Relevant Document Distribution Estimation Method for Resource Selection

Luo Si and Jamie Callan
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
lsi@cs.cmu.edu, callan@cs.cmu.edu

ABSTRACT

Prior research under a variety of conditions has shown the CORI algorithm to be one of the most effective resource selection algorithms, but the range of database sizes studied was not large. This paper shows that the CORI algorithm does not do well in environments with a mix of "small" and "very large" databases. A new resource selection algorithm is proposed that uses information about database sizes as well as database contents. We also show how to acquire database size estimates in uncooperative environments as an extension of the query-based sampling used to acquire resource descriptions. Experiments demonstrate that the database size estimates are more accurate for large databases than estimates produced by a competing method; the new resource ranking algorithm is always at least as effective as the CORI algorithm; and the new algorithm results in better document rankings than the CORI algorithm.

Categories & Subject Descriptors:

H.3.3 [**Information Search and Retrieval]:**

General Terms: Algorithms

Keywords: **Resource Selection**

1. INTRODUCTION

Distributed information retrieval, also known as *federated search*, is ad-hoc search in environments containing multiple, possibly many, text databases [1]. Distributed information retrieval includes three sub-problems: i) acquiring information about the contents of each database (*resource representation*) [1,6], ii) ranking the resources and selecting a small number of them for a given query (*resource ranking*) [1,3,5,7,8,12], and iii) merging the results returned from the selected databases into a single ranked list before presenting it to the end user (*result-merging*) [1,4,13]. Early distributed IR research focused on *cooperative* environments in which search engines could be relied upon to provide corpus vocabulary, corpus statistics, and search engine characteristics when requested to do so. Recent research also addresses *uncooperative* environments in which search engines only run queries and return documents.

Most resource ranking algorithms rank by how well database contents appear to match a query, so resource descriptions have tended to emphasize content [1]. Prior research suggests that it is important to compensate for database size when assessing similarity [12,7], but it has been unclear how to estimate database sizes accurately in uncooperative environments.

This paper presents ReDDE, a new resource-ranking algorithm that explicitly tries to estimate the distribution of relevant documents across the set of available databases. The ReDDE algorithm considers both content similarity and database size when making its estimates. A new algorithm for estimating database sizes in uncooperative environments is also presented. Previous research showed that improved resource selection is *correlated* with improved document rankings; this paper shows that better resource selection does not *always* produce better document rankings. An analysis of this contradiction leads to a more robust version of the ReDDE algorithm.

The next section discusses prior research. Section 3 describes a new method of estimating database sizes. Section 4 explains the new ReDDE resource selection algorithm. Section 5 discusses the subtle relationship between resource selection accuracy and document retrieval accuracy, and proposes the modified version of the ReDDE algorithm. Section 6 describes experimental methodology. Sections 7, 8 and 9 present experiment results for database size estimation, resource selection and document retrieval. Section 10 concludes.

2. PREVIOUS WORK

Our research interest is uncooperative environments, such as the Internet, in which resource providers provide basic services (e.g., running queries, retrieving documents), but don't provide detailed information about their resources. In uncooperative environments perhaps the best method of acquiring resource descriptions is query-based sampling [1], in which a resource description is constructed by sampling database contents via the normal process of running queries and retrieving documents. Query-based sampling has been shown to acquire accurate unigram resource descriptions using a small number of queries (e.g., 75) and a small number of documents (e.g., 300).

Database size is an important component of a resource description, but there has been limited prior research on how to estimate it in uncooperative environments. Liu and Yu proposed using a basic capture-recapture methodology to estimate the size of a database [11]. Capture-recapture assumes that there are two (or more) independent samples from a population. Let N be the population size, A the event that an item is included in the first sample, which is of size n1, B the event that an item is included in the second sample, which is of size n2, and m2 the number of items that appeared in both samples. The probabilities of events A and B, and the relationship between them, are shown below.

SIGIR '03, July 28-Aug 1, 2003, Toronto, Canada.

$$P(A) = \frac{n_1}{N} \quad (1)$$

$$P(B) = \frac{n_2}{N} \quad (2)$$

$$P(A \mid B) = \frac{m_2}{n_2} \quad (3)$$

The two samples are assumed to be independent, so:

$$P(A \mid B) = P(A) \quad (4)$$

Thus, the population size is estimated as:

$$\hat{N} = \frac{n_1 n_2}{m_2} \quad (5)$$

Liu and Yu used it to estimate database sizes by randomly sending queries to a database and sampling from the document ids that were returned. They reported a rather accurate ability to estimate database sizes [11], but the experimental methodology might be considered unrealistic. For example, when estimating the size of a database containing 300,000 documents, the sampling procedure used 2,000 queries and expected to receive a ranked list of 1,000 document ids per query, for a total of 2,000,000 (non-unique) document ids examined. This cost might be considered excessive. Our goal is a procedure that estimates database sizes at a far lower cost.

There is a large body of prior research on resource selection (e.g., [1,3,5,7,8,12]). Space limitations preclude discussing it all, so we restrict our attention to a few that have been studied often or recently in prior research.

gGlOSS [6] is based on the vector space model. It represents a database by a vector that is composed of the document frequencies of different words in the databases. Ipeirotis's Hierarchical Database Sampling and Selection algorithm [8] used information from the search engine to get document frequencies for some words and estimated the document frequencies of other words by using Mandelbrot's law. The document frequency information was used as a part of the database description to build a hierarchical structure for databases. They showed that the hierarchical structure with extra information had a better retrieval performance than the CORI resource selection algorithm, but did not explicitly evaluate resource selection performance. D'Souza and Thom's n-term indexing method [5] represents each resource by a set of document surrogates, with one surrogate per original document. They assume that they can access the content of each document in every database, which is not the case in our work.

The CORI resource selection algorithm [1,2] represents each database using the words it contains, their frequencies, and a small number of corpus statistics. Prior research indicates that the CORI algorithm is one of the most stable and effective resource algorithms (e.g., [7,12]) and it was used as a baseline in the research reported here. Resource ranking is done using a Bayesian inference network and an adaptation of the Okapi term frequency normalization [15]. Details can be found in [1,2].

The last step of distributed information retrieval is merging ranked lists produced by different search engines. It is usually treated as a problem of transforming database-specific document scores into database-independent document scores. The CORI merge algorithm [1] is a heuristic linear combination of the database score and the document score. Calv, et al., [4] used logistic regression and relevance information to learn merging models. The Semi-Supervised Learning algorithm [13] uses the documents acquired by query-based sampling as training data and linear regression to learn merging models.

3. DATABASE SIZE ESTIMATION

Database size estimates are desirable because people often ask about size when presented with a new database, and because resource selection errors are sometimes due to weaknesses in normalizing the term frequency statistics obtained from databases of varying sizes. Database size can be expressed in different ways, such as the size of the vocabulary, the number of word occurrences, and the number of documents. In this paper we define database size to be the number of documents. This metric can be easily converted to the other metrics if needed.

3.1 The Sample-Resample Method

The capture-recapture method of estimating database size [11] requires many interactions with a database. Our goal is an algorithm that requires few interactions, preferably one that can reuse information that is already acquired for other reasons.

We start by assuming that resource descriptions are created by query-based sampling, as in [13]. We assume that each resource description lists the number of documents sampled, the terms contained in sampled documents, and the number of sampled documents containing each term.

We also assume that the search engine indicates the number of documents that match a query. The number of matching documents is a common part of search engine user interfaces, even in uncooperative environments. Even if the search engine only approximates the number of documents that match the query, that number is an important clue to database size.

The *sample-resample* method of estimating database size is as follows. A term from the database's resource description is picked randomly and submitted to the database as a single-term query (*resampling*); the database returns the number of documents that match this one-term query (the *document frequency* of the term) and the ids of a few top-ranked documents, which are discarded. Let Cj be the database, and Cj_samp be the documents sampled from the database when the resource description was created. Let N_cj be the (unknown) size of Cj, and N_cj_samp be the size of Cj_samp. Let qi be the query term selected from the resource description for Cj. Let df_qicj be the number of documents in Cj that contain qi (returned by the search engine) and df_qicj_samp be the number of documents in Cj_samp that contain qi.

The event that a document sampled from the database contains term qi is denoted as A. The event that a document from the database contains the qi is denoted as B. The probabilities of these events can be calculated as shown below.

$$P(A) = \frac{df_{q_i c_{j\_samp}}}{N_{c_{j\_samp}}} \quad (6)$$

$$P(B) = \frac{df_{q_i c_j}}{N_{C_j}} \quad (7)$$

If we assume that the documents sampled from the database are a good representation of the whole database, then P(A) ≈ P(B), and N can be estimated as shown in Equation 8.

$$\hat{N}_{C_j} = \frac{df_{q_i c_j} * N_{c_j\_samp}}{df_{q_i c_j\_samp}} \quad (8)$$

Additional estimates of the database size are acquired by sending additional one-term queries to the database. An estimate based on the mean of the individual estimates reduces variance.

3.2 Database Size Evaluation Metrics

Liu and Yu [11] evaluated the accuracy of database size estimates using percentage error, which we call *absolute error ratio* (AER) in this paper. Let N* denote the actual database size and N the estimate. AER is calculated as shown below.

$$AER = \frac{\left|N - N^*\right|}{N^*} \quad (9)$$

When we evaluate a set of estimates for a set of databases, we calculate the mean absolute error ratio (MAER).

3.3 Database Size Estimation Costs

A fair comparison of algorithms must consider their costs. The significant cost for the capture-recapture and sample-resample methods is the number of interactions with the search engine.

Liu and Yu assumed that uncooperative databases would return ranked lists of up to 1,000 document ids [11]. This assumption is not true in some environments. For example, AltaVista and Google initially return only the top 10 or 20 document ids. If we assume the search engine returns document ids in pages of 20, then 50 interactions is required to obtain a sample of 1,000 ids.

A slight adjustment to Liu and Yu's original capture-recapture method reduces its cost. Only one document id from each sample is used by the capture-recapture method. If we assume that the method decides ahead of time which rank to take a sample from, the number of interactions can be reduced. If the search engine allows a particular range of the search results to be selected then only 1 interaction per sample is required. If the search engine requires that the list be scanned sequentially from the beginning, in pages containing 20 document ids each, then 25 interactions is required, on average, to obtain one sample.

The primary cost of the new sample-resample algorithm is the queries used to resample the document frequencies of a few terms. Each resample query requires one database interaction. The experiments reported in this paper used 5 resample queries per database. The sample-resample method also requires access to the resource descriptions that support resource selection.

4. RESOUCE SELECTION

The goal of resource selection is to select a small set of resources that contain a lot of relevant documents. If the distribution of relevant documents across the different databases were known, databases could be ranked by the number of relevant documents they contain; such a ranking is called a relevance based ranking (RBR) [1,7]. Typically, and in the work reported here, resource ranking algorithms are evaluated by how closely they approximate a relevance-based ranking.

The number of documents relevant to query q in database Cj is estimated as:

$$\hat{\text{Rel}}\_q(j) = \sum_{d_i \in C_j} P(rel \mid d_i) * P(d_i \mid C_j) * N_{C_j} \quad (10)$$

where Ncj is the number of documents in Cj; we can substitute the estimated database size $\hat{N}_{C_j}$ for Ncj. For the probabilities P (di | Cj), if we have downloaded all the documents from Cj and built a complete resource description for it, these probabilities will be 1/Ncj. For the sampled resource description, as long as the sampled resource description is representative, the number of relevant documents can be estimated as follows:

$$\hat{\text{Rel}}\_q(j) = \sum_{d_i \in C_{j\_samp}} P(rel \mid d_i) * \frac{1}{N_{C_j\_samp}} * \hat{N}_{C_j} \quad (11)$$

where Ncj_samp is the number of documents sampled from Cj.

The only item left to estimate is P (rel | di), the probability of relevance given a specific document. Calculating this probability is the goal of most probabilistic retrieval models, and is generally viewed as a difficult problem; we do not solve it in this research.

Instead, we define as a reference the *centralized complete database*, which is the union of all of the individual databases available in the distributed IR environment. We define P (rel | di), the probability of relevance given a document, as the probability of relevance given the document rank when the centralized complete database is searched by an effective retrieval method. This probability distribution is modeled by a step function, which means that for the documents at the top of the ranked list the probabilities are a positive constant, and for all other documents they are 0. Although this approximation of the relevance probability is rather rough, it is similar to the modeling of relevance by most automatic relevance feedback methods. Probability of relevance is modeled formally as:

$$P(rel \mid d_i) = \begin{cases} C_q & if\ \ Rank\_central(d_i) < ratio * \hat{N}_{all} \\ 0 & otherwise \end{cases} \quad (12)$$

where rank_central (di) is the rank of document di in the centralized complete database, and ratio is a threshold. This threshold indicates how the algorithm focuses attention on different parts of the centralized complete DB ranking. In our experiments, the ratio was set to 0.003, which is equivalent to considering the top 3,000 documents in a database containing 1,000,000 documents. $\hat{N}_{all}$ is the estimated total number of documents in the centralized complete database. Cq is a query-dependent constant. Although no single setting of the ratio parameter is optimal for every testbed, experiments (not reported here) show that the ReDDE algorithm is effective across a wide range of ratio settings (e.g., 0.002 to 0.005).

A centralized complete database is not available. However, a centralized *sample* database is easily available; it contains the

documents obtained by query-based sampling when database resource descriptions were constructed. The centralized sample database is a representative subset of the centralized complete database. Prior research showed that a centralized sample database is useful when normalizing document scores during result merging [13]. We use it here for resource ranking.

The query is submitted to the centralized sample database; a document ranking is returned. Given representative resource descriptions and database size estimates we can estimate how documents from the different databases would construct a ranked list if the centralized complete database existed and were searched. In particular, the rank is calculated as follows:

$$Rank\_Central(di) = \sum_{\substack{d_j \\ Rank\_Samp(dj)< \\ Rank\_Samp(di)}} \frac{\hat{N}_{c(dj)}}{N_{c(dj)\_samp}} \qquad (13)$$

Plugging Equation 12 and 13 into Equation 11, the values of $\hat{Rel}\_q(j)$ can be calculated. These values still contain a query dependant constant Cq, which comes from Equation 12. The useful statistic is the distribution of relevant documents in different databases. That information is sufficient to rank the databases. The estimated distribution can be calculated by normalizing these values from Equation 11, as shown below.

$$Dist\_\hat{Rel}\_q(j) = \frac{\hat{Rel}\_q(j)}{\sum_i \hat{Rel}\_q(i)} \qquad (14)$$

Equation 14 provides the computable distribution, without any constants. The databases can now be ranked by the estimated percentage of relevant documents they contain.

The experiments that test the effectiveness of this method are described in Section 8.

5. RETRIEVAL PERFORMANCE

Generally the effectiveness of a distributed information retrieval system is not evaluated by the Precision at Recall points metric (e.g., "11-point Recall Precision"). Only a subset of the databases is selected, so it is usually impossible to retrieve all of the relevant documents. Precision at specified document ranks is often used, particularly for interactive retrieval where someone may only look at the first several screens of results.

Suppose the goal of a distributed information retrieval system is to maximize Precision within the top-ranked 100 documents. The goal of the resource selection algorithm is to select a small number of databases that contain a large number of relevant documents. Improved resource selection usually produces improved retrieval accuracy [12], but not always. This was also observed in [3].

If a centralized complete database were accessible, a search algorithm would return the top 100 documents in the ranked list; this result set is only a tiny percentage of all the documents. The ReDDE algorithm evaluates a much larger percentage of the centralized complete database. For example, if the ratio (Equation 12) is 0.003 and the total testbed size is 1,000,000 documents, although the goal is to retrieve 100 relevant documents, the resource selection algorithm essentially attempts to maximize the number of relevant documents that would rank in the top 3,000 in the centralized complete database. Optimizing the number of relevant documents in the top 100 and top 3,000 retrieved documents is correlated, but not identical.

One could decrease the ratio used by the ReDDE resource selection algorithm. However, decisions are based on only a small number of sampled documents; using a small ratio would cause very few databases to have nonzero estimates. A better solution is to use two ratios. We call this the *modified ReDDE algorithm.* Databases that have large enough estimation values with the smaller ratio are sorted by these values. All other databases are sorted by estimation values created from a larger ratio. Thus for every database there are two estimation values (DistRel_r1j, DistRel_r2j), which are calculated by Equation 14 using two different ratios r1 and r2. In our experiments, r1 was empirically set to 0.0005 and r2 was set to 0.003. The procedure can be formalized as follows:

1. First rank all the databases that have DistRel_r1j >= backoff_Thres
2. For all the other databases rank them with the values DistRel_r2j

where the backoff_Thres is the backoff threshold. Backoff_Thres and was set to 0.1 in our experiments.

6. EXPERIMENT DATA

The database size estimation and ReDDE resource selection algorithms were tested on a variety of testbeds (Table 1).

Trec123-100col-bysource: 100 databases created from TREC CDs 1, 2 and 3. They are organized by source and publication date [1,13], and are somewhat heterogeneous. The sizes of the databases are not skewed.

Trec4-kmeans: 100 databases created from TREC 4 data. A k-means clustering algorithm was used to organize the databases by topic [14], so the databases are homogenous and the word distributions are very skewed. The sizes of the databases are moderately skewed.

In order to show the effects of database sizes on database size estimation, resource selection and document retrieval performance in environments containing many "small" DBs and a few "very large" DBs, an additional set of testbeds was built from the trec123-100col-bysource collection. In these testbeds there are many "small" DBs and a few "very large" DBs. Each new testbed contained 2 databases that are about an order of magnitude larger than other databases in the testbed Different testbeds were designed to test effectiveness with different types of "very large" databases: "representative" databases (Trec123-2ldb-60col), "relevant" databases (Trec123-AP-WSJ-60col) and "nonrelevant" databases (Trec123-FR-DOE-81col).

Trec123-2ldb-60col ("representative"): The databases in Trec123-100col-bysource were sorted alphabetically. Every fifth database, starting with the first, was collapsed into one large "representative" database called LDB1. Every fifth database, starting with the second, was collapsed into a large database

Table 1: Summary statistics for three distributed IR testbeds.

Testbed	Size (GB)	Num of documents (x 1000)			Size (MB)		
		Min	Avg	Max	Min	Avg	Max
Trec123-100col	3.2	0.7	10.8	39.7	28	32	42
Trec4-kmeans	2.0	0.3	5.7	82.7	4	20	249
Trec123-10 col	3.2	17.6	107.8	263.2	300	320	378

Table 2: Summary statistics for the "very large" databases.

Collection	Num of documents (x 1000)	Size (MB)
LDB1	231.3	665
LDB2	199.7	667
APall	242.9	764
WSJall	173.3	533
FRall	45.8	492
DOEall	226.1	194

Table 3: Query set statistics.

Collections	TREC Topic Set	TREC Topic Field	Average Length (Words)
Trec123	51-100	Title	3
Trec4	201-250	Description	7

called LDB2. The other 60 databases were left unchanged. LDB1 and LDB2 are about 20 times larger than the other databases but have about the same density of relevant documents as the other databases (Table 2).

Trec123-AP-WSJ-60col ("relevant"): The 24 Associated Press collections in the trec123-100col-bysource testbed were collapsed into a single large APall database. The 16 Wall Street Journal collections were collapsed into a single large WSJall collection (Table 2). The other 60 collections were unchanged. The APall and WSJall collections are much larger than the other databases, and they also have a higher density of documents relevant to TREC queries than the other 60 collections. Most relevant documents are contained in these two large databases.

Trec123-FR-DOE-81col ("nonrelevant"): The 13 Federal Register collections in the trec123-100col-bysource testbed were collapsed into a single large FRall collection. The 6 Department of Energy collections were collapsed into a single large DOEall collection (Table 2). The other 81 collections were unchanged. The FRall and DOEall are much larger than the other databases, but have a much lower density of relevant documents.

Trec123-10col: This testbed was created for testing the effectiveness of database size estimation algorithms on large databases (Table 1). The databases in trec123-100col-bysource were sorted alphabetically. Every tenth databases, starting with the first, was combined to create the first new database. Every tenth database, starting with the second, was combined to create the second new database. And so on. Altogether there are ten collections.

50 queries were created from the title fields of TREC topics 51-100 for the trec123 testbed and another 50 queries were created from the description fields of TREC topics 201-250 for the trec4-kmeans testbed (Table 3).

7. EXPERIMENT RESULTS: DATABASE SIZE ESTIMATION

The sample-resample database size estimation algorithm was evaluated on two testbeds: trec123-100col and trec123-10col. The first testbed contains 100 small databases and the second testbed contains 10 large databases (Table 1). The capture-recapture algorithm was used as a baseline.

The cost of capture-recapture and sample-resample is measured in the number of remote search engine interactions required to make an estimate. Both capture-recapture and sample-resample send queries to a search engine and get some information returned; for capture-recapture it is a page of 20 document ids; for sample-resample it is the number of matching documents. We consider these costs equivalent, i.e., one search engine interaction. Sample-resample also assumes access to a database resource description, which capture-recapture does not. Ordinarily this cost (about 80 queries and 300 document downloads [1]) would be allocated to resource selection, but to be completely fair, for this comparison we allocate it to the sample-resample algorithm. The total cost of the sample-resample algorithm is thus 385 search engine interactions: 5 sample-resample queries plus 380 interactions to build a resource description. Both algorithms are allotted 385 search engine allocations in the experiments reported below.

The cost of a capture-recapture algorithm is affected strongly by the type of ranked list access supported by the search engine (Section 3.3). The algorithm is most efficient when the search engine allows direct access to a specified section of the ranked list; we call this the "Direct" variant. If the search engine requires that ranked-list results be accessed sequentially in blocks of 20 ids, the capture-recapture algorithm would only be able to obtain about 15 samples, which is too few for an accurate estimate. In this case we use a variant that makes its choice from the first block of 20 ids; we call this the "Top" variant.

The basic capture-recapture algorithm [11] considered just one document id per sample. However, it acquires 20 document ids, so we examined the effects on accuracy of using just 1 or all 20 of the document ids returned by the search engine.

Each of the capture-recapture variants was allowed to send about 385 queries to the database; document ids gotten in the first half of the queries were the first sample; document ids gotten in the second half of the queries were the second sample.

For the sample-resample experiments, a resource description was created using query-based sampling in which randomly selected one-term queries were submitted to the search engine and the top 4 documents per query were downloaded until 300 documents had been downloaded. This process required about 80 queries. Only 5 resampling queries were used.

The experimental results are summarized in Table 4. Sample-resample was more accurate than all of the capture-recapture methods on the trec123-10col testbed, which contains larger databases (lower values are desired for the MAER metric). Sample-resample was more accurate than all but the "Direct All" variant of the capture-recapture methods on the trec123-

Table 4: Accuracy of the database size estimation methods on two testbeds in mean-average error ratio (MAER) metric.

Method	Trec123-100Col	Trec123-10Col
Cap-Recap Top1	0.729	0.943
Cap-Recap TopAll	0.377	0.849
Cap-Recap Direct1	0.566	0.845
Cap-Recap DirectAll	0.182	0.404
Sample-Resample	0.232	0.299

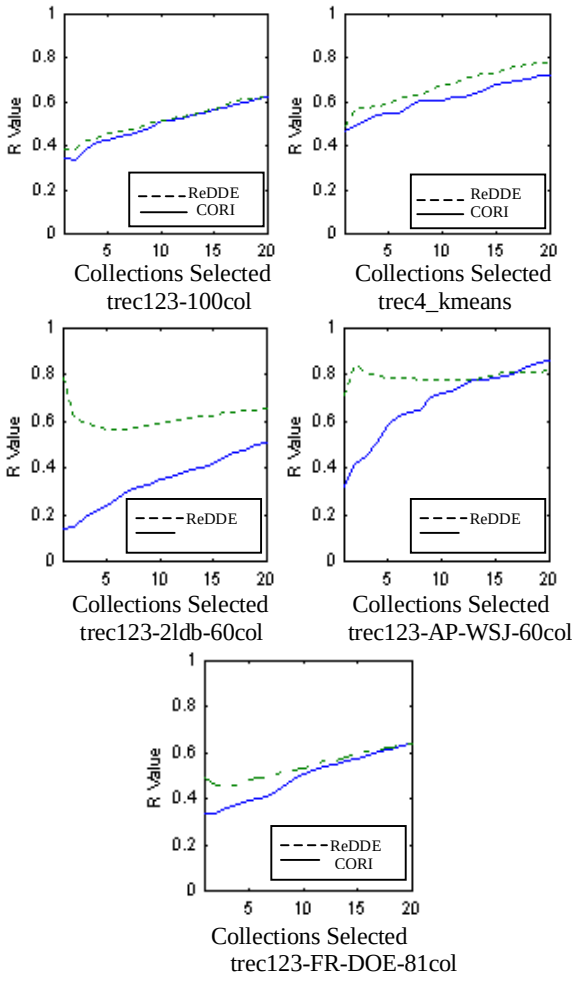

Figure 1. Resource selection accuracy on the five testbeds.

100col testbed, which contains smaller databases. Direct access to a desired section of the ranked list greatly improves the accuracy of the capture-recapture method, as does using all 20 of the document ids in a returned section of the ranked list.

Both methods tended to underestimate database size. The errors were due to the different assumptions the methods make. Capture-recapture makes an independence assumption (Equation 4). However, documents with more words, and with more diverse words, are more likely to be retrieved by different queries, so that some document ids in sample A are more likely to appear again in sample B. The sample-resample method assumes that a term's density in sampled documents is a good approximation of its density in the complete database. However the complete database contains a larger (unseen) vocabulary, so the percentage of documents that contain a sampled word tends to be overestimated. Smoothing might correct this problem.

Although the capture-recapture algorithm can be better than the sample-resample algorithm when databases are small, its success depends on a stronger assumption, i.e., that the search engine supports direct access to specific segments of a ranked list. The sample-resample algorithm makes no such assumption, so it can be used in a wider range of environments; it is also more accurate when databases are large.

8. EXPERIMENTAL RESULTS: RESOURCE SELECTION

The experiments evaluating the ReDDE resource selection algorithm were done on five testbeds (Section 6). These testbeds cover a wide range of conditions: relatively uniform sizes and content distribution (trec123-100col), relatively uniform sizes and skewed content distribution (trec4-kmeans), bimodal size distribution and a relatively uniform density of relevant documents (representative), bimodal size distribution and a much higher density of relevant documents in the large databases (relevant), and bimodal size distribution and a much lower density of relevant documents in the large databases (nonrelevant).

Database resource descriptions were obtained by query-based sampling, the same procedure as described in Section 7. The sample-resample method was used to estimate database sizes.

The recall metric R_n is commonly used to compare different resource selection algorithms [1,7]. Let B denote a baseline ranking, which is often the Relevance-Based Ranking, and E a ranking provided by a resource selection algorithm. Let Bi and Ei denote the number of relevant documents in the i'th ranked database of B or E. The metric R_n is defined as shown below.

$$R_k = \frac{\sum_{i=1}^{k} E_i}{\sum_{i=1}^{k} B_i} \qquad (15)$$

This metric measures the percentage difference between the estimated ranking and the relevant best ranking. For this metric, at a fixed k, the larger the value the better the estimated ranking.

The ReDDE algorithm was compared to the CORI resource selection algorithm in a series of experiments. Our goal is to select just a few databases for search, so the resource ranking algorithms were evaluated only on their ability to rank the top 20 databases. Results were averaged over 50 queries.

The ReDDE algorithm requires that the centralized sample database be searched to produce a ranked-list of documents. The Lemur toolkit implementation of the INQUERY document ranking algorithm [2, 9] was used for this task.

The resource ranking results on the five testbeds are summarized in Figure 1. The ReDDE resource ranking algorithm was as good as or better than the CORI algorithm on all of the five testbeds. There was little difference between the algorithms on the trec123-100col-bysource testbed. The distribution of

database sizes in this testbed is not very skewed, so there is less to be gained by considering it explicitly. On the trec4-kmeans testbed, there was a larger difference between the algorithms due to the somewhat more skewed distribution of database sizes. This testbed is organized by topic, so it is easier for the resource selection algorithms to identify databases that contain a higher percentage of relevant documents. Both algorithms were more accurate on this testbed than on the trec123-100col-bysource testbed, which confirms prior research.

The power of the new algorithm was more apparent on the three testbeds that have many small databases and two large databases. The ReDDE algorithm had a clear advantage over the CORI algorithm at database ranks 1-10 on all three testbeds, i.e., it did not matter whether the density of relevant documents was much higher, the same, or much lower in the large databases. The difference between the algorithms disappeared lower down the database rankings, but in practice it is rare to search so many databases (in this case, more than 10% of all collections).

Analysis of the results indicated that the CORI algorithm had significant difficulty with the large databases; it almost never ranked them in the top 10, which is the main reason for CORI's weaker performance on the trec123-2ldb-60col and trec123-AP-WSJ-60col testbeds. One might worry that the effectiveness of the ReDDE resource selection algorithm on these testbeds is due to a bias towards large databases, but it is also more effective on the trec123-FR-DOE-81col testbed, in which the large databases contain very few relevant documents.

These experiments confirm that explicit consideration of database size and explicit modeling of the distribution of relevant documents across the available databases provides improved resource ranking, particularly in environments where the distribution of database sizes is extremely skewed.

9. EXPERIMENT RESULTS: RETRIEVAL PERFORMANCE

In an operational distributed IR or federated search system, after resource ranking and selection, the query is forwarded to the selected databases, and then a result-merging algorithm merges the ranked lists returned from the different databases into a single, final ranked list. The final set of experiments tested the effect of the ReDDE algorithm on the ranked list of documents that a person would see. This type of evaluation requires a result-merging algorithm. We used the semi-supervised learning method described in [13], which is state-of-the-art, and available in the Lemur toolkit [9]. This algorithm can be, and was, used with both the ReDDE and CORI resource selection algorithms.

For this experiment we selected the trec123-100col-bysource and the trec123-2ldb-60col testbeds. ReDDE and CORI are about equally effective on the trec123-100col-bysource testbed, but ReDDE is much more effective than CORI on the trec123-2ldb-60col testbed, so these testbeds allow opposite extremes to be explored. Experiments were done in which the top-ranked 3 and 5 databases were selected for search, because these conditions emphasize the need for accurate resource ranking.

Each database was searched by the version of the INQUERY document retrieval algorithm [2] implemented by Lemur toolkit [9]. 100 document ids and scores were returned by from each selected database, which the result-merging algorithm compiled into a final ranked list of documents.

The first set of experimental results, on the trec123-100col-bysource testbed, is summarized in Table 5. Although the ReDDE resource ranking algorithm was more effective on this testbed (Section 8), it produces less accurate document rankings. This problem was discussed in Section 5. The modified ReDDE algorithm addresses this situation. The experiment was repeated using the modified ReDDE algorithm on the trec123-100col-bysource and the trec123-2ldb-60col testbeds.

In the second set of experiments, the modified ReDDE algorithm always led to better document rankings than the CORI algorithm (Tables 6 and 7). The modified ReDDE algorithm was notably better than the CORI algorithm on the trec123-2ldb-60col testbed. These results confirm that the modified ReDDE algorithm is more stable and effective than the basic ReDDE algorithm.

10. CONCLUSION

The CORI algorithm has been one of the most consistently effective resource ranking algorithms since it was introduced. This paper identifies a previously unknown weakness that is triggered in environments containing many small databases and a few very large databases, as might occur in corporate or government environments. This flaw motivates the development of the ReDDE algorithm.

The ReDDE algorithm uses explicit estimates of database sizes and a centralized sample database to directly estimate the distribution of relevant documents across the available databases. We believe it is the first resource ranking algorithm to take such an approach. The algorithm was evaluated on five testbeds that explore a wide range of database size, content skew, and relevant document skew configurations. The experimental evidence indicates that the new algorithm is always at least as effective as the CORI algorithm at database ranks 1-10; in some cases it is much more effective. The CORI algorithm may still have an advantage when many databases are searched.

Resource ranking is just one component of a distributed IR or federated search system. People using such a system will be more interested in the accuracy of the final document rankings. Previous research suggested that improved resource selection performance is correlated with improved document rankings. This paper shows that better resource selection algorithm does not always cause better document rankings. An analysis of this phenomenon motivated a modified version of the ReDDE resource selection algorithm. When tested, the document rankings produced with the ReDDE algorithm were always at least as good as document rankings produced with CORI.

The ReDDE algorithm uses constants to model the probability or relevance given a document, and to determine how much of a centralized complete database ranking to model. Experimental results indicate that the algorithm is effective across a range of parameter settings, but these constants are nonetheless ad-hoc and can be considered a weakness of the algorithm. It is likely that training data can be used to automatically determine

Table 5. Precision on the trec123-100col-bysource testbed with 3 and 5 databases were selected.

Docs Rank	3 DBs Selected		5 DBs Selected	
	CORI	ReDDE	CORI	ReDDE
5	0.3760	03760 (0.0%)	0.4360	0.4520 (+3.7%)
10	0.3660	0.3400 (-7.1%)	0.4040	0.4020 (-0.5%)
15	0.3453	0.3253 (-5.8%)	0.3747	0.3893 (+3.9%)
20	0.3140	0.3030 (-3.5%)	0.3650	0.3640 (-0.3%)
30	0.2873	0.2753 (-4.2%)	0.3320	0.3333 (+0.4%)
100	0.1750	0.1740(-0.6%)	0.2206	0.2164(-1.9%)

Table 6. Precision on the trec123-100col-bysource testbed with 3 and 5 databases were selected.

Docs Rank	3 DBs Selected		5 DBs Selected	
	CORI	Modified ReDDE	CORI	Modified ReDDE
5	0.3760	0.4120 (+9.6%)	0.4360	0.4560 (+4.6%)
10	0.3660	0.3720 (+1.6%)	0.4040	0.4180 (+3.5%)
15	0.3453	0.3640 (+5.4%)	0.3747	0.4027 (+7.5%)
20	0.3140	0.3350 (+6.7%)	0.3650	0.3800 (+4.1%)
30	0.2873	0.2930 (+2.0%)	0.3320	0.3393 (+2.2%)
100	0.1750	0.1920 (+9.7%)	0.2206	0.2314 (+4.9%)

Table 7. Precision on the trec123-2ldb-60col testbed with 3 and 5 databases were selected.

Docs Rank	3 DBs Selected		5 DBs Selected	
	CORI	Modified ReDDE	CORI	Modified ReDDE
5	0.3720	0.4480(+20.4%)	0.3960	0.4640 (+17.2%)
10	0.3680	0.4200 (+14.1%)	0.3860	0.4580 (+18.7%)
15	0.3440	0.3853 (+12.0%)	0.3667	0.4387 (+19.6%)
20	0.3240	0.3740 (+15.4%)	0.3520	0.4200 (+19.3%)
30	0.2853	0.3487 (+22.2%)	0.3347	0.4013 (+19.9%)
100	0.1692	0.2476 (+46.3%)	0.2054	0.3006 (+46.3%)

testbed-specific parameter settings, improving both accuracy and generality, but this remains a topic for future research.

This paper also studies the problem of estimating database sizes. A new algorithm called sample-resample is presented that is extremely efficient; in environments containing resource descriptions already created by query-based sampling, the sample-resample method uses several additional queries to provide an estimate of database size. The competing capture-recapture algorithm is also studied relatively carefully. Several modifications are presented that reduce its communications costs dramatically and improve its accuracy. The sample-resample algorithm was the more stable algorithm in our experiments, particularly when database sizes are large.

The work reported here continues a recent research trend studying the uses of query-based sampling in distributed IR (e.g., [1,13]). It also demonstrates another use for a centralized sample database, extending its use as a surrogate for the (unavailable) centralized complete database. Early research on distributed IR treated it primarily as a problem of ranking very large documents. The work reported here and the work that it builds upon continues development of a distinct model of distributed IR, with tools and models that differ significantly from the tools and models used for ranking documents.

ACKNOWLEDGEMENTS

This research was supported by NSF grants EIA-9983253 and IIS-0118767. Any opinions, findings, conclusions, or recommendations expressed in this paper are the authors', and do not necessarily reflect those of the sponsor.

REFERENCES

[1] J. Callan. (2000). Distributed information retrieval. In W.B. Croft, editor, *Advances in Information Retrieval*. Kluwer Academic Publishers. (pp. 127-150).

[2] J. Callan, W.B. Croft, and J. Broglio. (1995). TREC and TIPSTER experiments with INQUERY. *Information Processing and Management*, 31(3). (pp. 327-343).

[3] N. Craswell. (2000). Methods for distributed information retrieval. Ph. D. thesis, The Australian Nation University.

[4] A. Le Calv and J. Savoy. (2000). Database merging strategy based on logistic regression. *Information Processing and Management*, 36(3). (pp. 341-359).

[5] D. D'Souza, J. Thom, and J. Zobel. (2000). A comparison of techniques for selecting text collections. In *Proceedings of the Eleventh Australasian Database Conference (ADC).*

[6] L. Gravano, C. Chang, H. Garcia-Molina, and A. Paepcke. (1997). STARTS: Stanford proposal for internet meta-searching. In *Proceedings of the 20th ACM-SIGMOD International Conference on Management of Data*.

[7] J.C. French, A.L. Powell, J. Callan, C.L. Viles, T. Emmitt, K.J. Prey, and Y. Mou. (1999). Comparing the performance of database selection algorithms. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[8] P. Ipeirotis and L. Gravano. (2002). Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB).*

[9] The lemur toolkit. http://www.cs.cmu.edu/~lemur

[10] InvisibleWeb.com. http://www.invisibleweb.com/

[11] K.L. Liu, C. Yu, W. Meng, A. Santos and C. Zhang. (2001). Discovering the representative of a search engine. In *Proceedings of 10th ACM International Conference on Information and Knowledge Management (CIKM).*

[12] A.L. Powell, J.C. French, J. Callan, M. Connell, and C.L. Viles, (2000). The impact of database selection on distributed searching. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.*

[13] L. Si and J. Callan. (2002). Using sampled data and regression to merge search engine results. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retriev*al.

[14] J. Xu and J. Callan. (1998). Effective retrieval with distributed collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.*

[15] S. Robertson and S. Walker. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retriev*al.