Effective and Efficient Automatic Database Selection

James C. French Allison L. Powell*
Department of Computer Science
University of Virginia
Charlottesville, VA
{french|alp4g}@cs.virginia.edu

Jamie Callan†
Computer Science Department
University of Massachusetts
Amherst, MA
callan@cs.umass.edu

Abstract

We examine a class of database selection algorithms that require only document frequency information. The *CORI* algorithm is an instance of this class of algorithms. In previous work, we showed that *CORI* is more effective than *gGlOSS* when evaluated against a relevance-based standard. In this paper, we introduce a family of other algorithms in this class and examine components of these algorithms and of the *CORI* algorithm to begin identifying the factors responsible for their performance.

We establish that the class of algorithms studied here is more effective and efficient than *gGlOSS* and is applicable to a wider variety of operational environments. In particular, this methodology is completely decoupled from the database indexing technology so is as useful in heterogeneous environments as in homogeneous environments.

*This work supported in part by DARPA contract N66001-97-C-8542 and NASA GSRP NGT5-50062.

†This work supported in part by NSF, the Library of Congress, and the Department of Commerce under agreement EEC-9209623, and by the U.S. Patent and Trademark Office and DARPA/ITO under contract F19628-95-C-0235.

1 Introduction

Database or collection selection [2, 6, 10, 8, 13, 14, 15, 9] is a fundamental problem in distributed searching. Given an environment containing many databases, this aspect of query processing is concerned with selecting the databases that are to be searched to satisfy the query. In this paper, we use the term database to refer to a collection of text documents; we will refer to a group of databases as a collection of databases.

Database selection is the first step in a process that continues with search at the distributed sites and fusing or merging of result lists from the sites. The primary goal in this step is to select as small a set of databases as possible to send a query to without sacrificing retrieval effectiveness. The problem can alternately be defined as determining in what order databases should be visited to provide the most effective response to a query. The proliferation of online resources demands efficient techniques for pruning the search space in this distributed environment and is the impetus for the study of database selection algorithms.

In prior work[4] we conducted the first direct comparison of two well-known database selection techniques described in the literature. That study directly compared two techniques, *gGlOSS*[8, 9] and *CORI*[2], in a common test environment consisting of 236 databases containing 691,058 documents. Both techniques were evaluated using the same data and evaluation techniques. The evaluation sought to quantify the performance difference, if any, between the two algorithms under study. *CORI* was found to outperform *gGlOSS* in the test environment. *CORI* was consistently more effective when measured against a relevance-based ranking (RBR) baseline.

The *gGlOSS* and *CORI* algorithms are based on similar database statistics; however, these statistics are utilized differently by the two algorithms. Our goal

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1999		**00-00-1999 to 00-00-1999**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Effective and Efficient Automatic Database Selection	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Massachusetts Amherst,Department of Computer Science,140 Governors Drive,Amherst,MA,01003**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**12**	

in this study was to understand the characteristics of these algorithms, to identify strengths and weaknesses and to begin identifying the factors responsible for the performance difference noted in French *et al.*[4].

In Section 2 we discuss the database selection problem and the database selection algorithms that we are studying. Section 3 summarizes our previous findings and describes an abstraction of the database selection process. We then present and analyze the results from our current experiments in Section 4. Appendix A covers the testbed used for experiments, performance baselines and evaluation measures. These topics are covered in greater detail in [5, 4]. We suggest that readers unfamiliar with this material read the Appendix after reading Section 2.

2 Previous Database Selection Experiments

This section reviews background material from our earlier work[5, 4] that is crucial to the understanding of the study described in this paper. In the following sections we describe the problem and the algorithms investigated. We conclude with a summary of previous findings to motivate our present study.

2.1 The Problem

In this paper we consider the database selection problem. We are given a query, q, and a set of databases, $DB = \{db_1, db_2, ..., db_N\}$ and are required to rank the databases, that is, decide in what order they should be visited to provide the most effective response to the query. Effectiveness is expressed as a *baseline* ranking and our evaluation determines how well an algorithm estimates the baseline ranking.

Callan *et al.*[2] call this problem the *collection selection* problem while Gravano *et al.*[10] refer to it as the *text database resource discovery* problem. In French *et al.*[5, 4] we refer to this as *database selection* and will retain that terminology here for consistency.

2.2 The Algorithms Investigated

2.2.1 gGlOSS

Gravano *et al.*[10] proposed *GlOSS*, the *Glossary-of-Servers Server*, as an approach to the database selection problem for the Boolean IR model. Later *GlOSS* was generalized to *gGlOSS*[8] to handle the vector space information retrieval model. This generalization can be used for any IR model that computes a score to determine how well a document satisfies a query, provided that certain database statistics can be made available to *gGlOSS*.

gGlOSS assumes that the databases can be characterized according to their *goodness* with respect to any particular query. *gGlOSS*'s job is then to estimate the goodness of each candidate database with respect to a particular query and then suggest a ranking of the databases according to the estimated goodness.

Goodness for each database, *db*, is defined as follows.

$$Goodness(l, q, db) = \sum_{d \in \{db | sim(q,d) > l\}} sim(q, d) \quad (1)$$

where $sim(q, d)$ is a function that calculates the similarity between a query q and a document d. Once $Goodness(l, q, db)$ has been calculated for each database *db* with respect to q at threshold l, the ideal rank for the query at threshold l, $Ideal(l)$ can be formed by sorting the databases in descending order of their goodness.

Note that *gGlOSS* does not compute $Ideal(l)$ rather it is advanced as the goal to which *gGlOSS* estimated ranks $Max(l)$ and $Sum(l)$, defined in [8], will be compared. In French *et al.*[5] we showed that *gGlOSS* $Max(l)$ and $Sum(l)$ estimators do a good job of estimating $Ideal(l)$. We also showed that $Ideal(l)$ does not estimate well the number of relevant documents in a database.

Complete details for calculating the $Max(l)$ and $Sum(l)$ estimators are given in [8] and are not reproduced here. But, for later reference we note that

$$Max(0) = Sum(0) = Ideal(0), \quad (2)$$

that is, at threshold $l = 0$ both estimators give identically the $Ideal(0)$ ranking of databases for all queries. In addition, $l = 0$ allows a consistent comparison of $Ideal(l)$ rankings when comparing different underlying retrieval systems that produce differently scaled similarity values (i.e. $sim(q, d)$ in Equation 1). Hence, in the evaluation to follow we will use $Ideal(0)$ as the *gGlOSS* estimate since *gGlOSS* can compute this exactly.

Note that *gGlOSS* needs two vectors of information from each database db_i in order to make its estimates. They are combined into two matrices F and W.

1. $F = [df_{ij}]$ where df_{ij} is the document frequency (the number of documents in the database containing the term) for each term t_j in db_i; and
2. $W = [w_{ij}]$ where w_{ij} is the sum of the weights of each term t_j over all documents in db_i, that is, the column sums of the document-term matrix.

If the underlying database cannot be made to divulge this information directly, it is in principle possible to

recover the information by issuing a single-term query for each vocabulary term. Our choice of $Ideal(0)$ obviates this; we can compute $Ideal(0)$ directly from the databases by simply issuing the test queries.

2.2.2 *CORI*

Given a set of databases to search, the *CORI* approach creates a *database selection index* in which each database is represented by its terms and their document frequencies *df*. Databases are ranked for a query q by a variant of the Inquery document ranking algorithm. The belief $p(r_k|db_i)$ in database db_i due to observing query term r_k is determined by:

$$\begin{aligned} T &= \frac{df}{df + 50 + 150 \cdot cw/\overline{cw}} \\ I &= \frac{\log\left(\frac{|DB|+0.5}{cf}\right)}{\log\left(|DB| + 1.0\right)} \\ p(r_k|db_i) &= 0.4 + 0.6 \cdot T \cdot I \end{aligned} \quad (3)$$

where:

df	is the number of documents in db_i containing r_k,
cf	is the number of databases containing r_k,
$\|DB\|$	is the number of databases being ranked,
cw	is the number of words in db_i, and
$\overline{cw}$	is the mean cw of the databases being ranked.

The belief in a database depends upon the query structure, but is usually just the average of the $p(r_k|db_i)$ values for each query term [2].

The *CORI* approach to ranking databases can be summarized as $df \cdot icf$, where icf is inverse collection frequency. [1] *CORI* utilizes information about the number of terms in a database plus information from (or derivable from) the matrix F used by *gGlOSS*.

2.3 Summary of Previous Findings

To motivate the present paper it will be useful to summarize our findings from earlier work. In French *et al.*[5] we showed that:

1. *gGlOSS* estimates $Ideal(0)$ quite well; but
2. *gGlOSS* does not perform well against the relevance-based ranking (RBR).

We found that *gGlOSS* tended to choose the largest database for each query. We believe that is due to summing similarities to get $Goodness(\cdot)$ (see Equation 1) since larger databases will contribute more often to this sum. Subsequently in French *et al.*[4] we found that:

1. *CORI* is more effective than *gGlOSS* for database selection; and
2. *gGlOSS* is highly correlated to the size-base ranking (SBR) baseline while *CORI* is not.

This latter finding is consistent with the *gGlOSS* tendency to select large databases.

3 Investigating Database Selection Algorithms

Because *gGlOSS* and *CORI* both use similar database statistics as the basis of their ranking algorithms, we saw an opportunity to learn more about the behavior of database selection algorithms by means of a careful study of some of the components of the *CORI* algorithm. In particular, we hoped to discover why *CORI* did not exhibit the tendency to select large databases.

The object of our study was to discover what factors in the *CORI* algorithm were most important in terms of its performance. There were two obvious differences in the *CORI* and *gGlOSS* approaches.

1. *CORI* represents databases as virtual documents with suitable term weights and employs a document processing strategy on this representation.
2. *CORI* is based on the Inquery search engine and Inquery employs document length normalization. Since the "documents" indexed by *CORI* are representations of databases, it might be the case that document length normalization had the effect of compensating for the size of a database.

In part the *CORI* advantage over *gGlOSS* is due to the fact that it is not highly correlated to SBR. There is clearly some size compensation going on and perhaps it is due to length normalization. We undertook a systematic study to examine these and other effects.

One set of experiments examined *CORI* directly, individually disabling portions of the *CORI* computation to gauge the impact on performance. Additional experiments investigated the *CORI* virtual document representation. For these experiments, we abstracted the database selection process in a way that let us have reasonably fine control over a family of weighting functions so that we could observe the effects of changes in a controlled environment.

3.1 Abstracting the Problem

Database sites can be represented as virtual documents. That is, each database is represented by its terms and their document frequencies. In fact this is

[1] The term icf is used widely in the IR literature on collection selection and we retain it here for consistency.

nothing more than the matrix $F = [df_{ij}]$ that *gGlOSS* requires for its operation. The difference is simply the interpretation. For this interpretation, F is $F = [df_{st}]$ where df_{st} is the number of documents at site s that contain term t. Each row of F is a virtual document in the *CORI* terminology. F is an $S \times V$ matrix with S rows, one for each of the S sites indexed and V columns where V is the cardinality of the union vocabulary, the set of all unique terms in all the databases. It will be the case that $df_{st} = 0$ when term t does not occur in any document at site s. Depending on the heterogeneity of the databases, F can be a very sparse matrix.

CORI treats the virtual documents as a database and performs a similarity calculation of a query q against these "documents." The subsequent document ranking is the desired database ranking.

We abstracted the *CORI* $df \cdot icf$ approach to database selection by creating a set of virtual documents using the *df* information available from the F matrix. We indexed the virtual document database with SMART version 11.0[1] and examined the behavior of a variety of so-called $tf \cdot idf$ weighting functions in an attempt to isolate gross effects.

Before continuing we should clarify the terminology that we will subsequently use. Under the vector space model of information retrieval, a similarity calculation is made between the query and each document in the database. The documents in the database are subsequently ranked by these similarities. Document and query vectors are formed by assigning weights to each term (word) in the document (query) and then subsequently taking the inner product of the vector representations. When the vector representations are suitably normalized, the inner product calculates the cosine of the angle formed by the vectors. A variety of strategies is used to assign term weights based on statistics computed over the individual documents and the database as a whole. Two very common components in these weight functions are:

1. *term frequency* (tf_{ij}): the number of occurrences of term j in document i; and

2. *document frequency* (df_j): the number of documents in the database containing term j.

In the $tf \cdot idf$ indexing strategy, term weights (w_{ij}) are formed by taking the product of the term frequency and the inverse document frequency and have the following form:

$$w_{ij} = tf_{ij} \cdot log\left(\frac{N}{df_j}\right). \tag{4}$$

The intuition behind this weighting strategy is that a term j that occurs many times in a document i but relatively infrequently in other documents is a good discriminator and should receive a high weight. There are many variants on this strategy and SMART provides a facility for specifying the precise weighting function by means of parameters.

Since similarity calculations take the form of inner products, the $tf \cdot idf$ strategy results in the sum of terms that look like:

$$(tf_d \cdot idf_d)\,(tf_q \cdot idf_q) = tf_d \cdot tf_q \cdot idf_d^2 \tag{5}$$

where the subscripts d and q denote a document and query respectively. The squared term results from the fact the idf_q is estimated by idf_d. Our investigation centered on the simplest weighting strategies and considered weights of the form $tf^i \cdot idf^j$ so that terms in the inner product have the form

$$tf_d^l \cdot tf_q^m \cdot idf^n \tag{6}$$

Note that often we would like to ignore the term frequency of the query terms and just set $tf_q = 1$. This has some advantages and we investigated this option too.

There is a subtle difference between document retrieval and database selection. Note that in the vector representation for the databases we are using *df* as the tf component and inverse collection frequency (icf) as the idf component. This means that the weighting strategies are of the form $df \cdot icf$, and this is actually the form we investigate. The intuition is that a site having many documents containing a term t that is reasonably rare across all the databases (i.e., occurs in very few other databases) should be ranked highly; the term t is a good discriminator. Thus, we examined weighting strategies of the form

$$df_s^l \cdot tf_q^m \cdot icf_s^n \tag{7}$$

where s denotes the database site, that is, the vector representing site s.

We note that CORI does not have this precise form. In particular, the *df* component of the CORI ranking function is $df/(df + 50 + 150 \cdot cw/\overline{cw})$, and CORI does not have a squared icf term. However, CORI *is* a form of $df \cdot icf$ algorithm, and Equation 7 approximates it.

The abstraction defined by Equation 7 is convenient because its components are easy to vary in a systematic manner. We used it in the work reported below to study the effects of *df* and icf on database selection algorithms.

3.2 Summary of Approach

Given a set of S databases we can provide efficient and effective access by means of the following approach.

1. Each site constructs its indexing vocabulary and computes its local document frequencies for each term in that vocabulary.

2. Each site sends this information to a central database selection server.

3. The server creates a union vocabulary and constructs the matrix $F = [df_{st}]$. A B-tree or hashed implementation of an inverted file would be the most appropriate representation for F.

4. The server only needs to count the number of nonzero entries in each column of F to compute the collection frequency (cf) and subsequently the inverse collection frequency (icf) for each term.

5. Compute icf^k for the particular choice of k. We present results for different values of k in Section 4.

Given a query q, the conceptual database selection algorithm proceeds as follows. We compute a rank vector R of size S.

1. Extract the terms from the query and compute tf_t for each term t.

2. Initialize each entry in R to 0.

3. For each term t, for $s = 1, ..., S$,

$$R_s = R_s + tf_t \cdot F_{st}. \quad (8)$$

   This is a simple sum when query tf is being ignored.

4. Sort the sites according to the value in R.

This conceptual model of our approach also includes *CORI*.

4 Results

In this section we report performance results for database selection strategies that use only the information in F or that can be derived from F (df values are stored in F and icf values can be derived). As we will show, excellent performance can be obtained on the basis of this information alone.

4.1 *CORI* Length Normalization

We first considered the possibility that virtual document length normalization in *CORI* had the effect of compensating for the size of the databases. To test this possibility, we disabled *CORI*'s length normalization by replacing the $cw/\overline{cw}$ component of the *CORI* computation with the constant 1 (see Equation 3). This resulted in a negligible effect on performance. From this we conclude that normalization based on database vocabulary size in *CORI* is not responsible for its effectiveness.

4.2 *CORI* Default Belief Value

We also considered the effect of the default belief value. We disabled the default belief of 0.4 (see Equation 3) and set the default to 0.0. We evaluated these results and found no effect on performance. From this we concluded that the default belief value used in the *CORI* weighting scheme is also not responsible for *CORI*'s effectiveness.

4.3 Indexing Parameters

Our remaining experiments investigate the abstracted virtual document representation defined in Section 3.1. These experiments were designed to examine the efficacy of this general approach for database selection. We examined weighting strategies of the form $df_s^l \cdot tf_q^m \cdot icf_s^n$, varying the relative influence of the components. Since df is always associated with a site s and tf is only associated with a query, in the remainder of the paper we will drop the subscripts and use the abbreviated notation $df^l \cdot tf^m \cdot icf^n$ when labelling results.

We began our experiments by both selecting parameters shown to perform well for document retrieval and selecting parameters in an attempt to mimic the *CORI* approach. Because we did not use *CORI* directly, we were unable to reproduce the *CORI* approach exactly.

We examined the initial results and found that the simple approach of $df \cdot tf \cdot icf^2$ showed the most promise. $df \cdot tf \cdot icf^2$ performed better than the other initial parameter settings and better than $Ideal(0)$ but not as well as *CORI*. We used the $df \cdot tf \cdot icf^2$ parameter settings as a starting point for the experiments reported here.

From this point, we first explored the impact of vector length normalization on performance. We then examined a series of parameter settings to determine the impact of varying the influence of the df and icf components. Our final step was to determine if we

could simplify the computation without sacrificing performance.

In many cases, any change in performance is small or might be obscured by other plots in a graph. We include both graphs for visual inspection and tables of points over an operational range of interest.

4.4 Impact of Vector Length Normalization

In earlier experiments, we determined that the virtual document length normalization in *CORI* had negligible impact on effectiveness. In these experiments we wanted to determine if the different approach of vector length normalization had any impact on performance. We began with un-normalized vectors of the form $(df \cdot icf)(tf \cdot icf)$ (labelled **none**) then normalized

- only the query vectors (**queries**);
- only the database vectors (**virtual docs**); and finally
- both the database and query vectors (**both**).

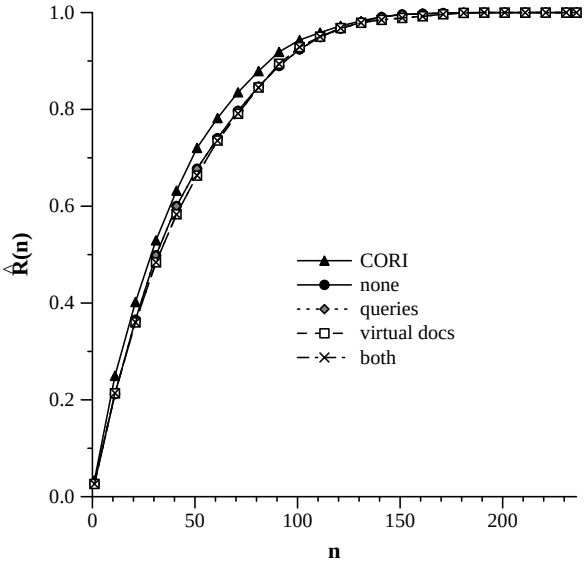

Figure 1: Impact of adding normalization.

Table 1 and Figure 1 show the results for these variations. Normalizing the query vectors had virtually no effect; normalizing the database vectors had a small adverse effect. However, the overall effect was negligible. Figure 1 also shows the performance of *CORI* for comparison.

In addition to the vector-length normalization experiments, we also investigated approaches that normalized only the *df* component in our $df \cdot tf \cdot icf$ representation, for example, using the log of the *df* component or dividing by the maximum *df* value. However, the alternate *df* normalization approaches that we tried degraded performance.

n	**none**	**queries**	**virtual docs**	**both**
1	0.027	0.027	0.026	0.026
11	0.212	0.212	0.213	0.214
21	0.365	0.365	0.360	0.360
31	0.499	0.499	0.484	0.484
41	0.600	0.600	0.583	0.583
51	0.677	0.677	0.663	0.664
61	0.740	0.740	0.735	0.735
71	0.797	0.797	0.791	0.791
81	0.847	0.847	0.845	0.845
91	0.890	0.890	0.894	0.894
101	0.924	0.924	0.929	0.929
111	0.949	0.949	0.950	0.950
121	0.967	0.967	0.968	0.968

Table 1: Impact of adding normalization, evaluation measure $\widehat{\mathcal{R}}_n$

4.5 Varying *df* and *icf* Components

Our next set of experiments examined the impact of varying the degree of use of the *df*, *icf* and query *tf* components. We began with $df \cdot tf$, added an *icf* component, then steadily increased the influence of the *icf* component. The results are plotted in Figure 2 and detailed in Table 2. In Table 2, the results of increasing the *icf* component are shown to the left of the double line. Note that overall the performance improved as the *icf* component was increased with $df \cdot tf \cdot icf^4$ exhibiting the best performance for the icf^k variants. Next we varied the influence of the *df* and query *tf* components using $df \cdot tf \cdot icf^4$ as a starting point. The results for modifying the *df* and query *tf* components are shown to the right of the double line in Table 2. Increasing the influence of the query *tf* component improved performance. However, increasing the influence of the *df* component degraded performance to better than $df \cdot tf$ but worse than $df \cdot tf \cdot icf$.

Note that $df \cdot tf^2 \cdot icf^4$ yields the best performance of the parameters examined so far.

4.6 Overview of Approaches

At this point, it is illuminating to compare the results from all of the general approaches that we have considered in this and in previous work. These approaches include *CORI* (described in Section 2.2.2), *gGlOSS* (described in Section 2.2.1), our simple Size-Based Rank-

		$df \cdot tf \cdot *$					$* \cdot icf^4$		
n	***CORI***	icf^0	icf^1	icf^2	icf^3	icf^4	$df \cdot tf^2$	$df^2 \cdot tf$	$df^2 \cdot tf^2$
1	0.34	0.024	0.026	0.027	0.029	0.029	0.029	0.021	0.021
11	0.250	0.188	0.204	0.212	0.221	0.223	0.228	0.195	0.202
21	0.402	0.338	0.361	0.365	0.373	0.380	0.384	0.349	0.354
31	0.529	0.475	0.496	0.499	0.502	0.508	0.520	0.486	0.493
41	0.632	0.573	0.598	0.600	0.606	0.613	0.623	0.590	0.595
51	0.720	0.639	0.667	0.677	0.686	0.690	0.703	0.665	0.671
61	0.728	0.684	0.721	0.740	0.755	0.757	0.768	0.722	0.733
71	0.835	0.729	0.769	0.797	0.811	0.812	0.827	0.781	0.790
81	0.879	0.781	0.819	0.847	0.857	0.859	0.871	0.831	0.836
91	0.919	0.831	0.870	0.890	0.903	0.902	0.909	0.881	0.881
101	0.943	0.881	0.906	0.924	0.932	0.933	0.940	0.918	0.916
111	0.959	0.918	0.938	0.949	0.955	0.956	0.960	0.946	0.945
121	0.973	0.953	0.962	0.967	0.970	0.972	0.974	0.966	0.966

Table 2: Impact of increasing influence of icf component, then df component, evaluation measure $\widehat{\mathcal{R}}_n$

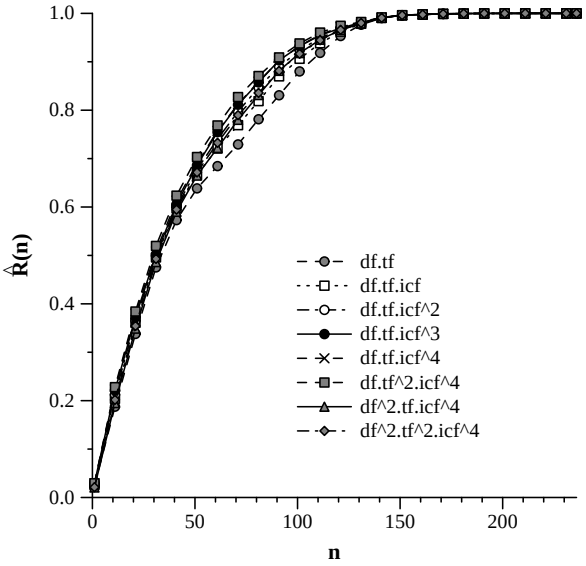

Figure 2: Impact of increasing influence of icf component, then df component.

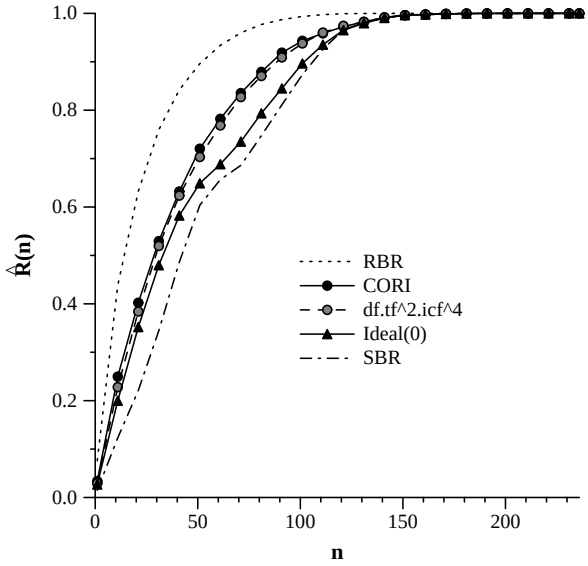

Figure 3: Comparison of approaches.

ing (SBR) approximator (described in Section A.3) and the best of the parameter settings that we have just examined, $df \cdot tf^2 \cdot icf^4$.

Figures 3, 4 and 5 and Table 3 show the comparison of these approaches. Also included in the figures is the data for RBR, the best possible performance. We use measures $\mathcal{R}_n$, $\widehat{\mathcal{R}}_n$ and $\mathcal{P}_n$ here so that the overall performance can be compared in more ways.

Examining Figure 3, there are a number of immediate observations. First, the performance of $CORI$ and $df \cdot tf^2 \cdot icf^4$ are very close with $CORI$ still having a slight advantage. Both $CORI$ and $df \cdot tf^2 \cdot icf^4$ outperform $Ideal(0)$. When comparing the performance of these three approaches, we note that there are three regions of interest in Figure 3. These regions were first identified in [4] and can also be seen in Figures 4 and 5. For all measures, for n roughly less than 50, the performance of $CORI$, $Ideal(0)$ and $df \cdot tf^2 \cdot icf^4$ differs only slightly. For n greater than roughly 120, all algorithms are faced with a situation in which most of the relevant documents have been located and little improvement is possible. However for n in the range of 50-120, $CORI$ and $df \cdot tf^2 \cdot icf^4$ have a more evident advantage. We have noted that this is partly due to the correlation of $Ideal(0)$ and the SBR approximator.

n	**RBR**	***CORI***	$df \cdot tf^2 \cdot icf^4$	$Ideal(0)$	**SBR**
1	0.076	0.034	0.029	0.027	0.015
11	0.438	0.250	0.228	0.200	0.123
21	0.632	0.402	0.384	0.352	0.221
31	0.759	0.529	0.520	0.480	0.344
41	0.842	0.632	0.623	0.582	0.486
51	0.896	0.720	0.703	0.649	0.604
61	0.934	0.782	0.768	0.688	0.657
71	0.960	0.835	0.827	0.735	0.686
81	0.977	0.879	0.870	0.794	0.747
91	0.987	0.919	0.909	0.845	0.811
101	0.994	0.943	0.940	0.896	0.873
111	0.997	0.959	0.960	0.935	0.926
121	0.999	0.973	0.974	0.965	0.967

Table 3: Comparison of approaches, evaluation measure $\widehat{\mathcal{R}}_n$

In the course of this investigation, we studied many other parameter settings that were not included in this paper. With a few exceptions, slight changes in parameters produced little or no improvement in overall performance.

4.7 Similarity of Rankings

In our abstraction experiments we have used the same raw information (df and icf) that *CORI* uses, but the way that we use that information to compute database weights is only loosely related. Still, we managed to achieve very similar performance. An obvious question is whether only the performance was similar or if the database rankings that we produced were also similar.

We computed the mean squared error for *CORI* and $df \cdot tf^2 \cdot icf^4$ rankings. The MSE values for queries 51-150 are plotted in Figure 6. The mean MSE over all queries was 613.9, the median MSE was 447.4. The two approaches produce similar performance results, implying that on average they locate databases with relevant documents equally well. However, they are not ranking the databases in exactly the same order. It is interesting to note that for no query did the two approaches produce exactly the same ranking. We calculated the Spearman rank correlation coefficient (see Appendix A.4.3) between the ranks produced by both approaches for each query in the testbed and found that the average correlation was 0.933. The two approaches are both effective and highly correlated, but not equivalent.

4.8 Simplifying Computation

A final consideration is the cost of computing the functions with which we rank databases. The icf component is not dependent upon query information and therefore can be incorporated completely in the indexing stage. Only the query tf component must be considered at query time. We have shown that increasing the influence of the query tf component improved performance; however, if we can remove that component from Equation 8 and simply keep track of the query terms, we can reduce the computation to a simple sum.

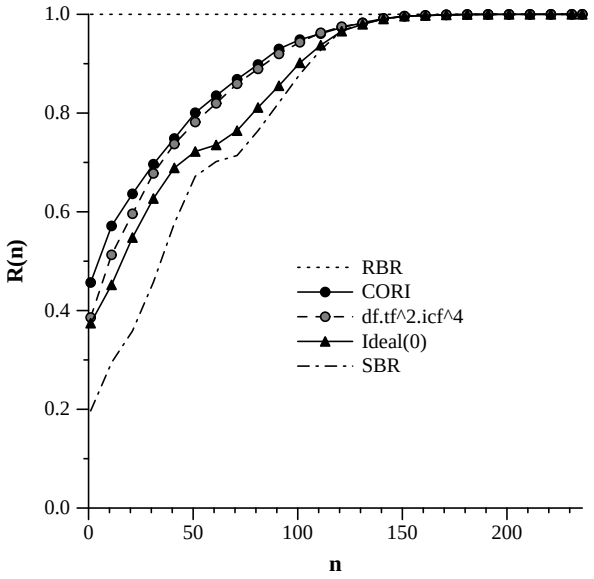

Figure 4: Comparison of approaches.

n	***CORI***	df	$df \cdot icf^2$	$df \cdot icf^4$	$Ideal(0)$
1	0.034	0.022	0.023	0.029	0.027
11	0.250	0.181	0.203	0.220	0.200
21	0.402	0.334	0.361	0.368	0.352
31	0.529	0.473	0.491	0.497	0.480
41	0.632	0.567	0.595	0.604	0.582
51	0.720	0.632	0.674	0.685	0.649
61	0.782	0.676	0.734	0.754	0.688
71	0.835	0.720	0.794	0.811	0.735
81	0.879	0.764	0.846	0.856	0.794
91	0.919	0.814	0.892	0.900	0.845
101	0.943	0.864	0.925	0.933	0.896
111	0.959	0.910	0.948	0.955	0.935
121	0.973	0.946	0.967	0.970	0.965

Table 4: Simplifying the computation, evaluation measure $\widehat{\mathcal{R}}_n$

Figure 7 and Table 4 show the performance of several simplified computations. $df \cdot icf^4$ is the simplified version of the best-performing $df \cdot tf^2 \cdot icf^4$. Simplifying the computation adversely affects the performance but $df \cdot icf^4$ is still a reasonable approximator in that

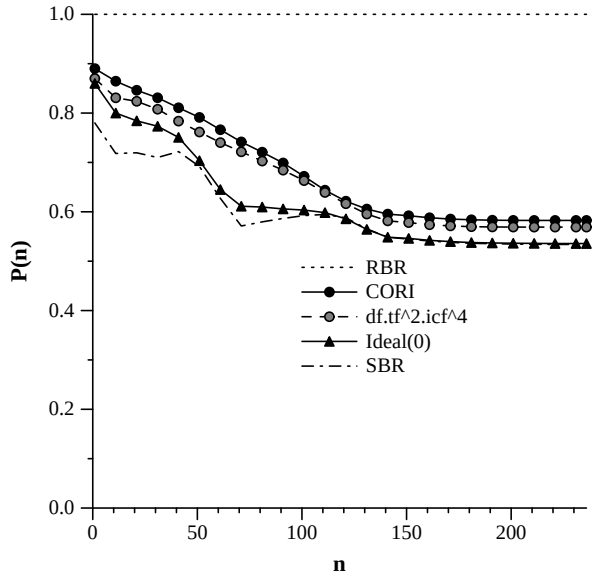

Figure 5: Comparison of approaches.

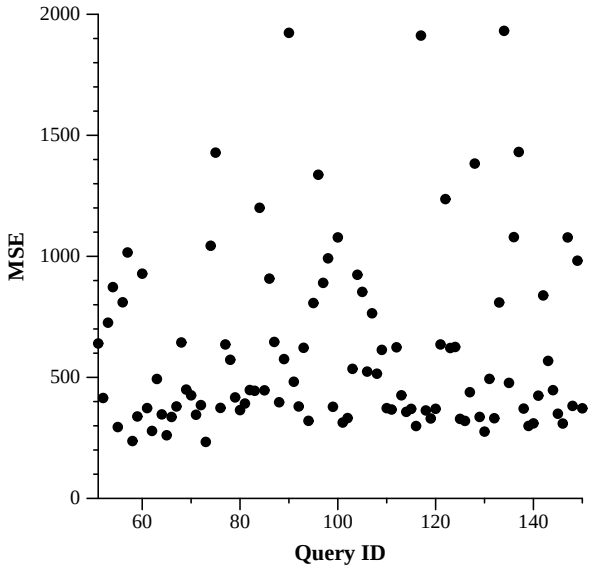

Figure 6: Mean squared error for $CORI$ and $df \cdot tf^2 \cdot icf^4$ rankings.

it performs better than $df \cdot tf \cdot icf^2$.

Simply summing the df values for each term in a query (df) is a very simple approach. This approach performs similarly to $Ideal(0)$ and also exhibits the same correlation with SBR. However, this approach performs slightly worse than $gGlOSS$. It outperforms SBR for $n < 95$.

It's interesting to note here that adding the icf component serves to lessen the visible correlation with SBR.

5 Summary of Requirements and Performance

5.1 Statistics Required

$gGlOSS$ uses two matrices, F and W in its selection algorithm; the $df \cdot icf$-based approaches we have studied here require only the information contained in the matrix F. $CORI$ also uses the vocabulary size of a database (cw) but we have shown here that this has little impact on the performance.

There is a very important point to be made about limiting the amount of information used by a database selection algorithm to the matrix F. The values of df that make up the entries in F are not affected by the indexing strategy used by the database; they are pure statistics from the text. In contrast, the matrix W used by $gGlOSS$ for its estimation is completely dependent on the underlying indexing strategy. Hence, algorithms based on F alone will not have to gather new statistics even if the underlying database is reindexed with some different set of parameters, or a different technology for that matter. $gGlOSS$, on the other hand, will have to recompute W. A related point is that when using F alone, there is no requirement that the underlying databases use the same or even similar search engines or IR models; database selection is completely decoupled. These are advantages that make these algorithms attractive and worthy of further study.

5.2 Performance and Generality

In previous work [4] we showed that $CORI$ is more effective than $gGlOSS$ when compared to a relevance-based baseline. Here, we have shown that the class of $tf \cdot idf$-based algorithms can achieve similar performance.

We also showed [4] that $CORI$ achieves similar performance over collections

- containing different numbers of databases (100, 236 and 921 database collections), and
- where the databases contained similar numbers of documents as well as collections where the databases were of different sizes.

We do not yet know the performance of $gGlOSS$ in all of these environments. However, we do know that $gGlOSS$ tends to select databases with a large number of documents (as shown by the high correlation with SBR [4]). We also know that due to its use of the W

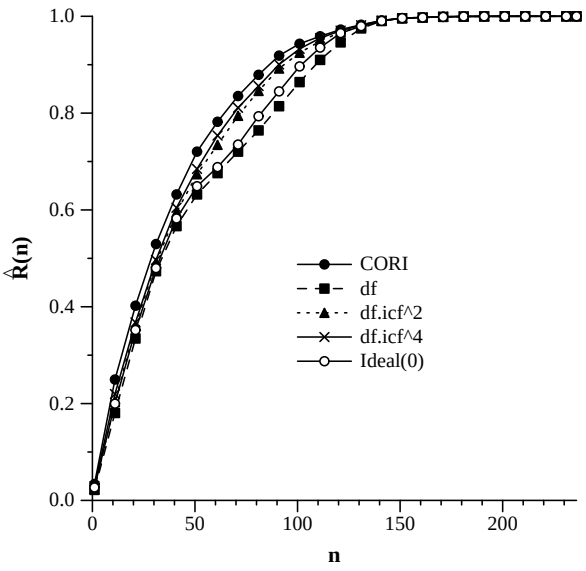

Figure 7: Simplifying the computation.

matrix, *gGlOSS* is vulnerable to indexing choices in the underlying databases.

6 Conclusions

We have demonstrated that efficient and effective database selection algorithms can be developed that only require information about document frequencies (df) at each participating database. These df values are not affected by the indexing strategies used by the databases, as a result, $df \cdot icf$-based strategies are applicable in both heterogeneous and homogeneous environments.

We have shown that the $df \cdot icf$-based approaches studied here (including *CORI*) require less computation. We have studied the components of these algorithms to determine which components contribute to the effectiveness of the algorithms. These approaches can be reduced to a table lookup summing a single value for each term in the query. Thus, they are very efficient. The data structure is relatively small and easy to update. It can also be widely replicated.

We have shown that one of the algorithms in this class (*CORI*) works well for both databases of similar sizes and databases of different sizes. However, we have been unable to determine precisely why *CORI* database rankings are less influenced by database size than *gGlOSS* rankings. We do not yet know how *gGlOSS* will perform in an environment where databases are of similar sizes.

In these and previous experiments, we have shown that it is possible to achieve better performance than *gGlOSS* while using less information than it currently uses. *gGlOSS* uses two matrices F and W in its selection algorithm; the $df \cdot icf$ approaches only require F. It might be possible to achieve better performance with the addition of the information contained in W, but so far that has not been shown. We should also note that, as we said at the outset, we used the same indexing parameters for *gGlOSS* that Gravano *et al.*[8] used. It is possible that the performance of *gGlOSS* could be improved with another choice of parameters.

It is interesting that the additional $df \cdot icf$ approaches studied here have been able to achieve performance equivalent to *CORI* but no better. It may be that these algorithms are doing the best that can be done within our testbed. We need to look at this further to see if we can determine the limiting behavior.

For now it can be said that efficient and effective database selection algorithms are possible using only document frequency information. This is an important class of algorithms because the algorithms are independent of the underlying indexing technology. It remains to be seen whether significant further improvements can be made by the application of additional information.

Acknowledgements

We thank Travis Emmitt, Kevin Prey and Yun Mou for their help in processing the data that was used in the experiments reported here. We also thank Charlie Viles for helpful discussions during the preparation of this paper.

References

[1] C. Buckley. SMART version 11.0, 1992. ftp://ftp.cs.cornell.edu/pub/smart.

[2] J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks. In *Proceedings of the 18th International Conference on Research and Development in Information Retrieval*, pages 21–29, 1995.

[3] J. C. French. Metrics for Evaluating Database Selection Techniques. Technical report, Department of Computer Science, University of Virginia, 1999. In preparation.

[4] J. C. French, A. L. Powell, J. Callan, C. L. Viles, T. Emmitt, K. J. Prey, and Y. Mou. Comparing the Performance of Database Selection Algorithms. Technical Report CS-99-03, Department of Computer Science, University of Virginia, January 1999.

[5] J. C. French, A. L. Powell, C. L. Viles, T. Emmitt, and K. J. Prey. Evaluating Database Selection Techniques: A Testbed and Experiment. In *Proceedings*

of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 121–129, August 1998.

[6] N. Fuhr. A Decision-Theoretic Approach to Database Selection in Networked IR. *ACM Transactions on Information Systems*. To appear.

[7] J. D. Gibbons. *Nonparametric Methods for Quantative Analysis*. Holt, Rinehart and Winston, 1976.

[8] L. Gravano and H. Garcia-Molina. Generalizing GlOSS to Vector-Space Databases and Broker Hierarchies. In *Proceedings of the 21st International Conference on Very Large Databases (VLDB)*, 1995.

[9] L. Gravano, H. Garcia-Molina, and A. Tomasic. GlOSS: Text-source discovery over the internet. *ACM Transactions on Database Systems*, To appear.

[10] L. Gravano, H. Garcia-Molina, and A. Tomasic. The Effectiveness of GlOSS for the Text Database Discovery Problem. In *SIGMOD94*, pages 126–137, 1994.

[11] D. Harman. Overview of the Fourth Text Retrieval Conference (TREC-4). In *Proceedings of the Fourth Text Retrieval Conference (TREC-4)*, 1996.

[12] Z. Lu, J. P. Callan, and W. B. Croft. Measures in collection ranking evaluation. Technical Report TR-96-39, Computer Science Department, University of Massachusetts, 1996.

[13] A. Moffat and J. Zobel. Information Retrieval Systems for Large Document Collections. In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, pages 85–94, 1995.

[14] J. Xu and J. Callan. Effective Retrieval with Distributed Collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–120, August 1998.

[15] B. Yuwono and D. L. Lee. Server Ranking for Distributed Text Retrieval Systems on Internet. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications*, pages 41–49, April 1997.

Appendix

A Experimental Methodology

This appendix covers the testbed used for our experiments, the treatment of the testbed for different facets of the experiments, performance baselines and evaluation measures. These topics are covered in more detail in [5, 4].

A.1 The Testbed

In French *et al.*[5] we proposed a test environment for the systematic study of distributed information retrieval algorithms. Our testbed is based on the TIPSTER data used in the TREC[11] conferences. We decompose the large TREC collections into smaller databases that serve as hypothetical "sites" in our distributed information retrieval test environment. The data is decomposed by source, year, and month resulting in 236 sites. We used TREC topics 51-150 as the test queries in our earlier studies[5, 4]. The characteristics of this testbed, the queries used, and other details can be found in French *et al.*[5].

A.2 The Experimental Environment

All our tests were conducted on our testbed using the full 236 databases of TREC data described earlier. We used the TREC topics 51-150 as the test query set.

We prepared the test collection as in French *et al.*[5] by using SMART version 11.0[1] using the same parameters as Gravano *et al.*[8]. Note that for those experiments each of the 236 sites used the same parameters and search engine (SMART) to process queries.

We took steps to guarantee that the same indexing vocabulary was used by all algorithms under investigation. The specific details of how we controlled the indexing vocabulary are given in French *et al.*[4]. We note that we have determined that the steps taken to control the vocabulary for *CORI* did not have any significant effect on its performance. This was established by comparing and unconstrained version of *CORI* running its own stop list and stemming algorithm against our controlled vocabulary.

A.3 Evaluation—Baselines for Comparison

We refer to a number of baselines in the evaluation below, specifically: the *gGlOSS* baseline, *Ideal*(0); the relevance-based ranking (RBR); and the size-based ranking (SBR). They are defined as follows.

Ideal(0): This ranking is produced by processing each query for each of the 236 databases and then using the goodness (see Equation 1) to rank the databases.

RBR: These rankings were produced for each query by using the relevance judgements supplied with the TREC data. In the RBR baseline databases are simply ordered by the number of relevant documents they contain.

SBR: Databases are ordered by the total number of documents they contain. Note that this ranking is constant for all queries.

A.4 Evaluation—Metrics for Comparison

As in our earlier work we use mean squared error, the Spearman coefficient of rank correlation[7], two recall metrics, $\mathcal{R}_n$ [8] and $\widehat{\mathcal{R}}_n$ [5], and a precision measure, $\mathcal{P}_n$ [8]. These are discussed below.

A.4.1 Mean Squared Error

Given a group of N databases to rank, for any candidate ranking we compute

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(base\_rank(db_i) - est\_rank(db_i)\right)^2 \tag{9}$$

where $base\_rank(db_i)$ is the baseline or desired rank and $est\_rank(db_i)$ is the predicted rank for db_i.

A.4.2 Recall and Precision Analogs

In [5] we discussed performance metrics that are analogous to the well known IR metrics of *recall* and *precision*. We briefly review the metrics here.

We provide a *baseline ranking*, B, that represents a desired goal for each query. An algorithm produces some *estimated ranking* for the query, E, and our goal is to decide how well E approximates B. We assume that each database db_i in the collection has some merit, $merit(q, db_i)$, to the query q. The baseline is expressed in terms of this merit; the estimate is formed by implicitly or explicitly estimating merit.

Let db_{b_i} and db_{e_i} denote the database in the i-th ranked position of rankings B and E respectively. Let

$$B_i = merit\left(q, db_{b_i}\right) \text{ and } E_i = merit\left(q, db_{e_i}\right) \tag{10}$$

denote the merit associated with the i-th ranked database in the baseline and estimated rankings respectively. In the results that follow we have the following convention. For the $Ideal(l)$ calculations we have $merit(q, db) = Goodness(l, q, db)$; for RBR we define $merit(q, db)$ to be the number of relevant documents in db; and for SBR we define $merit(q, db)$ to be the total number of documents in db.

Gravano *et al.*[8] defined $\mathcal{R}_n$ as follows.

$$\mathcal{R}_n = \frac{\sum_{i=1}^{n} E_i}{\sum_{i=1}^{n} B_i}. \tag{11}$$

This is a measure of how much of the available merit in the top n ranked databases of the baseline has been accumulated via the top n databases in the estimated ranking.

An alternative definition[5] is given by

$$\widehat{\mathcal{R}}_n = \frac{\sum_{i=1}^{n} E_i}{\sum_{i=1}^{n^*} B_i} \tag{12}$$

where

$$n^* = \max k \text{ such that } B_k \neq 0. \tag{13}$$

Intuitively, n^* is the breakpoint between the useful and useless databases. The denominator is just the total merit contributed by all the databases that are useful to the query. Thus, $\widehat{\mathcal{R}}_n$ is a measure of how much of the total merit has been accumulated via the top n databases in the estimated ranking. Lu *et al.*[12] have also suggested using this measure.

Let $M(E, B)$ denote an evaluation measure M for comparing an estimator E against a baseline B. We have shown elsewhere[3] that

$$\widehat{\mathcal{R}}_n(E, B) = \mathcal{R}_n(E, B) \cdot \widehat{\mathcal{R}}_n(B, B)$$

so that

$$\mathcal{R}_n(E, B) = \frac{\widehat{\mathcal{R}}_n(E, B)}{\widehat{\mathcal{R}}_n(B, B)}.$$

That is, $\mathcal{R}_n$ can be interpreted as the fraction of the available baseline merit that has been accumulated in the top n ranks.

Gravano *et al.*[8] have also proposed a precision-related measure, $\mathcal{P}_n$. It is defined as follows.

$$\mathcal{P}_n = \frac{|\{db \in Top_n(E) | merit(q, db) > 0\}|}{|Top_n(E)|} \tag{14}$$

This gives the fraction of the top n databases in the estimated ranking that have non-zero merit.

A.4.3 Spearman Coefficient of Rank Correlation

The Spearman coefficient of rank correlation, ρ, is given by

$$\rho = 1 - \frac{6\sum_{i=1}^{n} D_i^2}{n(n^2 - 1)} \tag{15}$$

where D_i is the difference in the i-th paired ranks. We have $-1 \leq \rho \leq 1$ where $\rho = 1$ when two rankings are in perfect agreement and $\rho = -1$ when they are in perfect disagreement.