\begin{aligned} Chosen_{Ind}(q, DB) = \\ &\{db \in DB | \mathsf{ESize}_{Ind}(q, db) > 0 \wedge \\ &\mathsf{ESize}_{Ind}(q, db) = \max_{db' \in DB} \mathsf{ESize}_{Ind}(q, db')\} \quad (2) \end{aligned}$$

To illustrate these definitions, let $DB$ ={INSPEC, PSYCINFO} (INSPEC and PSYCINFO are databases that we will use in our experiments, see Section 4). Also, let $q$ =*find author D. Knuth* $\wedge$ *title computer.* Figure 2 shows the statistics available to *Ind*. From this, *Ind* computes: $\mathsf{ESize}_{Ind}(q, \text{ INSPEC }) = \frac{13 \times 24{,}086}{1{,}416{,}823} \simeq 0.22$. Incidentally, the actual result size of the query $q$ in INSPEC, $\mathsf{RSize}(q, \text{INSPEC})$, is one document.

Since "*D. Knuth*" is not an author in the PSYCINFO database, and due to the boolean semantics of the query representation, the result size of query $q$ in the PSYCINFO database must be zero. This agrees with what Equation 1 predicts: $\mathsf{ESize}_{Ind}(q, \text{PSYCINFO}) = \frac{0 \times 2704}{323{,}952} = 0$. This holds in general for boolean queries: if $freq(t_i, db) = 0$ for some $1 \leq i \leq n$, then

$$\mathsf{ESize}_{Ind}(q, db) = \mathsf{RSize}(q, db) = 0$$

where $q = find\ t_1 \wedge \ldots \wedge t_n$. As we have seen, when all frequencies are non-zero, $\mathsf{ESize}_{Ind}$ can differ from RSize. In [1] we analyze how well $\mathsf{ESize}_{Ind}$ approximates RSize.

To continue with our example, since $DB$ ={INSPEC, PSYCINFO}, and INSPEC is the only database with a non-zero result size estimate, as given by $\mathsf{ESize}_{Ind}$, it follows that $Chosen_{Ind}(q, DB)$ = {INSPEC}. So, *Ind* chooses the only database in the pair that might contain some matching document for $q$. In fact, since $\mathsf{RSize}(q, \text{INSPEC}) = 1$, *Ind* succeeds in selecting the only database that actually contains a document matching query $q$.

### 3.4 Evaluation criteria

Let $DB$ be a set of databases. In order to evaluate the *Ind* estimator, we need to compare its prediction against what actually is the "right subset" of $DB$ to query. There are different notions of what the right subset means (see [1]). In this paper we will just study one definition of right subset: $Best(q, DB)$[5], those databases that yield the most matching documents.

Ideally, a "relevant" document is one that would interest the user that issued the query. Unfortunately, we have no way to know this. One way to address this problem is to consider as relevant any documents matching the user's query. However, this does not necessarily solve the problem: For example, a database might contain a document written by a psychologist named *Knuth* on how *computers* can alienate people. This document may not be relevant to the issuer of the query "*find Knuth* $\wedge$ *computers.*" However, if we have no additional information on what relevant is, it is fair to simply look at databases with matching documents. Therefore, we define $Best(q, DB)$ to be the set of databases that have the highest number of documents matching query $q$. More formally,

$$\begin{aligned} Best(q, DB) \quad = \quad &\{db \in DB | \mathsf{RSize}(q, db) > 0 \wedge \quad (3) \\ &\mathsf{RSize}(q, db) = \max_{db' \in DB} \mathsf{RSize}(q, db')\} \end{aligned}$$

Once this set has been defined for a query $q$ and a database set $DB$, we can state different criteria to evaluate $Chosen_{Ind}(q, DB)$:

- *All-Best Search:* We are interested in searching all of the *Best* databases for $q$. By searching these databases we seek a compromise between two potentially conflicting goals: obtaining an exhaustive answer to $q$ (this would be guaranteed if we searched all of the databases containing matching documents, not only those containing the highest number of matching documents) and searching databases that would deliver a significant number of answers, to compensate for access costs, for example. Thus, we say that $Chosen_{Ind}$ satisfies criterion $C_{AB}$ if:

  $$C_{AB} : Best \subseteq Chosen_{Ind}$$

  So, we ensure that at least those databases having the highest payoff (i.e., the largest number of documents) are searched.

- *Only-Best Search:* We are less demanding than with $C_{AB}$: we are just interested in searching (some of) the best databases for $q$. Our goal is to get a sample of the documents that match the query $q$ (we might be missing some of these best databases), but we do not want to waste any time and resources by searching a non-optimal database. So, we say that $Chosen_{Ind}$ satisfies criterion $C_{OB}$ if:

[5] In general, we will drop the parameters of the functions when this will not lead to confusion. For example, we refer to $Best(q, DB)$ as $Best$, whenever $q$ and $DB$ are clear from the context.

$$C_{OB} : Chosen_{Ind} \subseteq Best$$

The set $Chosen_{Ind}$ will be said to *strictly satisfy* both criteria $C_{AB}$ and $C_{OB}$ if $Chosen_{Ind} = Best$.

Now, let $C$ be either of the criteria above and $Q$ be a fixed set of queries. Then,

$$Success(C, Ind) = 100 \times \frac{|\{q \in Q | Chosen_{Ind}(q, DB) \text{ satisfies } C\}|}{|Q|} \quad (4)$$

In other words, $Success(C, Ind)$ is the percentage of $Q$ queries for which *Ind* produced the "right answer" under criterion $C$.

Following notions analogous to those used in Statistics, we define the *Alpha* and the *Beta* errors of *Ind* for an evaluation criterion $C$ as follows:

$$Alpha(C, Ind) = 100 - Success(C, Ind) \quad (5)$$

$$Beta(C, Ind) = Success(C, Ind) - 100 \times \frac{|\{q \in Q | Chosen_{Ind}(q, DB) \text{ strictly satisfies } C\}|}{|Q|} \quad (6)$$

So, $Alpha(C, Ind)$ is the percentage of queries in $Q$ for which the estimator gives the "wrong answer," that is, the $Chosen_{Ind}$ set does not satisfy criterion $C$ at all. $Beta(C, Ind)$ measures the percentage of queries for which the estimator satisfies the criterion, but not strictly. For the *Beta* queries, the estimator yields a correct but "overly conservative" (for $C_{AB}$) or "overly narrow" (for $C_{OB}$) answer. For example, consider an estimator, $TRIV$, that would always produce $\emptyset$ as the value for $Chosen_{TRIV}$. $TRIV$ would have $Success(C_{OB}, TRIV) = 100$ (and $Alpha(C_{OB}, TRIV) = 0$). However, *Beta* has a high value for conservative estimators: $Beta(C_{OB}, TRIV)$ would be quite high.

The definitions of *Success*, *Alpha*, and *Beta* can be expressed in terms of the *precision* and *recall* parameters of information retrieval theory [21]. In [1] we explore this issue. For the sake of clarity, we present the results in terms of the *Success*, *Alpha*, and *Beta* parameters.

# 4 Experimental framework

In order to evaluate the performance of *Ind*, the estimator of Section 3.3[6], according to the criteria of Section 3.4, we performed experiments using query traces from the FOLIO library information retrieval system at Stanford University.

[6] We will refer indistinctively to both the estimator and its corresponding $Chosen_{Ind}(q, DB)$ set as satisfying the criteria of Section 3.4, for a query $q$ and a set of databases $DB$.

| Database | DBSize | Area |
|---|---|---|
| INSPEC | 1,416,823 | Physics, Elect. Eng., Computer Sc., etc. |
| 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 |

Figure 3: Summary of the characteristics of the six databases considered.

## 4.1 Databases and the INSPEC query trace

Stanford University provides on-campus access to its information retrieval system FOLIO from terminals in libraries and from workstations via `telnet` sessions. FOLIO gives access to several databases. Figure 3 summarizes some characteristics of the six databases chosen for our experiments. Six is a relatively small number, given our interest in exploring hundreds of databases. However, we were limited to a small number of databases by their accessibility and by the high cost of our experiments. Thus, our results will have to be taken with caution, indicative of the *potential* benefits of *GlOSS*.

A trace of all user commands for the INSPEC database was collected from 4/12/1993 to 4/25/1993. This set of commands contained 8392 queries. As discussed in Section 3.1, we only considered correctly formed "and" queries. Also, we only used queries involving just the indexes listed in Figure 9. The final set of queries, $TRACE_{INSPEC}$, has 6897 queries, or 82.19% of the original set.

## 4.2 Database frequency information construction

In order to perform our experiments, we evaluated each of the $TRACE_{INSPEC}$ queries in the six databases described in Figure 3. This gives the data we need to build the *Best* set for each of the queries.

Also, to build the database word-frequency information needed by *GlOSS* (Section 3.2) we evaluated, for each query of the form *find* $t_1 \wedge \ldots \wedge t_n$, the $n$ queries *find* $t_1, \ldots,$ *find* $t_n$ in each of the six databases. Note that the result size of the execution of *find* $t_i$ in database $db$ is equal to $freq(t_i, db)$ as defined in Section 3. This is exactly the information the *Ind* estimator needs to define $Chosen_{Ind}$, for each query in $TRACE_{INSPEC}$[7]. We should note that this is just the way we gathered the data in order to perform our experiments. An actual implementation of such a system would require that each database communicate the length of each inverted

[7] In fact, we are not retrieving all of the word frequencies, but only those that are needed for the queries in $TRACE_{INSPEC}$.

| | |
|---|---|
| Database set ($DB$) | {INSPEC, COMPENDEX, ABI, GEOREF, ERIC, PSYCINFO} |
| Estimator | *Ind* |
| Query set | $TRACE_{INSPEC}$ |
| Query sizes considered | All |
| *threshold* | 0 |

Figure 4: Basic configuration of the experiments.

list to *GlOSS*.

### 4.3 Configuration of the experiments

There are a number of parameters to our experiments. Figure 4 shows an assignment of values to these parameters that will determine the *basic configuration*. In later sections, some of these parameters will be changed, to produce alternative results. The *threshold* parameter will be defined in Section 6.4.

## 5 *Ind* results

In this section we evaluate *Ind* by first studying its ability to distinguish between two databases and then generalizing the experiments to include six databases.

### 5.1 Evaluating *Ind* over pairs of databases

In this section, we report some results for the basic configuration (Figure 4), but with $DB$, the set of available databases, set to just two databases. Figure 5 shows a matrix classifying the 6897 queries in $TRACE_{INSPEC}$ for the case $DB$ ={INSPEC, PSYCINFO}. The sum of all of the entries of the matrix equals 6897. Each row represents an outcome for *Best*. The first row, for instance, represents queries where INSPEC had the most matching documents (*Best* ={INSPEC}). On the other hand, each column represents the prediction made by *Ind*. For example, the number 5572 means that for 5572 of the queries in $TRACE_{INSPEC}$, *Best* ={INSPEC} and *Ind* correctly selected INSPEC as its prediction ($Chosen_{Ind}$ ={INSPEC}). In the same row, there were 42 other queries where *Ind* failed to pick the best database. So, from this row we see that for most of the queries (5614 out of 6897) INSPEC was the best database. This is not surprising, since the queries used in the experiments were originally issued by users to the INSPEC database. The values in the diagonal in Figure 5 indicate those queries for which *Ind* produced exactly the set of best databases as an answer (and so, criteria $C_{AB}$ and $C_{OB}$ were strictly satisfied). So, this was the case for 6328, or 91.75%, of the queries.

In Figure 5, $Chosen_{Ind} = \emptyset$ only if $Best = \emptyset$. From Equations 1 and 2, it follows that this relationship holds in general, that is, as long as there is at least one database that contains matching documents, $Chosen_{Ind}$ will be non-empty. (This will not hold

| *Best* | $Chosen_{Ind}$ {I} | {P} | {I, P} | $\emptyset$ |
|---|---|---|---|---|
| {I} | 5572 | 42 | 0 | 0 |
| {P} | 16 | 258 | 0 | 0 |
| {I, P} | 3 | 5 | 15 | 0 |
| $\emptyset$ | 462 | 41 | 0 | 483 |

Figure 5: Results corresponding to $DB$ = {INSPEC (I), PSYCINFO (P)}.

| Criteria | *Success* | *Alpha* | *Beta* | *Success* − *Beta* |
|---|---|---|---|---|
| $C_{AB}$ | 99.04 | 0.96 | 7.29 | 91.75 |
| $C_{OB}$ | 91.87 | 8.13 | 0.12 | 91.75 |

Figure 6: Evaluation criteria for $DB$={INSPEC, PSYCINFO}.

with the modification to *GlOSS* of Section 6.4.) Also, note that very few times (15) does *Ind* determine a tie between the two databases (and so, $Chosen_{Ind}$ consists of both databases), since it is very unlikely that $\mathsf{ESize}_{Ind}(q,$ INSPEC) will be exactly equal to $\mathsf{ESize}_{Ind}(q,$ PSYCINFO). With the current definition of $Chosen_{Ind}$, if for some query $q$ and databases $db_1$ and $db_2$ it is the case that, say, $\mathsf{ESize}_{Ind}(q, db_1) = 9$ and $\mathsf{ESize}_{Ind}(q, db_2) = 8.9$, then $Chosen_{Ind}(q, \{db_1, db_2\}) = \{db_1\}$. We might want in such a case to include $db_2$ also in $Chosen_{Ind}$. We address this issue in [1].

Figure 6 reports the values of *Success*, *Alpha*, and *Beta* corresponding to the results of Figure 5, for our two different criteria. Consider for example the first row of the matrix in Figure 6. This row corresponds to criterion $C_{AB}$ (see Section 3.4). From the table, we see that $Success(C_{AB}, Ind) = 99.04\%$. This means that in 99.04% of the cases *Ind* gave the correct answer, that is, the $Chosen_{Ind}$ set of databases included the *Best* set of databases we were after. In 0.96% of the queries ($Alpha(C_{AB}, Ind)$) we got the "wrong" answer, that is, the $Chosen_{Ind}$ set did not contain one of the best databases. Out of the successful cases, sometimes *Ind* gives exactly the set of best databases, while in other cases it gives a larger set. The value $Beta(C_{AB}, Ind) = 7.29\%$ tells us how many queries were in the latter case. Finally, $Success(C_{AB}, Ind) - Beta(C_{AB}, Ind) = 91.75\%$ gives the number of queries in the former case, or the percentage of queries classified in the diagonal of Figure 5, as explained above. The $Success(C_{OB}, Ind)$ value is high, showing that in most cases $Chosen_{Ind}$ consists only of "best" databases. Also, notice that for both criteria $C_{AB}$ and $C_{OB}$ the $Success - Beta$ entries are identical: for both criteria, $Success - Beta$ measures the fraction of queries for which $Chosen_{Ind} = Best$.

In [1] we report the results for all the pairs of databases that can be obtained from {INSPEC, COMPENDEX, ABI, GEOREF, ERIC, PSYCINFO}: in

| Criteria | *Success* | *Alpha* | *Beta* | *Success − Beta* |
|---|---|---|---|---|
| $C_{AB}$ | 88.95 | 11.05 | 6.89 | 82.06 |
| $C_{OB}$ | 84.38 | 15.62 | 2.32 | 82.06 |

Figure 7: Evaluation criteria for the basic configuration.

general, the more unrelated the subject domains of the two databases considered were, the better *Ind* behaved in distinguishing the databases.

### 5.2 Evaluating *Ind* over six databases

In this section we report some results for the basic six database configuration, as defined in Figure 4. Figure 7 summarizes the results corresponding to our evaluation criteria. This figure shows that the same phenomena described in Section 5.1 prevail, although in general the success rates are lower. Still, $Success(C_{AB}, Ind)$ is relatively high (88.95%), showing *Ind*'s ability to predict what the best databases are. Also, the *Success* figure for $C_{OB}$ is high (84.38%), making *Ind* useful for exploring *some* of the best databases. This is particularly significant for *Ind*: $Chosen_{Ind}(q, DB)$ will be non-empty as long as there is some database in $DB$ that might contain some document matching query $q$. Therefore, for 84.38% of the queries, *Ind* chooses databases that actually are among the best ones, provided there are any, what makes *Ind* particularly good for the $C_{OB}$ semantics.

Another interesting point is the fact that for only 96 out of the 6897 $TRACE_{INSPEC}$ queries does $Chosen_{Ind}$ consist of more than one database. Furthermore, 95 of these 96 queries are one-atomic-subquery queries, for which $Chosen_{Ind} = Best$ necessarily [8]. So, revisiting the results of Figure 7, for 88.95% of the $TRACE_{INSPEC}$ queries *Ind* chooses *all* of the best databases, while it picks more than one database for just 96 queries. Therefore, in most of the cases, not only does *Ind* narrow down the search space to just one database (out of the six available ones), but it also manages to select the best database when there is one.

## 6 *GlOSS*' storage requirements

In this section we study the space requirements of *GlOSS* and compare them with those of a full index of the set of databases. We also analyze the impact on the effectiveness of *Ind* of eliminating information on low frequency words from *GlOSS*.

[8] *Ind* chooses exactly the *Best* set for queries of size one: from Equation 1, if $t$ is an atomic subquery and $db$ a database, then $\mathsf{ESize}_{Ind}(\mathit{find}\ t, db) = \mathit{freq}(t, db)$, and so, $\mathsf{ESize}_{Ind}(\mathit{find}\ t, db) = \mathsf{RSize}(\mathit{find}\ t, db)$.

### 6.1 Eliminating the "subject" index

Before we compute the frequency information size, we will analyze the way the "subject" index is treated in the six databases we considered. In all of these databases, "subject" is a compound index, built from other "primitive" indexes. For example, in the INSPEC database, the "subject" index is constructed from the "title," "abstract," "thesaurus," "organization," and "other subjects" indexes: a query "*find subject computers*" is equivalent to the "or" query: "*find title computers* ∨ *abstract computers* ∨ *thesaurus computers* ∨ *organization computers* ∨ *other subjects computers.*"

All of the experiments we reported so far treated "subject" as a primitive index, as though *GlOSS* kept the entries corresponding to the "subject" field designation as part of the database frequency information. However, given that *GlOSS* has the entries for the constituent indexes from which the "subject" index is formed, we could attempt to *estimate* the entries corresponding to the "subject" index using the entries for the primitive indexes. This way, we can save space by not having to store entries for the "subject" index.

There are different ways to estimate *freq(subject w, db)*, given the primitive indexes $index_1$, $index_2$, ..., $index_n$ that compose the "subject" index in database *db*. One such way takes the maximum of the individual frequencies for the primitive indexes:

$$freq(subject\ w,\ db) \approx \max_{i=1,\ldots,n} freq(index_i\ w,\ db) \qquad (7)$$

Note that this estimate constitutes a lower bound for the actual value of *freq(subject w, db)*.

Figure 8 shows the results obtained for the basic configuration (Figure 4) but estimating the "subject" frequencies as in Equation 7, with one difference: only those indexes that actually appeared in $TRACE_{INSPEC}$ queries were considered. The other indexes are seldom used so it does not make sense for *GlOSS* to keep statistics on them. The indexes considered are the ones that are listed in Figure 9. For example, we simply ignored the "other subjects" index for the INSPEC database. The last column in Figure 8 shows the *Success* figures for the basic configuration, using the exact frequencies for the "subject" index: there is very little change in performance if we estimate the "subject" frequencies as in Equation 7 [9]. Therefore, when we compute the size of the *GlOSS* frequency information in the next section, we will assume that *GlOSS* does not store "subject" entries. Thus, we will consider only primitive indexes that appear in $TRACE_{INSPEC}$ queries.

[9] In [1] we explore an alternative estimate for the "subject" frequencies whose corresponding experimental results were very similar to those for the Equation 7 estimate.

| Criteria | *Success* | *Alpha* | *Beta* | *Success–Beta* | *Success* (Fig. 7) |
|---|---|---|---|---|---|
| $C_{AB}$ | 88.23 | 11.77 | 6.93 | 81.30 | 88.95 |
| $C_{OB}$ | 83.82 | 16.18 | 2.52 | 81.30 | 84.38 |

Figure 8: Evaluation criteria for the basic configuration, but estimating the "subject" frequencies as the *maximum* of the frequencies of the primitive indexes. The last column shows the *Success* values for the basic configuration, using the exact "subject" frequencies.

| Field Designator | Full Index # of postings | *GlOSS* (*threshold=0*) # 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 |

Figure 9: Characteristics of the database frequency information kept by *GlOSS* vs. those of a full index, for the INSPEC database.

### 6.2 Characteristics of the database frequency information and full indexes

As explained in Section 3.2, *GlOSS* needs to keep, for each database, the number of documents that satisfy each possible keyword field-designation pair. Figure 9 was generated using information of the corresponding INSPEC indexes obtained 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. For example, there are $311,632$ different author surnames 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.

| Size of | Full Index | *GlOSS*/*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 |

Figure 10: Estimated storage costs of a full index vs. the *GlOSS* frequency information for the INSPEC database.

### 6.3 Storage cost estimates

In the following, we will roughly estimate the space requirements of a full index vs. those of the frequency information kept by *GlOSS*, for the INSPEC database. The figures we will produce should be taken just as an indication of the relative order of magnitude of the corresponding requirements.

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 *GlOSS* 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 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 [10], including only those indexes that appear in $TRACE_{INSPEC}$ queries, consists of $819,437$ words. If we dedicate four bytes to store each keyword (see [1]), around $4 \times 819,437$ bytes, or 3.13 MBytes are needed to store the INSPEC vocabulary. This 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 *GlOSS* 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

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

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 3):

$$G = \frac{\sum_{db \in DB} DBSize(db)}{DBSize(\text{INSPEC})} \approx 4.12$$

where $DB$ = {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 *GlOSS* 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 *GlOSS* and a full index. Note that the *GlOSS* frequency information is only 2.15% the size of the full index. This 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 this 10% reduction "benefits" both *GlOSS* and the full index case, the impact on *GlOSS* is higher, since the vocabulary size is a much larger fraction of the total storage needed by *GlOSS* 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 *GlOSS*: around 22.29 MBytes would suffice to keep the word frequencies for the six databases we studied.

### 6.4 Pruning the word-frequency information

To further reduce the amount of information that we keep about each database, we introduce the notion of a *threshold*. If a database *db* has fewer than *threshold* documents with a given keyword-field pair $t$, then *GlOSS* will not keep this information. Therefore, *GlOSS* will assume that $freq(t, db)$ is zero whenever this data is needed.

| Size of | Full index | *GlOSS/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 |
| $Success(C_{AB}, \_)$ | 100 | 88.23 |
| $Success(C_{OB}, \_)$ | 100 | 83.82 |
| $Success(C_{OB}, \_) - Beta(C_{OB}, \_)$ | 100 | 81.30 |

Figure 11: Storage estimates for *GlOSS* and a full index for the six databases. The entries for *GlOSS* in the last three rows correspond to the basic configuration, but estimating the "subject" frequencies as the maximum of the frequencies of the primitive indexes.

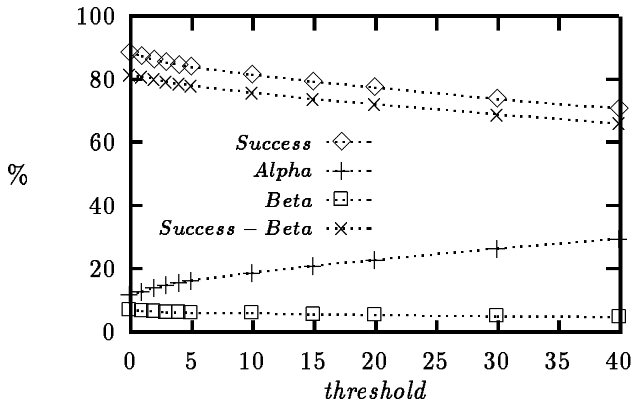


Figure 12: Criterion $C_{AB}$, for different values of *threshold*. The "subject" entries are estimated as the maximum of the entries corresponding to the primitive indexes.

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

To see if *Ind*'s performance deteriorates by the use of this *threshold*, Figures 12 and 13 show some results for different values of *threshold*, for the basic configuration, but estimating the "subject" index entries as in Equation 7. These figures show that the performance for the different criteria is only slightly sensitive to (small) increases in *threshold*. Ironically, the *Success* values

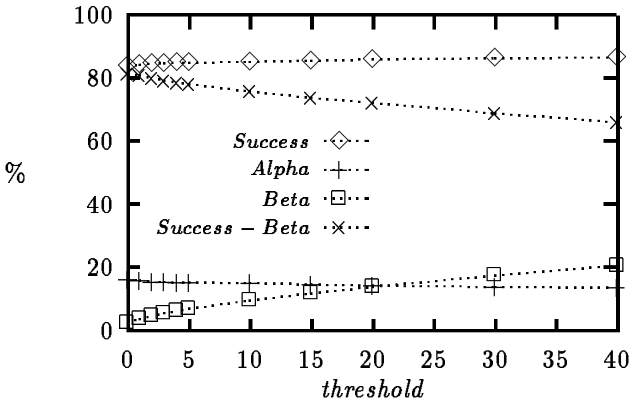


Figure 13: Criterion $C_{OB}$, for different values of *threshold*. The "subject" entries are estimated as the maximum of the entries corresponding to the primitive indexes.

for criterion $C_{OB}$ tend to improve for higher values of *threshold*. The reason for this is that $Chosen_{Ind}$ does not include databases with $\mathsf{ESize}_{Ind} = 0$. By increasing *threshold*, the number of such databases will presumably increase, thus making $Chosen_{Ind}$ smaller, and more likely to satisfy $C_{OB} : Chosen_{Ind} \subseteq Best$.

The reason for introducing *thresholds* is to have to store less information for the estimator. Figure 14 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 *GlOSS* 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. Therefore, the size of the *GlOSS* frequency information can be substantially reduced beyond the already small size estimated in Figure 11.

## 7 Conclusions

In this paper we presented *GlOSS*, a solution to the text database discovery problem. We also developed a formal framework for this problem, together with different semantics to answer a user query. We used this framework to evaluate the effectiveness of *Ind*, an estimator for *GlOSS*. The experimental results we obtained, although involving only six databases, are encouraging: the *Success* values for our evaluation criteria are higher than 84%.

The storage cost of *GlOSS* is relatively low (see Figure 11). A rough estimate suggested that 22.29 MBytes would be enough to keep all the data needed for the six databases we studied. Given this low space requirement, *GlOSS* itself can be replicated to increase its availability. Furthermore, we considered a variation of *GlOSS* to reduce its storage cost: for example, with only under 2% loss in *Success* values, we reduce the number of entries kept by *GlOSS* by about half.

Because of space limitations, we have only included in this paper some of the results of our experiments. In [1] we describe additional experiments we conducted. In particular, we believe that our results are independent of the query trace we used, since we obtained very similar results using a different query trace. Also, we analyzed two other estimators for *GlOSS*, namely *Min* and *Bin*. *Min* is an estimator built assuming that the keywords that appear together in a user query are strongly correlated (as opposed to *Ind*'s "independence" assumption). Surprisingly, the results we obtained for *Min* are very similar to those for *Ind*. *Bin* is an estimator aimed at addressing the Exhaustive Search semantics briefly described in the Introduction. We analyze this semantics, together with the others discussed in the Introduction, in [1].

Our approach could also deal with information servers that would charge for their use. Since we are selecting what databases to search according to a quantitative measure of their "goodness" for a given query ($\mathsf{ESize}_{Ind}$), we could easily incorporate this cost factor into the computation of $\mathsf{ESize}_{Ind}$ so that, for example, given two equally promising databases, a higher value would be assigned to the least expensive of the two.

We are currently implementing a *GlOSS* server that will keep information on databases having WAIS [7] indexes. These databases can correspond to WAIS servers, or to World Wide Web servers [4] with WAIS indexes, for example. The *GlOSS* server will be available through World Wide Web.

## 8 Acknowledgments

This research was sponsored by the Advanced Research Projects Agency (ARPA) of the Department of Defense under Grant No. MDA972-92-J-1029 with the Corporation for National Research Initiatives (CNRI). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of ARPA, the U.S. Government or CNRI. This work was supported by an equipment grant from Digital Equipment Corporation.
We would also like to thank Jim Davis, Daniela Rus, and the anonymous referees for their helpful comments.

## References

[1] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. The efficacy of *GlOSS* for the text database

| 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 |
| *Success*($C_{AB}$, Ind) | 88.23 | 87.12 | 86.07 | 85.28 | 84.44 | 83.82 |
| *Success*($C_{OB}$, Ind) | 83.82 | 84.24 | 84.53 | 84.65 | 84.76 | 84.79 |
| *Success*($C_{OB}$, Ind) − *Beta*($C_{OB}$, Ind) | 81.30 | 80.64 | 79.85 | 79.15 | 78.44 | 77.85 |

Figure 14: Number of entries left for the different thresholds and field designators in the INSPEC database. The last three rows correspond to the basic configuration, but estimating the "subject" frequencies as the maximum of the frequencies of the primitive indexes.


discovery problem. Technical Report STAN-CS-TN-93-002, Stanford University, November 1993. Available by anonymous `ftp` from `db.stanford.edu` in `/pub/gravano/1993/stan.cs.tn.93.002.ps`.

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

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

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

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

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

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

[8] Jim Fullton, Archie Warnock, et al. Release notes for freeWAIS 0.2, October 1993.

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

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

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

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

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

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

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

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

[17] Ran Giladi and Peretz Shoval. Routing queries in a network of databases driven by a meta knowledge-base. In *Proceedings of the International Workshop on Next Generation Information Technologies and Systems*, June 1993.

[18] Mark A. Sheldon, Andrzej Duda, Ron Weiss, James W. O'Toole, and David K. Gifford. A content routing system for distributed information servers. To appear in EDBT '94.

[19] Alice Y. Chamis. Selection of online databases using switching vocabularies. *Journal of the American Society for Information Science*, 39(3), 1988.

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

[21] Gerard Salton and Chris Buckley. Parallel text search methods. *Communications of the ACM*, 31(2), February 1988.