Classification-Aware Hidden-Web Text Database Selection

PANAGIOTIS G. IPEIROTIS
New York University

LUIS GRAVANO
Columbia University

March 13, 2007

Abstract

Many valuable text databases on the web have non-crawlable contents that are "hidden" behind search interfaces. Metasearchers are helpful tools for searching over multiple such "hidden-web" text databases at once through a unified query interface. An important step in the metasearching process is database selection, or determining which databases are the most relevant for a given user query. The state-of-the-art database selection techniques rely on statistical summaries of the database contents, generally including the database vocabulary and the associated word frequencies. Unfortunately, hidden-web text databases typically do not export such summaries, so previous research has developed algorithms for constructing *approximate* content summaries from document samples extracted from the databases via querying. We present a novel "focused probing" sampling algorithm that detects the topics covered in a database and adaptively extracts documents that are representative of the topic coverage of the database. Our algorithm is the first that constructs content summaries that include the frequencies of the words in the database. Unfortunately, Zipf's law practically guarantees that, for any relatively large database, content summaries built from moderately sized document samples will fail to cover many low-frequency words; in turn, incomplete content summaries might negatively affect the database selection process, especially for short queries with infrequent words. To enhance the sparse document samples and improve the database selection decisions, we exploit the fact that topically similar databases tend to have similar vocabularies, so samples extracted from databases with a similar topical focus can complement each other. We have developed two database selection algorithms that exploit this observation. The first algorithm proceeds hierarchically and selects the best category for a query, and then sends the query to the appropriate databases in the chosen category. The second algorithm uses "shrinkage," a statistical technique for improving parameter estimation in the face of sparse data, to enhance the database content summaries with category-specific words. We describe how to modify existing database selection algorithms to adaptively decide –at run-time– whether shrinkage is beneficial for a query. A thorough evaluation over a variety of databases, including 315 real web databases as well as TREC data, suggests that the proposed sampling methods generate high-quality content summaries and the database selection algorithms produce significantly more relevant database selection decisions and overall search results than existing algorithms.

1 Introduction

The World-Wide Web continues to grow rapidly, which makes exploiting all useful information that is available a standing challenge. Although general Web search engines crawl and index a large amount of information, typically they ignore valuable data in text databases that are "hidden" behind search interfaces and whose contents are not directly available for crawling through hyperlinks.

Example 1.1 *Consider the U.S. Patent and Trademark (USPTO) database, which contains*[1] *the full text of all the patents awarded in the US since 1976.*[2] *If we query*[3] *USPTO for patents with the keywords "wireless" and "network", USPTO returns 36,686 matches as of June 22nd, 2005, corresponding to distinct patents that contain these keywords. In contrast, a query*[4] *on Google that finds the pages in the USPTO database*

[1]The full text of the patents is stored at the USPTO site.
[2]The query interface is available at `http://patft.uspto.gov/netahtml/search-adv.htm`.
[3]The query is [wireless AND network].
[4]The query is [wireless network site:patft.uspto.gov].

with the keywords "wireless" and "network" returns 0 matches as of June 22nd, 2005, which illustrates that the valuable content available through the USPTO database is ignored by this search engine.

One way to provide one-stop access to the information in text databases is through *metasearchers*, which can be used to query multiple databases simultaneously. A metasearcher performs three main tasks. After receiving a query, it finds the best databases to evaluate the query (*database selection*), it translates the query in a suitable form for each database (*query translation*), and finally it retrieves and merges the results from the different databases (*result merging*) and returns them to the user. The *database selection* component of a metasearcher is of crucial importance in terms of both query processing efficiency and effectiveness.

Database selection algorithms are traditionally based on statistics that characterize each database's contents [YL97, XC98, MLY+98, GGMT99]. These statistics, to which we will refer as *content summaries*, usually include the *document frequencies* of the words that appear in the database, plus perhaps other simple statistics. These summaries provide sufficient information to the database selection component of a metasearcher to decide which databases are the most promising to evaluate a given query.

Constructing the content summary of a text database is a simple task if the full contents of the database are available (e.g., via crawling). However, this task is challenging for the so-called "*hidden-Web text databases*," whose contents are only available via querying. In this case, a metasearcher could rely on the databases to supply the summaries (e.g., by following a protocol like STARTS [GCGMP97], or possibly using Semantic Web [BLHL01] tags in the future). Unfortunately, many Web-accessible text databases are completely autonomous and do not report any detailed metadata about their contents to facilitate metasearching. To handle such databases, a metasearcher could rely on manually generated descriptions of the database contents. Such an approach would not scale to the thousands of text databases available on the Web [Bri00], and would likely not produce the good-quality, fine-grained content summaries required by database selection algorithms.

In this article, we first present a technique to automate the extraction of high-quality content summaries from hidden-Web text databases. Our technique constructs these summaries from a *biased sample* of the documents in a database, extracted by adaptively *probing* the database using the topically focused queries sent to the database during classification. Our algorithm selects what queries to issue based in part on the results of the earlier queries, thus focusing on the topics that are most representative of the database in question. Our technique resembles biased sampling over *numeric* databases, which focuses the sampling effort on the "densest" areas. We show that this principle is also beneficial for the text-database world. Interestingly, our technique moves beyond the document sample and attempts to include in the content summary of a database accurate estimates of the actual document frequency of the words in the database. For this, our technique exploits well studied statistical properties of text collections.

Unfortunately, all efficient techniques for building content summaries via document sampling suffer from a "sparse-data" problem: many words in any text database tend to occur in relatively few documents, so any document sample of reasonably small size will necessarily miss many words that occur in the associated database a small number of times. To alleviate this sparse-data problem, we exploit the observation –which we validate experimentally– that incomplete content summaries of topically related databases can be used to complement each other. Based on this observation, we explore two alternative algorithms that make database selection more resilient to incomplete content summaries. Our first algorithm selects databases hierarchically based on the categorization of the databases. The algorithm first chooses the categories to explore for a query and then picks the best databases in the most appropriate categories. Our second algorithm is a "flat" selection strategy that exploits the database categorization implicitly, by using "shrinkage," a statistical technique for improving parameter estimation in the face of sparse data. Our shrinkage-based algorithm enhances the database content summaries with category-specific words. As we will see, the shrinkage-enhanced summaries often characterize the database contents better than their "unshrunk" counterparts do. Then, during database selection, our algorithm decides in an adaptive and query-specific way whether the application of shrinkage is beneficial.

We evaluate the performance of our content summary construction algorithms using a variety of databases, including 315 real Web databases. We also evaluate our database selection strategies with extensive experiments that involve text databases and queries from the TREC testbed, together with the relevance judgments associated with the queries and the database documents. We compare our methods with a variety of state-of-the-art database selection algorithms. As we will see, our techniques result in a significant

CANCERLIT 3,801,351 documents	
Word	*df*
breast	181,102
cancer	1,893,838
...	...

CNN Money 13,313 documents	
Word	*df*
breast	65
cancer	255
...	...

Table 1: A fragment of the content summaries of two databases.

improvement in performance over the state of the art, achieved *efficiently* just by exploiting the database classification information and without increasing the document-sample size.

In brief, the main contributions presented in this article are:

- A technique to *sample* text databases that results in higher quality database content summaries than those produced by the state-of-the-art alternatives.
- A technique to estimate the *absolute* document frequencies of the words in the content summaries.
- A technique to improve the quality of sample-based content summaries using *shrinkage*.
- A *hierarchical database selection algorithm* that works over a topical classification scheme.
- An *adaptive database selection algorithm* that decides in an adaptive and query-specific way whether to use the shrinkage-based content summaries.
- A thorough, extensive experimental evaluation of the presented algorithms using a variety of data sets, including TREC data and 315 real Web databases.

The rest of the article is organized as follows. Section 2 gives the necessary background. Section 3 outlines our new technique for producing content summaries of text databases and presents our frequency estimation algorithm. Section 4 describes our hierarchical and our shrinkage-based database selection algorithms, which build on our observation that topically similar databases have similar content summaries. Section 5 describes the settings for the experimental evaluation of Sections 6 and 7. Finally, Section 8 describes related work and Section 9 concludes the article.

2 Background

In this section, we provide the required background as well as describe related efforts. Section 2.1 briefly summarizes how existing database selection algorithms work, stressing their reliance on database "content summaries." Then, Section 2.2 describes the use of "uniform" query probing for extraction of content summaries from text databases and identifies the limitations of this technique. Finally, Section 2.3 discusses how focused query probing has been used in the past for the classification of text databases.

2.1 Database Selection Algorithms

Database selection is an important task in the metasearching process, since it has a critical impact on the efficiency and effectiveness of query processing over multiple text databases. We now briefly outline how typical database selection algorithms work and how they depend on database content summaries to make decisions.

A database selection algorithm attempts to find the best text databases to evaluate a given query, based on information about the database contents. Usually this information includes the number of different documents that contain each word, to which we refer as the *document frequency* of the word, plus perhaps some other simple related statistics [GCGMP97, MLY+98, XC98], such as the number of documents stored in the database.

Definition 2.1 *The* content summary $S(D)$ *of a database* D *consists of:*

- *The actual number of documents in* D, $|D|$, *and*

- *For each word w, the number $df(w)$ of documents in D that include w.*

For notational convenience, we also use $p(w|D) = \frac{df(w)}{|D|}$ to denote the fraction of D documents that include w.

Table 1 shows a small fraction of what the content summaries for two real text databases might look like. For example, the content summary for the *CNN Money* database, a database with articles about finance, indicates that 255 out of the 13,313 documents in this database contain the word "cancer," while there are 1,893,838 documents with the word "cancer" in *CANCERLIT*, a database with research articles about cancer. Given these summaries, a database selection algorithm estimates the relevance of each database for a given query (e.g., in terms of the number of matches that each database is expected to produce for the query):

Example 2.2 *bGlOSS [GGMT99] is a simple database selection algorithm that assumes that query words are independently distributed over database documents to estimate the number of documents that match a given query. So, bGlOSS estimates that query* [breast cancer] *will match $|D| \cdot \frac{df(breast)}{|D|} \cdot \frac{df(cancer)}{|D|} \cong 90,225$ documents in database* CANCERLIT*, where $|D|$ is the number of documents in the* CANCERLIT *database and $df(\cdot)$ is the number of documents that contain a given word. Similarly, bGlOSS estimates that only 1 document will match the given query in the other database,* CNN Money*, of Table 1.*

bGlOSS is a simple example of a large family of database selection algorithms that rely on content summaries such as those in Table 1. Furthermore, database selection algorithms expect content summaries to be accurate and up to date. The most desirable scenario is when each database exports its content summary directly and reliably (e.g., via a protocol such as STARTS [GCGMP97]). Unfortunately, no protocol is widely adopted for Web-accessible databases, and there is little hope that such a protocol will emerge soon. Hence, we need other solutions to automate the construction of content summaries from databases that cannot or are not willing to export such information. We review one such approach next.

2.2 Uniform Probing for Content Summary Construction

As discussed above, we cannot extract perfect content summaries for hidden-Web text databases whose contents are not crawlable. When we do not have access to the complete content summary $S(D)$ of a database D, we can only hope to generate a good approximation and use it for database selection purposes.

Definition 2.3 *The* approximate content summary $\hat{S}(D)$ *of a database D consists of:*

- *An estimate $\widehat{|D|}$ of the number of documents in D, and*
- *For each word w, an estimate $\widehat{df}(w)$ of $df(w)$.*

Using the values $\widehat{|D|}$ and $\widehat{df}(w)$, we can define an approximation $\hat{p}(w|D)$ of $p(w|D)$ as $\hat{p}(w|D) = \frac{\widehat{df}(w)}{\widehat{|D|}}$.

Callan et al. [CCD99, CC01] presented pioneering work on automatic extraction of approximate content summaries from "uncooperative" text databases that do not export such metadata. Their algorithm extracts a document sample via querying from a given database D and approximates $df(w)$ using the frequency of each observed word w in the sample, $sf(w)$ (i.e., $\widehat{df}(w) = sf(w)$). In detail, the algorithm proceeds as follows:

1. Start with an empty content summary where $sf(w) = 0$ for each word w, and a general (i.e., not specific to D), comprehensive word dictionary.
2. Pick a word (see below) and send it as a query to database D.
3. Retrieve the top-k documents returned for the query.
4. If the number of retrieved documents exceeds a prespecified threshold, stop. Otherwise continue the sampling process by returning to Step 2.

Callan et al. suggested using $k = 4$ for Step 3 and that 300 documents are sufficient (Step 4) to create a representative content summary of a database. Also they describe two main versions of this algorithm that differ in how Step 2 is executed. The algorithm *QueryBasedSampling-OtherResource* (*QBS-Ord* for short) picks a random word from the dictionary for Step 2. In contrast, the algorithm *QueryBasedSampling-LearnedResource* (*QBS-Lrd* for short) selects the next query from among the words that have been already discovered during sampling. *QBS-Ord* constructs better profiles, but is more expensive than *QBS-Lrd* [CC01]. Other variations of this algorithm perform worse than *QBS-Ord* and *QBS-Lrd*, or have only marginal improvements in effectiveness at the expense of probing cost.

Unfortunately, both *QBS-Lrd* and *QBS-Ord* have a few shortcomings. Since these algorithms set $\widehat{df}(w) = sf(w)$, the approximate frequencies $\widehat{df}(w)$ range between one and the number of retrieved documents in the sample. In other words, the actual document frequency $df(w)$ for each word w in the database is not revealed by this process. Hence, two databases with the same focus (e.g., two medical databases) but differing significantly in size might be assigned similar content summaries. Also, *QBS-Ord* tends to produce inefficient executions in which it repeatedly issues queries to databases that produce no matches. According to Zipf's law [Zip49], most of the words in a collection occur very few times. Hence, a word that is randomly picked from a dictionary (which hopefully contains a superset of the words in the database), is likely not to occur in any document of an arbitrary database. Similarly, for *QBS-Lrd*, the queries are derived from the already acquired vocabulary, and many of these words appear only in one or two documents, so a large fraction of the *QBS-Lrd* queries return only documents that have been retrieved before. These queries increase the number of queries sent by *QBS-Lrd*, but do not retrieve any new documents. In Section 3, we present our algorithm for approximate content summary construction that overcomes these problems and, as we will see, produces content summaries of higher quality than those produced by *QBS-Ord* and *QBS-Lrd*.

2.3 Focused Probing for Database Classification

Another way to characterize the contents of a text database is to classify it in a Yahoo!-like hierarchy of topics according to the type of the documents that it contains. For example, *CANCERLIT* can be classified under the category *"Health,"* since it contains mainly health-related documents. Gravano et al. [GIS03] presented a method to automate the classification of Web-accessible text databases, based on *"focused probing."*

The rationale behind this method is that queries closely associated with topical categories retrieve mainly documents about that category. For example, a query *[breast cancer]* is likely to retrieve mainly documents that are related to the *"Health"* category. Gravano et al. [GIS03] automatically construct these topic-specific queries using document classifiers, derived via supervised machine learning. By observing the number of matches generated for each such query at a database, we can place the database in a classification scheme. For example, if one database generates a large number of matches for the queries associated with the *"Health"* category and only a few matches for all other categories, we might conclude that this database should be under category *"Health."* If the database does not return the number of matches for a query or does so unreliably, we can still classify the database by retrieving and classifying a sample of documents from the database. Gravano et al. [GIS03] showed that a sample-based classification has both lower accuracy and higher cost than the algorithm that relies on the number of matches; however, in the absence of reliable matching statistics, classifying the database based on a document sample is a viable alternative.

To classify a database, the algorithm in [GIS03] (see Figure 1) starts by first sending the query probes associated with the subcategories of the top node C of the topic hierarchy, and extracting the number of matches for each probe, without retrieving any documents. Based on the number of matches for the probes for each subcategory C_i, the classification algorithm then calculates two metrics, the *Coverage*(D, C_i) and *Specificity*(D, C_i) for the subcategory: *Coverage*(D, C_i) is the *absolute* number of documents in D that are estimated to belong to C_i, while *Specificity*(D, C_i) is the *fraction* of documents in D that are estimated to belong to C_i. The algorithm decides to classify D into a category C_i if the values of *Coverage*(D, C_i) and *Specificity*(D, C_i) exceed two prespecified thresholds τ_{ec} and τ_{es}, respectively. These thresholds are determined by "editorial" decisions on how "coarse" a classification should be. For example, higher levels of the specificity threshold τ_{es} result in assignments of databases mostly to higher levels of the hierarchy, while lower values tend to assign the databases to nodes closer to the leaves.[5] When the algorithm detects that a database satisfies the specificity and coverage requirement for a subcategory C_i, it proceeds recursively in

[5]Gravano et al. [GIS03] suggest that $\tau_{ec} \approx 10$ and $\tau_{es} \approx 0.3 - 0.4$ work well for the task of database classification.

```
Classify(Category C, Database D, τec, τes, Specificity(D, C))
    Result = ∅
    if C is a leaf node
        then return {C}
    Probe database D with the query probes for the subcategories of C
    Calculate Coverage from the number of matches for the probes
    Coverage=M^-1 × Coverage // Confusion Matrix Adjustment [GIS03]
    for each subcategory Ci of C
        Calculate Specificity(D, Ci) using Coverage and Specificity(D, C)
        if Specificity(D, Ci) ≥ τes AND Coverage(D, Ci) ≥ τec
            then Result = Result ∪ Classify(Ci, D, τec, τes, Specificity(D, Ci))
    if Result == ∅
        then return {C} // D was not "pushed" down
        else return Result
```

Figure 1: Algorithm for classifying a database D into the category subtree rooted at category C.

the subtree rooted at C_i. By not exploring other subtrees that did not satisfy the coverage and specificity conditions, the algorithm avoids exploring portions of the topic space that are not relevant to the database.

Next, we introduce a novel technique for constructing content summaries that are highly accurate and efficient to build. Our new technique builds on the document sampling approach used by the *QBS* algorithms [CC01] and on the text-database classification algorithm from [GIS03]. Just as *QBS*, which we summarized in Section 2.2, our new technique probes the databases and retrieves a small document sample to construct the approximate content summaries. The classification algorithm, which we summarized in this section, provides a way to focus on the topics that are most representative of a given database's contents, resulting in turn in accurate and efficiently extracted content summaries.

3 Constructing Approximate Content Summaries

We now describe our algorithm for constructing content summaries for a text database. Our algorithm exploits a topic hierarchy to adaptively send focused probes to the database (Section 3.1). Our technique retrieves a "biased" sample that contains documents that are representative of the database contents. Furthermore, our algorithm exploits the number of matches reported for each query to estimate the absolute document frequencies of the words in the database (Section 3.2).

3.1 Classification-Based Document Sampling

Our algorithm for approximate content summary construction exploits a topic hierarchy to adaptively send focused probes to a database. These queries tend to efficiently produce a document sample that is representative of the database contents, which leads to highly accurate content summaries. Furthermore, our algorithm classifies the databases along the way. In Section 4, we will show that we can exploit categorization to improve further the quality of both the generated content summaries and the database selection decisions.

Our content summary construction algorithm is based on the classification algorithm from [GIS03], an outline of which we presented in Section 2.3 (see Figure 1). Our content summary construction algorithm is shown in Figure 2. The main difference with the classification algorithm is that we exploit the focused probing to retrieve a document sample. We have enclosed in boxes the portions directly relevant to content summary extraction. Specifically, for each query probe we retrieve k documents from the database in addition to the number of matches that the probe generates (box β in Figure 2). Also, we record two sets of word frequencies based on the probe results and extracted documents (boxes β and γ):

1. $df(w)$: the actual number of documents in the database that contain word w. The algorithm knows this number only if $[w]$ is a single-word query probe that was issued to the database.[6]

[6]The number of matches reported by a database for a single-word query $[w]$ might differ slightly from $df(w)$, for example, if the database applies stemming [SM83] to query words so that a query [*computers*] also matches documents with word *"computer."*

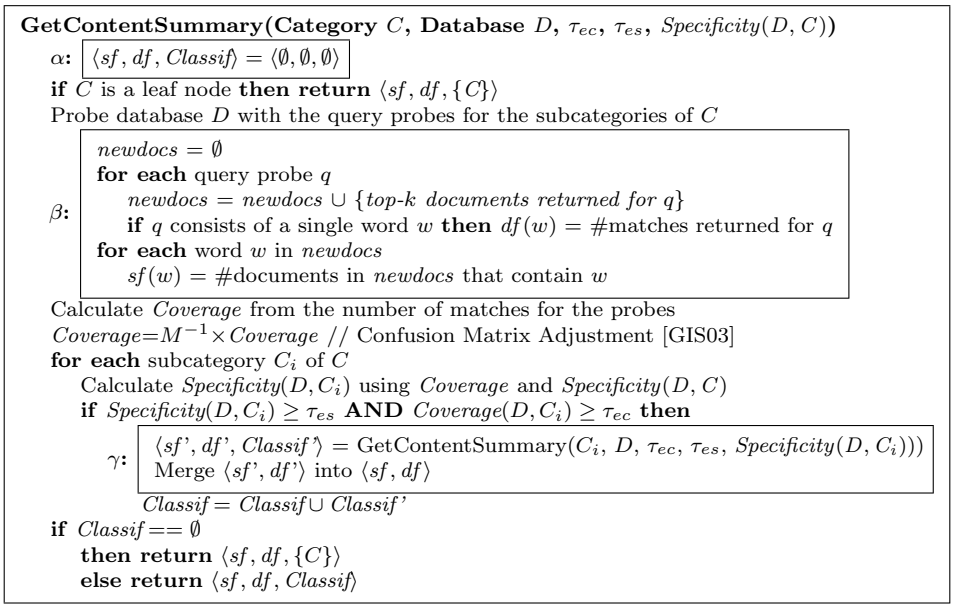

Figure 2: Generalizing the classification algorithm from Figure 1 to generate a content summary for a database using focused query probing.

2. $sf(w)$: the number of documents in the extracted sample that contain word w.

The basic structure of the probing algorithm is as follows. We explore (and send query probes for) only those categories with sufficient specificity and coverage, as determined by the τ_{es} and τ_{ec} thresholds (for details, see Section 2.3). As a result, this algorithm categorizes the databases into the classification scheme during probing. We will exploit this categorization to improve the quality of the generated content summaries in Section 4.2.

Figure 3 illustrates how our algorithm works for the *CNN Sports Illustrated* database, a database with articles about sports, and for a toy hierarchical scheme with four categories under the root node: *"Sports," "Health," "Computers,"* and *"Science."* We pick specificity and coverage thresholds $\tau_{es} = 0.4$ and $\tau_{ec} = 10$, respectively, which work well for the task of database classification [GIS03]. The algorithm starts by issuing the query probes associated with each of the four categories. The *"Sports"* probes generate many matches (e.g., query *[baseball]* matches 24,520 documents). In contrast, the probes for the other sibling categories (e.g., *[metallurgy]* for category *"Science"*) generate just a few or no matches. The *Coverage* of category *"Sports"* is the sum of the number of matches for its probes, or 32,050. The *Specificity* of category *"Sports"* is the fraction of matches that correspond to *"Sports"* probes, or 0.967. Hence, *"Sports"* satisfies the *Specificity* and *Coverage* criteria (recall that $\tau_{es} = 0.4$ and $\tau_{ec} = 10$) and is further explored to the next level of the hierarchy. In contrast, *"Health," "Computers,"* and *"Science"* are not considered further. By *pruning* the probe space, we improve the efficiency of the probing process by giving attention to the topical focus (or foci) of the database. (Out-of-focus probes would tend to return few or no matches.)

During probing, our algorithm retrieves the top-k documents returned by each query (box β in Figure 2). For each word w in a retrieved document, the algorithm computes $sf(w)$ by measuring the number of documents in the sample, extracted in a probing round, that contain w. If a word w appears in document samples retrieved during later phases of the algorithm for deeper levels of the hierarchy, then all $sf(w)$ values are added together ("merge" step in box γ). Similarly, during probing the algorithm keeps track of the number of matches produced by each single-word query $[w]$. As discussed, the number of matches for such a query is (an approximation of) the $df(w)$ frequency (i.e., the number of documents in the database with word w). These $df(\cdot)$ frequencies are crucial to estimate the absolute document frequencies of all words that appear in the document sample extracted, as discussed next.

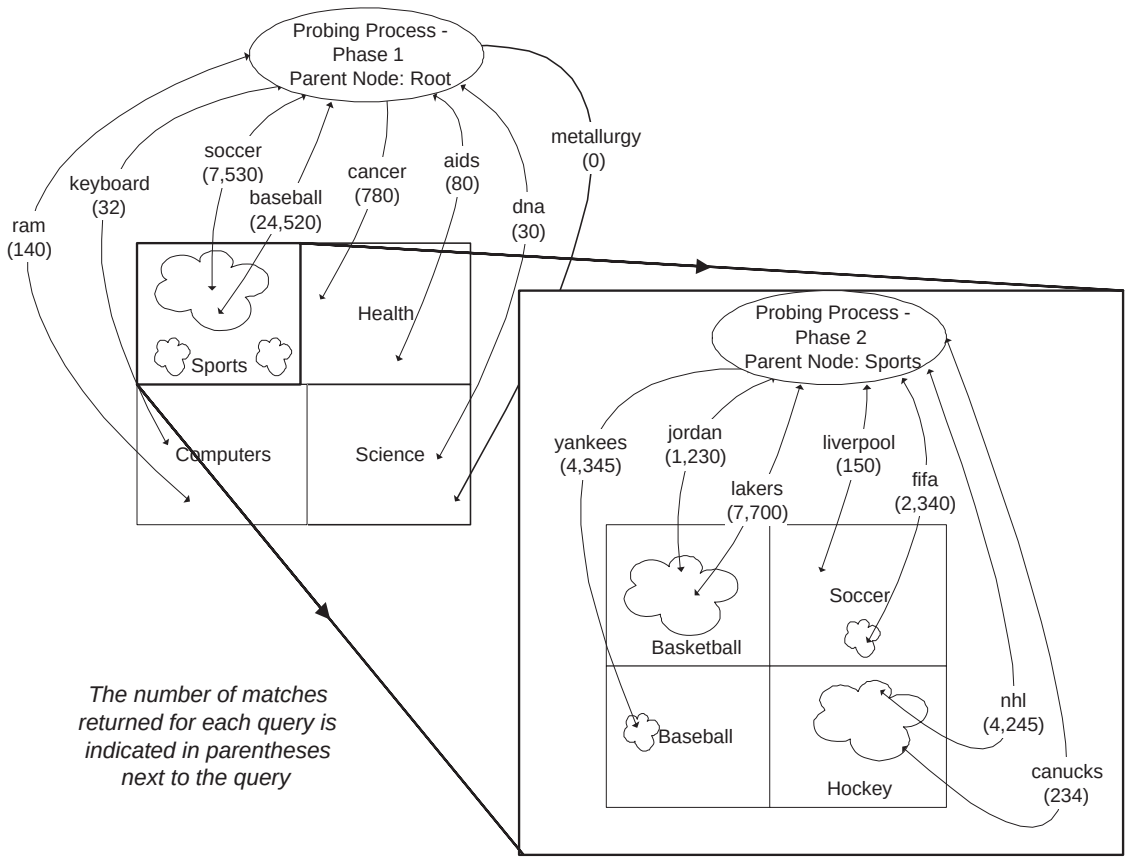

Figure 3: Querying the *CNN Sports Illustrated* database with focused probes.

3.2 Estimating Absolute Document Frequencies

The *QBS-Ord* and *QBS-Lrd* techniques return the frequency of the words in the document sample (i.e., the $sf(\cdot)$ frequencies), with no absolute frequency information. We now show how we can exploit the $df(\cdot)$ and $sf(\cdot)$ document frequencies that we extract from a database to build a content summary for the database with accurate absolute document frequencies.

Before turning to the details of the algorithm, we describe a (simplified) example in Figure 4 to introduce the basic intuition behind our approach.[7] After probing the *CANCERLIT* database using the algorithm in Figure 2, we rank all words in the extracted documents according to their $sf(\cdot)$ frequency. For example, *"cancer"* has the highest $sf(\cdot)$ value and *"hepatitis"* the lowest such value in Figure 4. The $sf(\cdot)$ value of each word is noted by an associated vertical bar. Also, the figure shows the $df(\cdot)$ frequency of each word that appeared as a single-word query. For example, $df(hepatitis) = 200,000$, because query probe *[hepatitis]* returned 200,000 matches. Note that the *df* value of some words (e.g., *"stomach"*) is unknown. These words are in documents retrieved during probing, but did not appear as single-word probes. Finally, note from the figure that $sf(hepatitis) \approx sf(stomach)$, and so we might want to estimate $df(stomach)$ to be close to the (known) value of $df(hepatitis)$.

To specify how to "propagate" the known *df* frequencies to "nearby" words with similar *sf* frequencies, we exploit well known laws on the distribution of words over text documents. Zipf [Zip49] was the first to observe that word-frequency distributions follow a power law, an observation that was later refined by Mandelbrot [Man88]. Mandelbrot identified a relationship between the rank r and the frequency f of a word in a text database, $f = P(r + p)^B$, where P, B, and p are database-specific parameters ($P > 0$, $B < 0$, $p \geq 0$). This formula indicates that the most frequent word in a collection (i.e., the word with rank $r = 1$)

[7] The figures in this example are coarse approximations of the real ones, and we use them just to illustrate our approach.

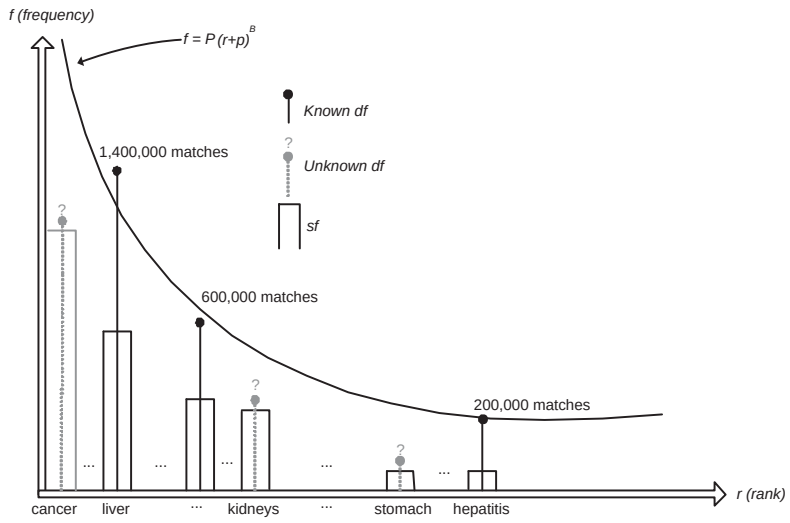

Figure 4: Estimating unknown *df* values.

will tend to appear in about $P(1+p)^B$ documents, while, say, the tenth most frequent word will appear in just about $P(10+p)^B$ documents. Therefore, given Mandelbrot's formula for the database and the word ranking, we can estimate the frequency of each word.

Our technique relies on Mandelbrot's formula to define the content summary of a database and consists of two steps:

1. During probing, exploit the $sf(\cdot)$ frequencies derived during sampling to estimate the rank-frequency distribution of the words over the entire database (Section 3.2.1).

2. After probing, exploit the $df(\cdot)$ frequencies obtained from one-word query probes to estimate the rank of these words in the actual database; then, estimate the document frequencies of all words by "propagating" the known rank and document frequencies to "nearby" words w for which we only know $sf(w)$ but not $df(w)$ (Section 3.2.2).

3.2.1 Estimating the Word Rank-Frequency Distribution

The first part of our technique estimates the parameters P and B (of a slightly simplified version[8]) of Mandelbrot's formula for a given database. To do this, we examine how the parameters of Mandelbrot's formula change for different sample sizes. We observed that in all the databases that we examined for our experiments, $\log(P)$ and B tend to increase logarithmically with the sample size $|S|$. (This is actually an effect of sampling from a power-law distribution [Baa06].) Specifically:

$$\log(P) = P_1 \log(|S|) + P_2 \tag{1a}$$

$$B = B_1 \log(|S|) + B_2 \tag{1b}$$

and P_1, P_2, B_1, and B_2 are database-specific constants, independent of sample size.

Based on the above empirical observations, we proceed as follows for a database D. At different points during the document sampling process, we calculate P and B. After sampling, we use regression to estimate the values of P_1, P_2, B_1, and B_2. We also estimate the size of database D using the "sample-resample" method [SC03] with five resampling queries. Finally, we compute the values of P and B for the database by substituting the estimated $|D|$ for $|S|$ in Equations 1a and 1b. At this point, we have a description of the frequency-rank distribution for the *actual* database.

[8]For numerical stability, we define $f = Pr^B$, which allows us to use linear regression in the log-log space to estimate parameters P and B.

3.2.2 Estimating Document Frequencies

Given the parameters of Mandelbrot's formula, the actual document frequency $df(w)$ of each word w can be derived from its rank in the database. For high-frequency words, the rank in the sample is usually a good approximation of the rank in the database. Unfortunately, this is rarely the case for low-frequency words, for which we rely on the observation that $df(\cdot)$ frequencies derived from one-word query probes can help estimate the rank and $df(\cdot)$ frequency of all words in the database. Our rank and frequency estimation algorithm works as follows:

1. Sort words in descending order of their $sf(\cdot)$ frequencies to determine the *sample rank* $sr(w_i)$ of each word w_i; do not break ties for words with equal $sf(\cdot)$ frequency and assign the same sample rank $sr(\cdot)$ to these words.
2. For each word w in a one-word query probe ($df(w)$ is known), use Mandelbrot's formula and compute the *database rank* $ar(w) = \left(\frac{df(w)}{P}\right)^{\frac{1}{B}}$.
3. For each word w not in a one-word query probe ($df(w)$ is unknown):
   (a) Find two words w_1 and w_2 with known df and consider their ranks in the sample (i.e., $sr(w_1)$, $sr(w_2)$) and in the database (i.e., $ar(w_1)$, $ar(w_2)$).[9]
   (b) Use interpolation in the log-log space to compute the database rank $ar(w)$.[10]
   (c) Use Mandelbrot's formula to compute $\widehat{df}(w) = P \cdot ar(w)^B$, where $ar(w)$ is the rank of word w as computed in the previous step.

Using the procedure above, we can estimate the df frequency of each word that appears in the sample.

Example 3.1 *Consider the medical database* CANCERLIT *and Figure 4. We know that df(liver)* $= 1,400,000$ *and df(hepatitis)* $= 200,000$*, since the respective one-word queries reported as many matches. Furthermore, the ranks of the two words in the sample is sr(liver)* $= 4$ *and sr(hepatitis)* $= 10$*, respectively. While we know that the rank of the word* "kidneys" *in the sample is sr(kidneys)* $= 8$*, we do not know df(kidneys) because* [kidneys] *was not a query probe. However, the known values of df(hepatitis) and df(liver) can help us estimate the rank of* "kidneys" *in the database and, in turn, the df(kidneys) frequency. For the* CANCERLIT *database, we estimate that* $P = 6 \cdot 10^6$ *and* $B = -1.15$*. Thus, we estimate that* "liver" *is the fourth most frequent word in the database (i.e.,* $ar(liver) = 4$*), while* "hepatitis" *is ranked number 20 (i.e.,* $ar(hepatitis) = 20$*). Therefore, 15 words in the database are ranked between* "liver" *and* "hepatitis"*, while in the sample there are only 5 such words. By exploiting this observation and by interpolation, we estimate that* "kidneys" *(with rank 8 in the sample) is the 14th most frequent word in the database. Then, using the rank information with Mandelbrot's formula, we compute* $\widehat{df}(kidneys) = 6 \cdot 10^6 \cdot 14^{-1.15} \cong 288,472$.

During sampling, we also send to the database query probes that consist of more than one word. (Recall that our query probes are derived from an underlying automatically learned document classifier.) We do not exploit multi-word queries for determining df frequencies of their words, since the number of matches returned by a boolean-AND multi-word query is only a lower bound on the df frequency of each intervening word. However, the average length of the query probes that we generate is small (less than 1.5 words in our experiments), and their median length is one. Hence, the majority of the query probes provide us with df frequencies that we can exploit.

Finally, a potential problem with the current algorithm is that it relies on the database reporting a value for the number of matches for a one-word query $[w]$ that is equal (or at least close) to the value of $df(w)$. Sometimes, however, these two values might differ (e.g., if a database applies stemming to query words). In this case, frequency estimates might not be reliable. However, it is rather easy to detect such configurations [MYL99] and adapt the frequency estimation algorithm properly. For example, if we detect that a database uses stemming, we might decide to compute the frequency and rank of each word in the sample after the application of stemming and adjust the algorithms accordingly.

[9] It is preferable, but not essential, to pick w_1 and w_2 such that $sr(w_1) < sr(w) < sr(w_2)$.

[10] The exact formula is $ar(w) = \exp(\frac{\ln(ar(w_2)) \cdot \ln(sr(w)/sr(w_1)) + \ln(ar(w_1)) \cdot \ln(sr(w_2)/sr(w))}{\ln(sr(w_2)/sr(w_1))})$.

In summary, we have presented a novel technique for estimating the absolute document frequency of the words in a database. As we will see, this technique produces relatively accurate frequency estimates for the words in a document sample of the database. However, the database words that are not in the sample documents in the first place are ignored and are not part of the resulting content summary. Unfortunately, any document sample of moderate size will necessarily miss many words that occur in the associated database only a small number of times. The absence of these words from the content summaries can negatively affect the performance of database selection algorithms for queries that mention such words. To alleviate this sparse-data problem, we exploit the observation that incomplete content summaries of topically related databases can be used to complement each other, as discussed next.

4 Database Selection with Sparse Content Summaries

So far, we discussed how to efficiently construct approximate content summaries using document sampling. However, any efficient algorithm for constructing content summaries through query probes is likely to produce incomplete content summaries, which can adversely affect the effectiveness of the database selection process. To alleviate this sparse-data problem, we exploit the observation that incomplete content summaries of topically related databases can be used to complement each other. In this section, we present two alternative algorithms that exploit this observation and make database selection more resilient to incomplete content summaries. Our first algorithm (Section 4.1) selects databases hierarchically based on the categorization of the databases. Our second algorithm (Section 4.2) is a "flat" selection strategy that exploits the database categorization implicitly, by using "shrinkage," and enhances the database content summaries with category-specific words that appear in topically similar databases.

4.1 Hierarchical Database Selection

We now introduce a hierarchical database selection algorithm that exploits the database categorization and content summaries to alleviate the negative effect of incomplete content summaries. This algorithm consists of two basic steps:

1. "Propagate" the database content summaries to the categories of the hierarchical classification scheme and create the associated category content summaries using Definition 4.1.

2. Use the content summaries of categories and databases to perform database selection hierarchically by zooming in on the most relevant portions of the topic hierarchy.

The intuition behind our approach is that databases classified under similar topics tend to have similar vocabularies. (We present supporting experimental evidence for this statement in Section 6.2.) Hence, we can view the (potentially incomplete) content summaries of all databases in a category as complementary, and exploit this for better database selection. For example, consider the *CANCER.gov* database and its associated content summary in Figure 5. As we can see, *CANCER.gov* was correctly classified under *"Cancer"* by the algorithm of Section 3.1. Unfortunately, the word *"metastasis"* did not appear in any of the documents extracted from *CANCER.gov* during probing, so this word is missing from the content summary. However, we see that *CancerBACUP*[11], another database classified under *"Cancer"*, has $\widehat{df}(\textit{metastasis}) = 3,569$, a relatively high value. Hence, we might conjecture that the word *"metastasis"* is an important word for all databases in the *"Cancer"* category and that this word did not appear in *CANCER.gov* because it was not discovered during sampling, not because it does not occur in the database. Therefore, we can create a content summary with category *"Cancer"* in such a way that the word *"metastasis"* appears with a relatively high frequency. This summary is obtained by merging the summaries of all databases under the category.

In general, we define the content summary of a category as follows:

Definition 4.1 *Consider a category C and the set $db(C) = \{D_1, \ldots, D_n\}$ of databases classified (not necessary immediately) under C.*[12] *The* approximate content summary $\hat{S}(C)$ *of category C contains, for each word*

[11] `http://www.cancerbacup.org.uk`

[12] If a database D_i is classified under multiple categories, we can treat D_i as multiple disjoint "sub-databases," with each sub-database being associated with one of the D_i categories and containing only the documents in the respective category.

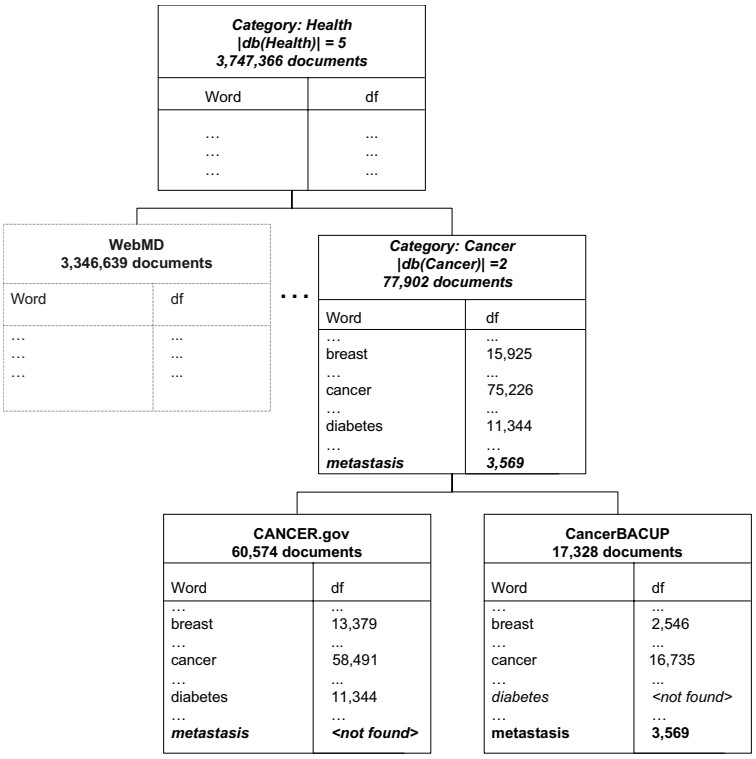

Figure 5: Associating content summaries with categories.

w, an estimate $\hat{p}(w|C)$ of $p(w|C)$, where $p(w|C)$ is the probability that a randomly selected document from a database in $db(C)$ contains the word w. The $\hat{p}(w|C)$ estimates in $\hat{S}(C)$ are derived from the approximate content summaries of the databases in $db(C)$ as:[13]

$$\hat{p}(w|C) = \frac{\sum_{D \in db(C)} \hat{p}(w|D) \cdot \widehat{|D|}}{\sum_{D \in db(C)} \widehat{|D|}} \tag{2}$$

where $\widehat{|D|}$ is an estimate of the number of documents in D (see Definition 2.3)[14]*. The approximate content summary $\hat{S}(C)$ also includes:*

- *The number of databases $|db(C)|$ under C (n in this definition).*
- *An estimate $\widehat{|C|} = \sum_{D \in db(C)} \widehat{|D|}$ of the number of documents in all databases under C.*
- *For each word w, an estimate $\widehat{df}_C(w)$ of the total number of documents under C that contain the word w: $\widehat{df}_C(w) = \hat{p}(w|C) \cdot \widehat{|C|}$.*

By having content sumaries associated with categories in the topic hierarchy, we can select databases for a query by proceeding hierarchically from the root category. At each level, we use existing flat database algorithms such as CORI [CLC95] or bGlOSS [GGMT99]. These algorithms assign a *score* to each database (or category in our case), which specifies how promising the database (or category) is for the query, as indicated by the content summaries (see Example 2.2). Given the scores for the categories at one level of the hierarchy, the selection process continues recursively down the most promising subcategories. As further

[13] An alternative is to define $\hat{p}(w|C) = \frac{\sum_{D \in db(C)} \hat{p}(w|D)}{|db(C)|}$, which "weights" each database equally, regardless of its size. We implemented this alternative and obtained results that were virtually identical to those for Equation 2.

[14] We estimate the number of documents in the database as described in Section 3.2.1.

HierSelect(Query q, Category C, int K)
1: Use a database selection algorithm to assign a score for q to each subcategory of C
2: **if** there is a subcategory of C with a non-zero score
3: Pick the subcategory C_j with the highest score
4: **if** $|db(C_j)| \geq K$ //C_j *has enough databases*
5: **return** HierSelect(q,C_j,K)
6: **else** // C_j *does not have enough databases*
7: **return** $db(C_j) \cup$ FlatSelect(q,$C - C_j$,$K - |db(C_j)|$)
8: **else** // *no subcategory of* C *has non-zero score*
9: **return** FlatSelect(q,C,K)

Figure 6: Selecting the K most specific databases for a query hierarchically.

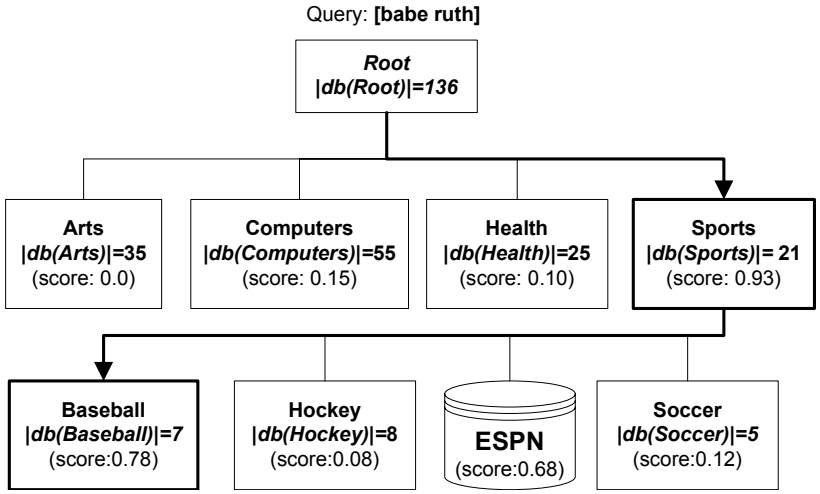

Figure 7: Exploiting a topic hierarchy for database selection.

motivation for our approach, earlier research has indicated that distributed information retrieval systems tend to produce better results when documents are organized in topically cohesive clusters [XC99, LCC00].

Figure 6 specifies our hierarchical database selection algorithm in detail. The algorithm receives as input a query and the target number of databases K that we are willing to search for the query. Also, the algorithm receives the top category C as input, and starts by invoking a flat database selection algorithm to score all subcategories of C for the query (Step 1), using the content summaries associated with the subcategories. We assume in our discussion that the scores produced by the database selection algorithms are greater than or equal to zero, with a zero score indicating that a database or category should be ignored for the query. If at least one "promising" subcategory has a non-zero score (Step 2), then the algorithm picks the best such subcategory C_j (Step 3). If C_j has K or more databases under it (Step 4), the algorithm proceeds recursively under that branch only (Step 5). This strategy privileges "topic-specific" databases over databases with broader scope. On the other hand, if C_j does not have sufficiently many (i.e., K or more) databases (Step 6), then intuitively the algorithm has gone as deep in the hierarchy as possible (exploring only category C_j would result in fewer than K databases being returned). Then, the algorithm returns all $|db(C_j)|$ databases under C_j, plus the best $K - |db(C_j)|$ databases under C but not in C_j, according to the "flat" database selection algorithm of choice (Step 7). If no subcategory of C has a non-zero score (Step 8), again this indicates that the execution has gone as deep in the hierarchy as possible. Therefore, we return the best K databases under C, according to the flat database selection algorithm (Step 9).

Figure 7 shows an example of an execution of this algorithm for query [*babe ruth*] and for a target of $K = 3$ databases. The top-level categories are evaluated by a flat database selection algorithm for the query, and the *"Sports"* category is deemed best, with a score of 0.93. Since the *"Sports"* category has more than three databases, the query is "pushed" into this category. The algorithm proceeds recursively by pushing the

query into the *"Baseball"* category. If we had initially picked $K = 10$ instead, the algorithm would have still picked *"Sports"* as the first category to explore. However, *"Baseball"* has only 7 databases, so the algorithm picks them all, and chooses the best 3 databases under *"Sports"* to reach the target of 10 databases for the query.

In summary, our hierarchical database selection algorithm attempts to choose the most-specific databases for a query. By exploiting the database categorization, this hierarchical algorithm manages to compensate for the necessarily incomplete database content summaries produced by query probing. However, by selecting first the most appropriate categories, this algorithm might miss some relevant databases that are not under the selected categories. One solution would be to try different hierarchy-traversal strategies that could alleviate this problem, and could lead to the selection of databases from multiple branches of the hierarchy. Instead of following this direction of finding the appropriate traversal strategy, we opt for an alternative, "flat" selection scheme: we use the classification hierarchy only for improving the extracted content summaries, and we allow the database selection algorithm to choose among all available databases. Next, we describe this approach in detail.

4.2 Shrinkage-Based Database Selection

As argued above, content summaries built from relatively small document samples are inherently incomplete, which in turn might affect the performance of database selection algorithms that rely on such summaries. Now, we show how we can exploit database category information to improve the quality of the database summaries and subsequently the quality of the database selection decisions. Specifically, Section 4.2.1 presents an overview of our general approach, which builds on the shrinkage ideas from document classification [MRMN98], while Section 4.2.2 explains in detail how we use shrinkage to construct content summaries. Finally, Section 4.2.3 presents a database selection algorithm that uses the shrinkage-based content summaries in an adaptive and query-specific way.

4.2.1 Overview of our Approach

In Sections 2.2 and 3.1, we discussed sampling-based techniques for building content summaries from hidden-Web text databases, and argued that low-frequency words tend to be absent from those summaries. Additionally, other words might be disproportionately represented in the document samples. One way to alleviate these problems is to increase the document sample size. Unfortunately, this solution might be impractical, since it would involve extensive querying of the (remote) databases. Even more importantly, increases in document sample size do not tend to result in comparable improvements in content summary quality [CC01]. An interesting challenge is then to improve the quality of the approximate content summaries without necessarily increasing the document sample size.

This challenge has a counterpart in the problem of hierarchical document classification. Document classifiers rely on training data to associate words with categories. Often, only limited training data is available, which might lead to poor classifiers. Classifier quality can be increased with more training data, but creating large numbers of training examples might be prohibitively expensive. As a less expensive alternative, McCallum et al. [MRMN98] suggested "sharing" training data across related topic categories. Specifically, their *shrinkage* approach compensates for sparse training data for a category by using training examples for more general categories. For example, the training documents for the "*Heart*" category can be augmented with those from the more general *"Health"* category. The intuition behind this approach is that the word distribution in *"Health"* documents is hopefully related to the word distribution in the "*Heart*" documents.

We can apply the same shrinkage principle to our problem, which requires that databases be categorized into a topic hierarchy. This categorization might be an existing one (e.g., if the databases are classified under Open Directory[15] or InvisibleWeb). Alternatively, databases can be classified automatically using the classification algorithm briefly reviewed in Section 2.3. Regardless of how databases are categorized, we can exploit this categorization to improve content summary coverage. The key intuition behind the use of shrinkage in this context is that databases under similar topics tend to have related content summaries.

[15] `http://www.dmoz.org`

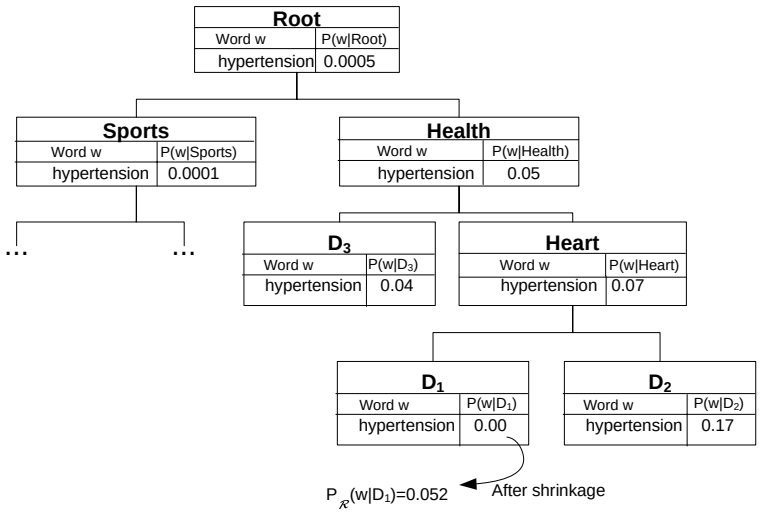

Figure 8: A fraction of a classification hierarchy and content summary statistics for word "hypertension."

Hence, we can use the approximate content summaries for similarly classified databases to complement each other, as illustrated in the following example.

Example 4.2 *Figure 8 shows a fraction of a classification scheme with two text databases D_1 and D_2 classified under "*Heart*," and one text database D_3 classified under the (higher-level) category "*Health*." Assume that the approximate content summary of D_1 does not contain the word "*hypertension*," but that this word appears in many documents in D_1. ("*Hypertension*" might not have appeared in any of the documents sampled to build $\hat{S}(D_1)$.) In contrast, "*hypertension*" appears in a relatively large fraction of D_2 documents as reported in the content summary of D_2, which is also classified under the "*Heart*" category. Then, by "shrinking" $\hat{p}(hypertension|D_1)$ towards the value of $\hat{p}(hypertension|D_2)$, we can capture more closely the actual (and unknown) value of $p(hypertension|D_1)$. The new, "shrunk" value is in effect exploiting documents sampled from both D_1 and D_2.*

We expect databases under the same category to have similar content summaries. Furthermore, even databases classified under relatively general categories can help improve the approximate content summary of a more specific database. Consider database D_3, classified under *"Health"* in the example in Figure 8. $\hat{S}(D_3)$ can help complement the content summary approximation of databases D_1 and D_2, which are classified under a subcategory of "*Health*," namely "*Heart*." Database D_3, however, is a more general database that contains documents in topics other than heart-related. Hence, the influence of $\hat{S}(D_3)$ on $\hat{S}(D_1)$ should perhaps be smaller than that of, say, $\hat{S}(D_2)$. In general, and just as for document classification [MRMN98], each category level might be assigned a different "weight" during shrinkage. We discuss this and other specific aspects of our technique next.

4.2.2 Using Shrinkage over a Topic Hierarchy

We now define more formally how we can use shrinkage for content summary construction. For this, we use the notion of content summaries for the categories of a classification scheme (Definition 4.1) from Section 4.1.

Creating Shrunk Content Summaries Section 4.2.1 argued that mixing information from content summaries of topically related databases may lead to more complete approximate content summaries. We now formally describe how to use shrinkage for this purpose. In essence, we create a new content summary for each database D by "shrinking" the approximate content summary of D, $\hat{S}(D)$, so that it is "closer" to the content summaries $S(C_i)$ of each category C_i under which D is classified.

Definition 4.3 *Consider a database D classified under categories $C_1, \ldots, C_m$ of a hierarchical classification scheme, with $C_i = Parent(C_{i+1})$ for $i = 1, \ldots, m-1$. Let C_0 be a dummy category whose content summary $\hat{S}(C_0)$ contains the same estimate $\hat{p}(w|C_0)$ for every word w. Then, the* shrunk content summary $\widehat{\mathcal{R}}(D)$ *of database D consists of:*

- *An estimate $|\widehat{D}|$ of the number of documents in D, and*
- *For each word w, a shrinkage-based estimate $\hat{p}_{\mathcal{R}}(w|D)$ of $p(w|D)$, defined as:*

$$\hat{p}_{\mathcal{R}}(w|D) = \lambda_{m+1} \cdot \hat{p}(w|D) + \sum_{i=0}^{m} \lambda_i \cdot \hat{p}(w|C_i) \tag{3}$$

for a choice of λ_i values such that $\sum_{i=0}^{m+1} \lambda_i = 1$ (see below).

As described so far, the $\hat{p}(w|C_i)$ values in the $\hat{S}(C_i)$ content summaries are not independent of each other: since $C_i = Parent(C_{i+1})$, all the databases under C_{i+1} are also used to compute $\hat{S}(C_i)$ (Definition 4.1). To avoid this overlap, before estimating $\widehat{\mathcal{R}}(D)$ we subtract from $\hat{S}(C_i)$ all the data used to construct $\hat{S}(C_{i+1})$. Also note that a simple version of Equation 3 is used for database selection based on language models [SJCO02]. Language model database selection "smoothes" the $\hat{p}(w|D)$ probabilities with the probability $\hat{p}(w|G)$ for a "global" category G. Our technique extends this principle and does multilevel smoothing of $\hat{p}(w|D)$, using the hierarchical classification of D. We now describe how to compute the λ_i weights used in Equation 3.

Calculating Category Mixture Weights We define the λ_i mixture weights from Equation 3 so as to make the shrunk content summaries $\widehat{\mathcal{R}}(D)$ for each database D as "similar" as possible to *both* the starting summary $\hat{S}(D)$ and the summary $\hat{S}(C_i)$ of each category C_i under which D is classified. Specifically, we use expectation maximization (EM) [MRMN98] to calculate the λ_i weights, using the algorithm in Figure 9. (This is a simple version of the EM algorithm from [DLR77].)

The "Expectation" step calculates the "likelihood" that content summary $\widehat{\mathcal{R}}(D)$ corresponds to each category. The "Maximization" step weights the λ_i's to maximize the total likelihood across all categories. The result of the algorithm is the shrunk content summary $\widehat{\mathcal{R}}(D)$, which incorporates information from multiple content summaries and is thus hopefully closer to the complete (and unknown) content summary $S(D)$ of database D.

For illustration purposes, Table 2 reports the computed mixture weights for two databases that we used in our experiments. As we can see, in both cases the original database content summary and that of the most specific category for the database receive the highest weights (0.421 and 0.414, respectively, for the AIDS.org database, and 0.411 and 0.297, respectively, for the American Economics Association database). However, higher-level categories also receive non-negligible weights. In general, the λ_{m+1} weight associated with a database (as opposed to with the categories under which it is classified) is usually highest among the λ_i's and so, the word-distribution statistics for the database are not "eclipsed" by the category statistics. (We verify this claim experimentally in Section 6.3.)

Shrinkage might in some cases (incorrectly) reduce the estimated frequency of words that distinctly appear in a database. Fortunately, this reduction tends to be small, because of the relatively high value of λ_{m+1}, and hence these distinctive words remain with high frequency estimates. As an example, consider the *AIDS.org* database from Table 2. The word *chlamydia* appears in 3.5% of the documents in the *AIDS.org* database. This word appears in 4% of the documents in the document sample from *AIDS.org* and in approximately 2% of the documents in the content summary for the *AIDS* category. After applying shrinkage, the estimated frequency of the word *chlamydia* is somewhat reduced but is still high. The shrinkage-based estimate is that *chlamydia* appears in 2.85% of the documents in *AIDS.org*, which is still close to the real frequency.

Shrinkage might in some cases (incorrectly) cause the inclusion of words in the content summary that do not appear in the corresponding database. Fortunately, such spurious words tend to be introduced in the summaries with low weight. Using once again the *AIDS.org* database as an example, we observed that the word *metastasis* was (incorrectly) added by the shrinkage process to the summary: *metastasis* does not appear in the database, but is included in documents in other databases under the *Health* category and hence

Initialization step:
Set each λ_i to any normalized, non-zero value such that $\sum_{i=0}^{m+1} \lambda_i = 1$. (For example, $\lambda_i = \frac{1}{m+2}$, for every $i = 0, \ldots, m+1$.)
Create the shrunk content summary $\widehat{\mathcal{R}}(D)$, using the current λ_i values (Definition 4.3).

Expectation step:
Calculate the "similarity" β_i of each category C_i with $\widehat{\mathcal{R}}(D)$, for $i = 0, \ldots, m$:

$$\beta_i = \sum_{w \in \hat{S}(D)} \frac{\lambda_i \cdot \hat{p}(w|C_i)}{\hat{p}_{\mathcal{R}}(w|D)}$$

Calculate the "similarity" β_{m+1} of database D with $\widehat{\mathcal{R}}(D)$:

$$\beta_{m+1} = \sum_{w \in \hat{S}(D)} \frac{\lambda_{m+1} \cdot \hat{p}(w|D)}{\hat{p}_{\mathcal{R}}(w|D)}$$

Maximization step:
Maximize the total "similarity" of $\widehat{\mathcal{R}}(D)$ with the category content summaries $\hat{S}(C_i)$ and with $\widehat{S}(D)$, by redefining each λ_i as follows:

$$\lambda_i = \frac{\beta_i}{\sum_{j=0}^{m+1} \beta_j}$$

Modify the shrunk content summary $\widehat{\mathcal{R}}(D)$ using the current λ_i values.

Termination check:
When calculation of the λ_i weights converges (within a small ϵ) return the "current" shrunk content summary $\widehat{\mathcal{R}}(D)$. Otherwise, go to the "Expectation" step.

Figure 9: Using expectation maximization to determine the λ_i mixture weights for the shrunk content summary of a database D.

Database	*Category*	λ	*Database*	*Category*	λ
AIDS.org	Uniform	0.075	American Economics Association	Uniform	0.041
	Root	0.026		Root	0.041
	Health	0.061		Science	0.055
	Diseases	0.003		Social Sciences	0.155
	AIDS	0.414		Economics	0.297
	AIDS.org	0.421		**A.E.A.**	0.411

Table 2: The category mixture weights for two databases.

the word is in the *Health* category content summary. The shrunk content summary for *AIDS.org* estimates that *metastasis* appears in just 0.03% of the database documents, so such a low estimate is unlikely to adversely affect database selection decisions. (We will evaluate the positive and negative effects of shrinkage experimentally later in Sections 6 and 7.)

Finally, note that the λ_i weights are computed off-line for each database when the sampling-based database content summaries are created. This computation does not involve any overhead at query-processing time.

4.2.3 Improving Database Selection using Shrinkage

So far, we introduced a shrinkage-based strategy to complement the incomplete content summary of a database with the summaries of topically related databases. In principle, existing database selection algorithms could proceed without modification and use the shrunk summaries to assign scores for *all* queries and databases. However, sometimes shrinkage might not be beneficial and should not be used. Intuitively, shrinkage should be used to determine the score $s(q, D)$ for a query q and a database D only if the "uncertainty" associated with this score would otherwise be large.

The uncertainty associated with an $s(q, D)$ score depends on a number of sample-, database-, and query-related factors. An important such factor is the size of the document sample relative to that of database

D. If an approximate summary $\hat{S}(D)$ was derived from a sample that included most of the documents in D, then this summary is already "sufficiently complete." (For example, this situation might arise if D is a small database.) In this case, shrinkage is not necessary and might actually be undesirable, since it might introduce spurious words into the content summary from topically related (but not identical) databases. Another factor is the frequency of the query words in the sample used to determine $\hat{S}(D)$. If, say, every word in a query appears in close to all the sample documents, and the sample is representative of the entire database contents, then there is little uncertainty on the distribution of the words over the database at large. Therefore, the uncertainty about the score assigned to the database from the database selection algorithm is also low, and there is no need to apply shrinkage. Analogously, if every query word appears in only a small fraction of the sample documents, then most probably the database selection algorithm would assign a low score to the database, since it is unlikely for the database to be a good candidate for evaluating the query. Again, in this case shrinkage would provide limited benefit and should be avoided. However, consider the following scenario, involving *bGlOSS* and a multi-word query for which most words appear very frequently in the sample, but where one query word is missing from the document sample altogether. In this case, *bGlOSS* would assign a zero score to the database. The missing word, though, may have a non-zero frequency in the complete content summary, and the score assigned by *bGlOSS* to the database would have been significantly higher in the presence of this knowledge because of *bGlOSS*'s boolean nature. So, the *uncertainty about the database score* that *bGlOSS* would assign if given the complete summary is high, and it is then desirable to apply shrinkage. In general, for query-word distribution scenarios where the approximate content summary is not sufficient to reliably establish the query-specific score for a database, shrinkage should be used.

More formally, consider a query $q = [w_1, \ldots, w_n]$ with n words $w_1, \ldots, w_n$, a database D, and an approximate content summary for D, $\hat{S}(D)$, derived from a random sample S of D. Furthermore, suppose that word w_k appears in exactly s_k documents in the sample S. For every possible combination of values $d_1, \ldots, d_n$ (see below), we compute:

- The probability P that w_k appears in exactly d_k documents in D, for $k = 1, \ldots, n$:

$$P = \prod_{k=1}^{n} \frac{d_k^{\gamma} \left(\frac{d_k}{|D|}\right)^{s_k} \left(1 - \frac{d_k}{|D|}\right)^{|S|-s_k}}{\sum_{i=0}^{|D|} i^{\gamma} \cdot \left(\frac{i}{|D|}\right)^{s_k} \left(1 - \frac{i}{|D|}\right)^{|S|-s_k}} \tag{4}$$

  where γ is a database-specific constant. (For details, see Appendix A.)

- The score $s(q, D)$ that the database selection algorithm of choice would assign to D if $p(w_k|D) = \frac{d_k}{|D|}$, for $k = 1, \ldots, n$.

So for each possible combination of values $d_1, \ldots, d_n$, we compute both the probability of the value combination and the score that the database selection algorithm would assign to D for this document frequency combination. Then, we can approximate the "uncertainty" behind the $s(q, D)$ score by examining the mean and variance of the database scores over the different $d_1, \ldots, d_n$ values. This computation can be performed efficiently for a generic database selection algorithm: given the sample frequencies $s_1, \ldots, s_n$, a large number of possible $d_1, \ldots, d_n$ values have virtually zero probability of occurring, so we can ignore them. Additionally, mean and variance converge fast, even after examining only a small number of $d_1, \ldots, d_n$ combinations. Specifically, we examine random $d_1, \ldots, d_n$ combinations and periodically calculate the mean and variance of the score distribution. Usually, after examining just a few hundred random $d_1, \ldots, d_n$ combinations, mean and variance converge to a stable value. The mean and variance computation typically requires less than 0.1 seconds for a single-word query, and approximately 4-5 seconds for a 16-word query.[16] This computation can be even faster for a large class of database selection algorithms that assume independence between the query words (e.g., [GGMT99, CLC95, XC99]). For these algorithms, we can calculate the mean and variance for each query word separately, and then combine them into the final mean score and variance, respectively. (In Appendix B we provide more details.) For algorithms that assume independence between the query words, the computation time is typically less than 0.1 seconds.

Figure 10 summarizes the discussion above, and shows how we can adaptively use shrinkage with an existing database selection algorithm. Specifically, the algorithm takes as input a query q and a set of

[16] We measured the time on a PC with a dual AMD Athlon CPU, running at 1.8 GHz.

Input: Query $q = [w_1, \ldots, w_n]$; databases $D_1, \ldots, D_m$

Content Summary Selection step:
For each D_i:
  For every possible choice of values for $d_1, \ldots, d_n$ (see text):
    Compute the probability P that w_k appears in
    exactly d_k documents in D_i, for $k = 1, \ldots, n$.

    Compute the score $s(q, D_i)$ assuming that w_k appears
    in exactly d_k documents in D_i, for $k = 1, \ldots, n$.

  If the standard deviation of the score distribution across d_k values is larger than its mean
  then $A(D_i) = \widehat{\mathcal{R}}(D_i)$ // use "shrunk" content summary
  else $A(D_i) = \hat{S}(D_i)$ // use "unshrunk" content summary

Scoring step:
For each D_i:
  Compute $s(q, D_i)$ using the $A(D_i)$ content summary,
  as selected in the "Content Summary Selection" step.

Ranking step:
Rank the databases by decreasing score $s(q, D_i)$.

Figure 10: Using shrinkage adaptively for database selection.

URL	*Documents*	*Classification*
`http://www.bartleby.com/`	375,734	Root→ Arts→ Literature→ Texts
`http://java.sun.com/`	78,870	Root→ Computers→ Programming→ Java
`http://mathforum.org/`	29,602	Root→ Science→ Mathematics
`http://www.uefa.com/`	28,329	Root→ Sports→ Soccer

Table 3: Some of the real Web databases in the *Web* data set.

databases $D_1, \ldots, D_m$. The "Content Summary Selection" step decides whether to use shrinkage for each database D_i, as discussed above. If the distribution of possible scores has high variance, then $\hat{S}(D_i)$ is considered unreliable and the shrunk content summary $\widehat{\mathcal{R}}(D_i)$ is used instead. Otherwise, shrinkage is not applied. Then, the "Scoring" step computes the score $s(q, D_i)$ for each database D_i, using the content summary chosen for D_i in the "Content Summary Selection" step. Finally, the "Ranking" step orders all databases by their final score for the query. The metasearcher then uses this rank to decide what databases to search for the query.

In this section, we presented two database selection strategies that exploit database classification to improve the selection decisions in presence of incomplete content summaries. Next, we present the settings for the experimental evaluation of the content summary construction algorithm of Section 3 and of the database selection algorithms of Section 4.

5 Experimental Setting

In this section, we describe the data (Section 5.1), strategies for computing content summaries (Section 5.2), and database selection algorithms (Section 5.3) that we use for the experiments reported in Sections 6 and 7.

5.1 Data Sets

The content summary construction techniques that we proposed above rely on a hierarchical categorization scheme. For our experiments, we use the classification scheme from [GIS03], with 72 nodes organized in a 4-level hierarchy. To evaluate the algorithms described in this article, we use four data sets in conjunction with the hierarchical classification scheme.

- **Controlled:** This is a data set that was also used for evaluating the task of database classification

in [GIS03]. To construct this data set, we used postings from Usenet newsgroups where the signal-to-noise ratio was high and where the documents belonged (roughly) to one of the categories of our classification scheme. For example, the newsgroups `comp.lang.c` and `comp.lang.c++` were considered relevant to category "C/C++." We collected 500,000 articles from April through May 2000. Out of the 500,000 articles, 81,000 articles were used to train and test the document classifiers that we used for the *Focused Probing* algorithm (see Section 5.2.1). We removed all headers from the newsgroup articles, with the exception of the "Subject" line; we also removed the e-mail addresses contained in the articles. Except for these modifications, we made no other changes to the collected documents.

We used the remaining 419,000 articles to build the 500 databases in the *"Controlled"* data set. The size of the 500 *Controlled* databases that we created ranged from 25 to 25,000 documents. Out of the 500 databases, 350 are "homogeneous," with documents from a single category, while the remaining 150 are "heterogeneous," with a variety of category mixes. We define a database as "homogeneous" when it has articles from only one node, regardless of whether this node is a leaf node or not. If it is not a leaf node, then it has equal number of articles from each leaf node in its subtree. The "heterogeneous" databases, on the other hand, have documents from different categories that reside in the same level in the hierarchy (not necessarily siblings), with different mixture percentages. We believe that these databases model real-world searchable Web databases, with a variety of sizes and foci.

- **TREC4**: This is a set of 100 databases created using documents from TREC-4 [Har96] and separated into disjoint databases via clustering using the K-means algorithm, as specified in [XC99]. By construction, the documents in each database are on roughly the same topic.
- **TREC6**: This is a set of 100 databases created using documents from TREC-6 [VH98] and separated into disjoint databases using the same methodology as for *TREC4*.
- **Web**: This set contains the top-5 *real Web databases* from each of the 54 leaf categories of the hierarchy and from each of the 17 internal nodes of the hierarchy[17] (except for the root), as ranked in the Google Directory,[18] for a total of 315 databases.[19] The sizes of these databases range from 100 to about 376,000 documents. Table 3 lists four example databases. We used the *GNU Foundation's wget* crawler to download the HTML contents of each site, and we kept only the text from each file by stripping the HTML tags using the "*lynx –dump*" command.

We use the *Controlled* data set in Section 6 to extensively test the quality of the generated content summaries and to pick the variation of our probing strategy from Section 3.1 that we will use for our subsequent experiments in Section 7. We also use the *Web* data set in Section 6 to further validate the results on the quality of the summaries. Finally, we use the *TREC4* and *TREC6* data sets both for examining the quality of the content summaries and for testing the performance of the database selection algorithms in Section 7. (The *TREC4* and *TREC6* data sets are the only ones that include queries and associated relevance judgments.) For indexing and searching the files in all data sets, we used *Jakarta Lucene*,[20] an open-source full-text search engine.

5.2 Content Summary Construction Algorithms

Our experiments evaluate a number of content-summary construction techniques, which vary in their underlying document sampling algorithms (Section 5.2.1) and on whether they use shrinkage and absolute frequency estimation (Section 5.2.2).

5.2.1 Sampling Algorithms

We use different sampling algorithms for retrieving the documents based on which we build the approximate content summaries $\hat{S}(D)$ of each database D:

[17] Instead of retrieving the top-5 databases from each category, a plausible alternative is to select a number of databases from each hierarchy node that is proportional to the size of the respective hierarchy subtree. In our work, we give equal "weight" to each category.

[18] `http://directory.google.com/`

[19] We have fewer than $71 \times 5 = 355$ databases, because not all internal nodes included at least five databases.

[20] `http://lucene.apache.org/`

- **Query-Based Sampling (QBS)**: We experimented with the two versions of *QBS* described in Section 2, namely *QBS-Ord* and *QBS-Lrd*. As the initial dictionary D for these two methods, we used all words in the *Controlled* databases.[21] Each query retrieves up to four previously unseen documents. Sampling stops after retrieving 300 distinct documents. In our experiments, sampling also stops when 500 consecutive queries retrieve no new documents. To minimize the effect of randomness, we run each experiment over five *QBS* document samples for each database and report average results.

- **Focused Probing (FPS)**: We evaluate our *Focused Probing* technique, which we introduced in Section 3.1, with a variety of underlying document classifiers. The document classifiers are used by *Focused Probing* to generate the queries sent to the databases. Specifically, we consider the following variations of the *Focused Probing* technique:

  - **FP-RIPPER**: *Focused Probing* using RIPPER [Coh96] as the base document classifier.
  - **FP-C4.5**: *Focused Probing* using C4.5RULES, which extracts classification rules from decision tree classifiers generated by C4.5 [Qui92].
  - **FP-Bayes**: *Focused Probing* using Naive-Bayes classifiers [DHS00] in conjunction with the technique to extract rules from numerically-based Naive-Bayes classifiers from [GIS03].
  - **FP-SVM**: *Focused Probing* using Support Vector Machines with linear kernels [Joa98] in conjunction with the same rule-extraction technique used for *FP-Bayes*.

  The query probes of these classifiers are typically short: the median query length is one word, the average query length is 1.35 words, and the maximum query length is four words. Further details about the characteristics of the classifiers are available in [GIS03].

  We also consider different values for the τ_{es} and τ_{ec} thresholds, which affect the granularity of sampling performed by the algorithm (see Section 3.1). All variations were tested with threshold τ_{es} ranging between 0 and 1. Low values of τ_{es} result in databases being "pushed" to more categories, which in turn results in larger document samples. To keep the number of experiments manageable, we fix the coverage threshold to $\tau_{ec} = 10$, varying only the specificity threshold τ_{es}.

5.2.2 Shrinkage and Frequency Estimation

Our experiments also evaluate the usefulness of our shrinkage (Section 4.2) and frequency estimation techniques (Section 3.2). To evaluate the effect of shrinkage on content summary quality, we create the shrunk content summary $\widehat{\mathcal{R}}(D)$ for each database D and contrast its quality against that of the unshrunk content summary $\hat{S}(D)$. Similarly, to evaluate the effect of our frequency estimation technique on content summary quality, we consider the *QBS* and *FPS* summaries both with and without this frequency estimation. We report results on the quality of the content summaries before and after the application of our shrinkage algorithm.

To apply shrinkage, we need to classify each database into the 72-node topic hierarchy. Unfortunately, such classification is not available for TREC data, so for the *TREC4* and *TREC6* data sets we resort to our classification technique from [GIS03], which we reviewed briefly in Section 2.3.[22] A manual inspection of the classification results confirmed that they are generally accurate. For example, the *TREC4* database *all-83*, with articles about AIDS, was correctly classified under the "Root→ Health→ Diseases→ AIDS" category. Interestingly, in the case in which databases were not classified correctly, "similar" databases were still classified into the same (incorrect) category. For example, *all-14*, *all-21*, and *all-44* are about middle-eastern politics and were classified under the "Root→ Science→ Social Sciences→ History" category.

Unlike for *TREC4* and *TREC6*, for which no "external" classification of the databases is available, for the *Web* databases we do not have to rely on query probing for classification. Instead we can use the categories assigned to the databases in the Google Directory. For *QBS*, the classification of each database in our data set was indeed derived from the Google Directory. For *FPS*, we can either use the (correct) Google Directory

[21] Note that this slightly favors *QBS* in the experiments over the *Controlled* databases: the initial dictionary contains a superset of the words that appear in each database in the *Controlled* data set. The experiments that use the *Web*, *TREC4*, and *TREC6* data sets are not affected by this bias.

[22] We adapted the technique slightly so that each database is classified under exactly one category.

database classification, as for *QBS*, or rely on the automatically computed database classification that this technique derives during document sampling. We tried both choices and found only small differences in the experimental results. Therefore, for conciseness, we only report the *FPS* results for the automatically derived database classification. Finally, for the *Controlled* data set, we use the automatically derived classification with $\tau_{es} = 0.25$ and $\tau_{ec} = 10$.

5.3 Database Selection Algorithms

The algorithms presented in this article (Sections 4.1 and 4.2.3) are built on top of underlying "base" database selection algorithms. We consider three well-known such algorithms from the literature.

- **bGlOSS**, as described in [GGMT99]. Databases are ranked for a query q by decreasing score $s(q, D) = |D| \cdot \prod_{w \in q} \hat{p}(w|D)$.
- **CORI**, as described in [FPC+99]. Databases are ranked for a query q by decreasing score $s(q, D) = \sum_{w \in q} \frac{0.4+0.6 \cdot T \cdot I}{|q|}$, where $T = (\hat{p}(w|D) \cdot |D|)/(\hat{p}(w|D) \cdot |D| + 50 + 150 \cdot \frac{cw(D)}{mcw})$, $I = \log\left(\frac{m+0.5}{cf(w)}\right) / \log(m + 1.0)$, $cf(w)$ is the number of databases containing w, m is the number of databases being ranked, $cw(D)$ is the number of words in D, and mcw is the mean cw among the databases being ranked. One potential problem with the use of CORI in conjunction with shrinkage is that virtually every word has $cf(w)$ equal to the number of databases in the data set: every word appears with non-zero probability in every shrunk content summary. Therefore, when we calculate $cf(w)$ for a word w in our CORI experiments, we consider w as "present" in a database D only when $round(|D| \cdot \hat{p}_{\mathcal{R}}(w|D)) \geq 1$.
- **Language Modeling (LM)**, as described in [SJCO02]. Databases are ranked for a query q by decreasing score $s(q, D) = \prod_{w \in q}(\lambda \cdot \hat{p}(w|D) + (1 - \lambda) \cdot \hat{p}(w|G))$. The LM algorithm is equivalent to the KL-based database selection method described in [XC99]. For LM, $p(w|D)$ is defined differently than in Definition 2.1, Section 3.1. Specifically, $p(w|D) = \frac{tf(w,D)}{\sum_i tf(w_i,D)}$, where $tf(w, D)$ is the total number of occurrences of w in D. The algorithms described in Section 4.2 can be easily adapted to reflect this difference, by substituting this definition of $p(w|D)$ for that in Definition 2.1. LM smoothes the $\hat{p}(w|D)$ probability with the probability $\hat{p}(w|G)$ for a "global" category G. In our experiments, we derive the probabilities $\hat{p}(w|G)$ from the "Root" category summary and we use $\lambda = 0.5$ as suggested in [SJCO02].

We experimentally evaluate the three database selection algorithms above with three variations:

- **Plain**: Using "unshrunk" (incomplete) database content summaries extracted via *QBS* or *FPS*.
- **Shrinkage**: Using shrinkage when appropriate (as discussed in Section 4.2.3), again over database content summaries extracted via *QBS* or *FPS*.
- **Hierarchical**: Using "unshrunk" database content summaries (extracted via *QBS* or *FPS*) in conjunction with the hierarchical database selection algorithm from Section 4.1.

Finally, to evaluate the effect of our frequency estimation technique (Section 3.2) on database selection accuracy, we consider the *QBS* and *FPS* summaries both with and without this frequency estimation. Also, since stemming can help alleviate the data sparseness problem, we consider content summaries both with and without stemming.

6 Experimental Results for Content Summary Quality

In this section, we evaluate the alternate content summary construction techniques. We first focus on the impact of the choice of sampling algorithm on content summary quality in Section 6.1. Then, in Section 6.2 we show that databases classified under similar categories tend to have similar content summaries. Finally, in Section 6.3 we show that shrinkage-based content summaries are of higher quality than their unshrunk counterparts.

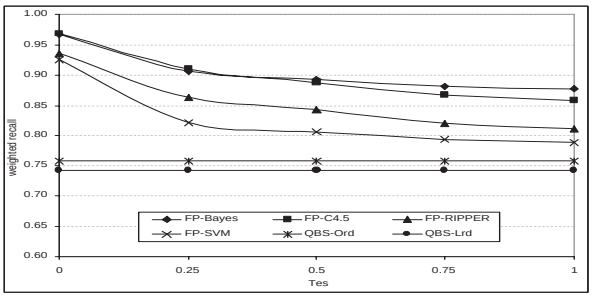

Figure 11a: *Weighted recall* as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

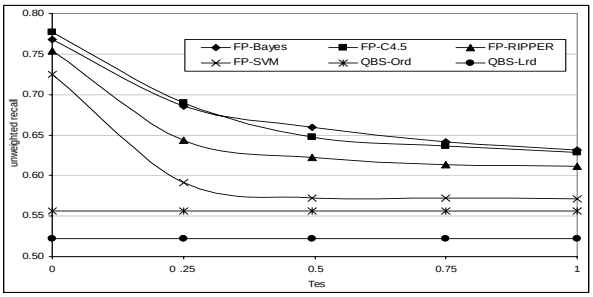

Figure 11b: *Unweighted recall* as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

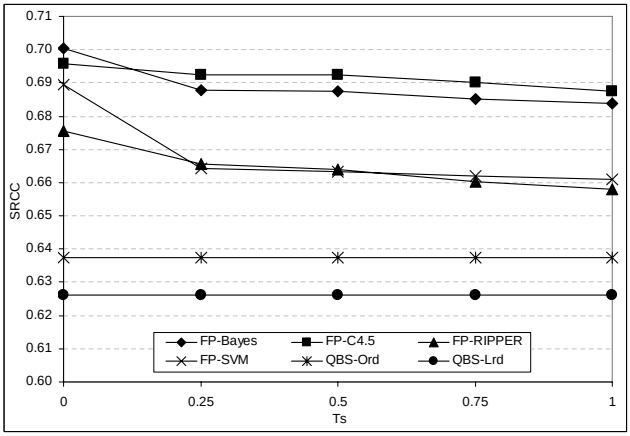

Figure 11c: *Spearman Rank Correlation Coefficient* as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

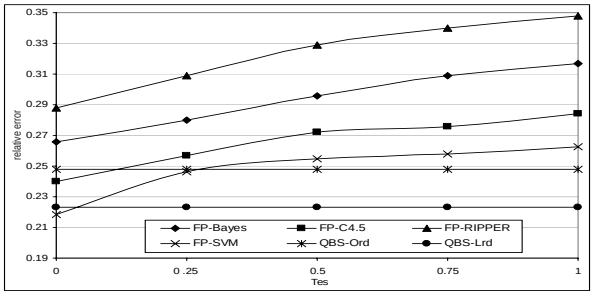

Figure 11d: Relative error of the *df* estimations, for words with $df > 3$, as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

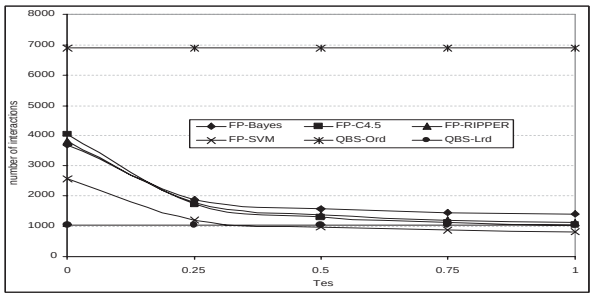

Figure 11e: Number of interactions per database, as a function of the specificity threshold τ_{es} and for the *Controlled* data set.

6.1 Effect of Sampling Algorithm

Consider a database D and a content summary $A(D)$ computed using an arbitrary sampling technique. We now evaluate the quality of $A(D)$ in terms of how well it approximates the "perfect" content summary $S(D)$, determined by examining every document in D. In the following definitions, W_A is the set of words that appear in $A(D)$, while W_S is the (complete) set of words that appear in $S(D)$. Our experiments are over the *Controlled* data set.

Recall: An important property of content summaries is their coverage of the actual database vocabulary. The *weighted recall (wr)* of $A(D)$ with respect to $S(D)$ is defined as $wr = \frac{\sum_{w \in W_A \cap W_S} df(w)}{\sum_{w \in W_S} df(w)}$, which corresponds to the *ctf ratio* in [CC01]. This metric gives higher weight to more frequent words, but is calculated *after* stopwords (e.g., "a", "the") are removed, so this ratio is not artificially inflated by the discovery of common words. We report the *weighted recall* for the different content summary construction algorithms in Figure 11a. The variants of the *Focused Probing* technique achieve substantially higher *wr* values than *QBS-Ord* and *QBS-Lrd* do. Early during probing, *Focused Probing* retrieves documents covering different topics, and then sends queries of increasing specificity, retrieving documents with more specialized words. As expected, the coverage of the *Focused Probing* summaries increases for lower values of the specificity threshold τ_{es}, since the number of documents retrieved for lower thresholds is larger (e.g., 493 documents for *FP-SVM* with $\tau_{es} = 0.25$ vs. 300 documents for *QBS-Lrd*): a sample of larger size, everything else being the same, is better for content summary construction. In general, the difference in weighted recall between *QBS-Lrd* and *QBS-Ord* is small, but *QBS-Lrd* has slightly lower *wr* values due to the bias induced from querying only using previously discovered words. To understand whether low-frequency words are present in the approximate summaries, we resort to the *unweighted recall (ur)* metric, defined as $ur = \frac{|W_A \cap W_S|}{|W_S|}$. The *ur* metric is the fraction of words in a database that are present in a content summary. Figure 11b shows trends similar to the ones for weighted recall, but the numbers are smaller, showing that lower frequency words are not well represented in the approximate summaries.

Correlation of Word Rankings: The recall metric can be helpful to compare the quality of different content summaries. However, this metric alone is not enough, since it does not capture the relative "rank" of words in the content summary by their observed frequency. To measure how well a content summary orders words by frequency with respect to the actual word frequency order in the database, we use the *Spearman Rank Correlation Coefficient* (*SRCC* for short), which is also used in [CC01] to evaluate the quality of the content summaries. (We use the version of *SRCC* that accounts for ties, as suggested by Callan and Connell [CC01].) When two rankings are identical, then *SRCC*=1; when they are uncorrelated, *SRCC*=0; and when they are in reverse order, *SRCC*=-1. The results for the different algorithms are shown in Figure 11c. Again, the content summaries produced by the *Focused Probing* techniques have higher *SRCC* values than those for *QBS-Lrd* and *QBS-Ord*, hinting that *Focused Probing* retrieves a more "representative" sample of documents from the database.

Accuracy of Frequency Estimations: In Section 3.2, we introduced a technique to estimate the actual absolute frequencies of the words in a database. To evaluate the accuracy of our predictions, we report the average relative error for words with actual frequencies greater than three. (Including the large tail of less-frequent words would highly distort the relative-error computation, even for small estimation errors.) Figure 11d reports the average relative error estimates for our algorithms. We also applied our absolute frequency estimation algorithm of Section 3.2 to *QBS-Ord* and *QBS-Lrd*, even though this estimation is not part of the original algorithms in [CC01]. As a general conclusion, our technique provides a good ballpark estimate of the absolute frequency of the words.

Efficiency: To measure the efficiency of the probing methods, we report the sum of the number of queries sent to a database and the number of documents retrieved ("number of interactions") in Figure 11e. (See [GIS03] for a justification of this metric.) The *Focused Probing* techniques retrieve –on average– one document per query, while *QBS-Lrd* retrieves about one document for every two queries.[23] *QBS-Ord* unnecessarily issues many queries that produce no document matches. The efficiency of the other techniques is correlated with their effectiveness. The more expensive techniques tend to give better results. The exception is *FP-SVM*, which for $\tau_{es} > 0$ has the lowest cost (or cost close to the lowest one) and gives results of

[23] The average is computed over the databases in the *Controlled* data set, after the different content summary construction algorithms run to completion.

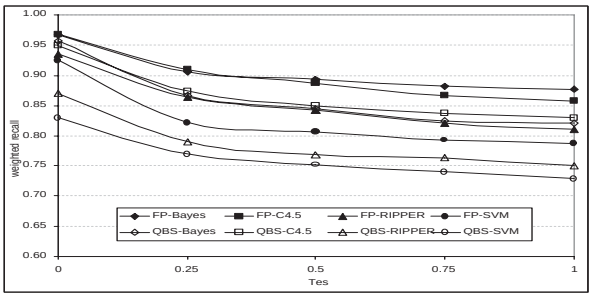

Figure 12a: *Weighted recall* as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

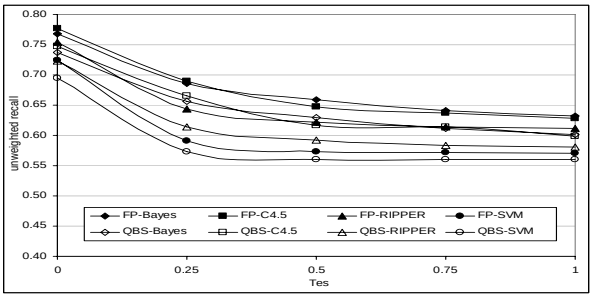

Figure 12b: *Unweighted recall* as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

comparable quality with respect to the more expensive methods. As discussed above, the *Focused Probing* probes were generally short, with a maximum of four words and a median of one word per query.

Recall, Rank Correlation, and Efficiency for Identical Sample Size: We have seen that the *Focused Probing* algorithms achieve better *wr* and *SRCC* values than the *QBS-Lrd* and *QBS-Ord* algorithms do. However, the *Focused Probing* algorithms generally retrieve a (moderately) larger number of documents than *QBS-Ord* and *QBS-Lrd* do, and the number of documents retrieved depends on how deeply into the categorization scheme the databases are classified. To test whether the improved performance of *Focused Probing* is just a result of the larger sample size, we increased the sample size for *QBS-Lrd* to retrieve the same number of documents as each *Focused Probing* variant.[24] We refer to the versions of *QBS-Lrd* that retrieve the same number of documents as *FP-Bayes*, *FP-C4.5*, *FP-RIPPER*, and *FP-SVM* as *QBS-Bayes*, *QBS-C4.5*, *QBS-RIPPER*, and *QBS-SVM*, respectively.

The *wr*, *ur*, and *SRCC* values for the alternative versions of *QBS-Lrd* are shown in Figures 12a, 12b, and 12c, respectively. We observe that the *wr*, *ur*, and *SRCC* values of the *QBS* methods improve with the larger document sample, but are still lower than their *Focused Probing* counterparts; the difference is statistically significant at the 1% level according to a paired t-test. In general, the results show that the *Focused Probing* methods are more effective than their *QBS* counterparts: the *Focused Probing* queries are generated by document classifiers and tend to "cover" distinct parts of the document space. In contrast, the *QBS* methods query the database with words that appear in the retrieved documents, and these documents tend to contain words already present in the sample. This difference is more pronounced in the earlier stages of sampling, where *Focused Probing* sends more general queries. When *Focused Probing* starts sending queries for lower levels of the classification hierarchy, both *Focused Probing* and *QBS* demonstrate similar rates of vocabulary growth. The exact point where the two techniques start performing similarly depends on the size of the database. For large databases, *Focused Probing* dominates *QBS* even at deep levels of the hierarchy, while for smaller databases the benefits of *Focused Probing* are only visible during the first and second levels of sampling.

Finally, we measured the number of interactions performed by the *Focused Probing* and *QBS* methods when they retrieve the same number of documents. The sum of the number of queries sent to a database and the number of documents retrieved ("number of interactions") is shown in Figure 12d. The average number of queries sent to each database is larger for the *QBS* methods than for their *Focused Probing* counterparts when they retrieve the same number of documents: the *QBS* queries are derived from the already acquired vocabulary, and many of these words appear only in one or two documents, so a large fraction of the *QBS* queries return only documents that have been retrieved before. These queries increase the number of interactions for *QBS*, but do not retrieve any new documents.

For completeness, we ran the same set of experiments for the *Web*, *TREC4*, and *TREC6* data sets. We use content summaries extracted from *FP-SVM* with specificity threshold $\tau_{es} = 0.25$ and coverage threshold $\tau_{ec} = 10$: *FP-SVM* exhibits the best accuracy-efficiency tradeoff while $\tau_{es} = 0.25$ leads to good

[24] We pick *QBS-Lrd* over *QBS-Ord* because *QBS-Ord* requires a much larger number of queries to extract its document sample: most of its queries return no results (see Figure 11e), making it the most expensive method.

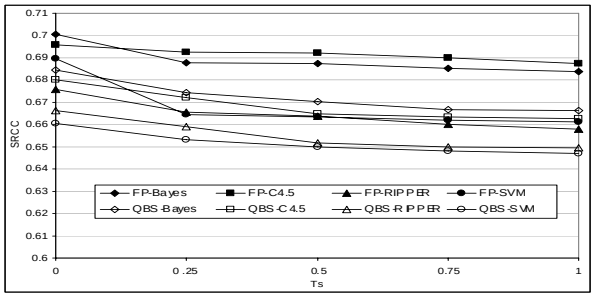

Figure 12c: *Spearman Rank Correlation Coefficient* as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

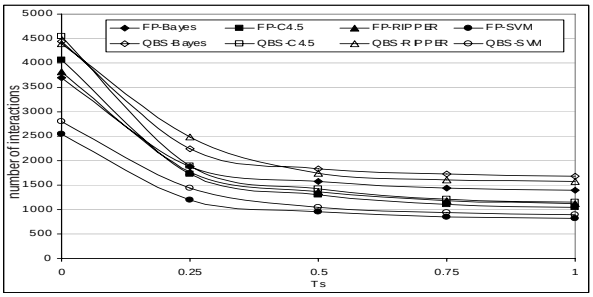

Figure 12d: Number of interactions per database as a function of the specificity threshold τ_{es}, for the *Controlled* data set and for the case where the *FP* and *QBS* methods retrieve the same number of documents.

Data Set	*Method*	*Metric*			
		wr	*ur*	*SRCC*	*Interactions*
Web	*FP-SVM*	0.887	0.520	0.813	623
Web	*QBS-SVM*	0.879	0.456	0.810	650
TREC4	*FP-SVM*	0.972	0.599	0.884	650
TREC4	*QBS-SVM*	0.943	0.428	0.850	702
TREC6	*FP-SVM*	0.975	0.662	0.905	684
TREC6	*QBS-SVM*	0.952	0.545	0.883	694

Table 4: *Weighted recall*, *unweighted recall*, *Spearman Rank Correlation Coefficient*, and number of interactions per database, for the *Web*, *TREC4*, and *TREC6* data sets and for the case where *FP-SVM* and *QBS-SVM* retrieve the same number of documents.

database classification decisions as well (see [GIS03]). We also use the respective *QBS-SVM* version of *QBS*. The results that we obtained (Table 4) were in general similar to those for the *Controlled* data set. The main difference with the results obtained for the *Controlled* data set is that the number of interactions is substantially lower for *FP-SVM* (and hence for *QBS-SVM*): the databases in the *Controlled* data set are typically classified under multiple categories; in contrast, the databases in *Web*, *TREC4*, and *TREC6* are generally classified under only one or two categories, and hence require much fewer queries for content summary construction than the *Controlled* databases do.

Evaluation Conclusions: Overall, the *Focused Probing* techniques produce summaries of better quality than *QBS-Ord* and *QBS-Lrd* do, both in terms of vocabulary coverage and word-ranking preservation. The cost of *Focused Probing* in terms of number of interactions with the databases is comparable to that for *QBS-Lrd* (for $\tau_{es} > 0$), and significantly lower than that for *QBS-Ord*. Finally, the absolute frequency estimation technique of Section 3.2 gives good ballpark approximations of the actual frequencies.

6.2 Relationship between Content Summaries and Categories

A key conjecture behind our database selection algorithms is that databases under the same category tend to have closely related content summaries. Thus, we can use the content summary of a database to complement the (incomplete) content summary of another database in the same category (Section 4.2). We now explore this conjecture experimentally using the *Controlled*, *Web*, *TREC4*, and *TREC6* data sets.

Each database in the *Controlled* set is classified using $\tau_s = 0.25$ and $\tau_c = 10$, and following the definition in [GIS03]. By construction, we know the contents of all the databases in the *Controlled* set, as well as their correct classification. For the *Web* data set, we use the database classification as given by Open Directory. Finally, we classify the databases in the *TREC4* and *TREC6* data sets using the classification algorithm from [GIS03], with $\tau_{es} = 0.25$ and $\tau_{ec} = 10$. Then, for each pair of databases D_i and D_j we measure:

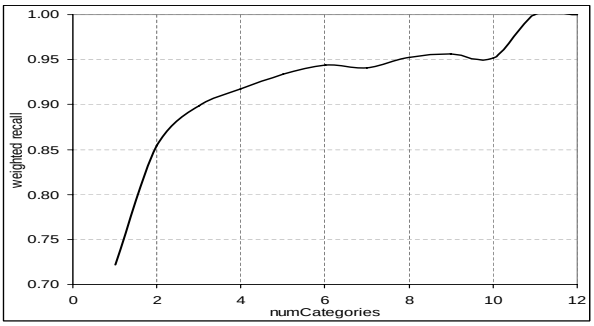

Figure 13a: *Weighted recall* for pairs of database content summaries, for the *Controlled* data set as a function of the number of common categories in the database pairs.

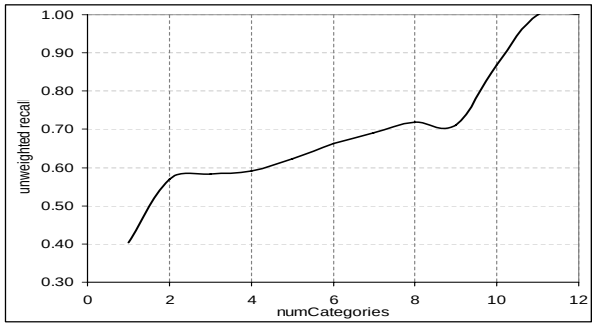

Figure 13b: *Unweighted recall* for pairs of database content summaries, for the *Controlled* data set as a function of the number of common categories in the database pairs.

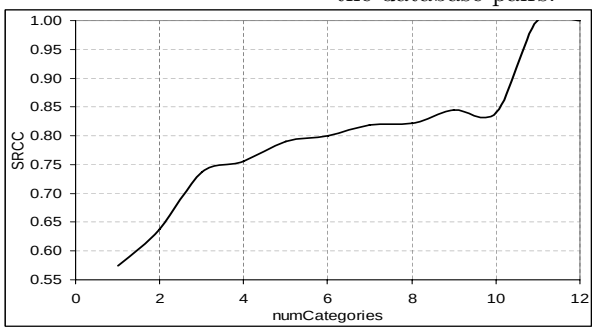

Figure 13c: *Spearman Rank Correlation Coefficient* for pairs of database content summaries, for the *Controlled* data set as a function of the number of common categories in the database pairs.

Data Set	*numCategories*			
	1	2	3	4
Web	0.83	0.89	0.90	0.91
TREC4	0.85	0.89	0.92	0.95
TREC6	0.86	0.88	0.89	0.92

Table 5a: *Weighted recall* for pairs of database content summaries, as a function of the number of common categories in the database pairs and for the *Web*, *TREC4*, and *TREC6* data sets.

Data Set	*numCategories*			
	1	2	3	4
Web	0.46	0.51	0.53	0.55
TREC4	0.52	0.57	0.59	0.60
TREC6	0.53	0.57	0.58	0.61

Table 5b: *Unweighted recall* for pairs of database content summaries, as a function of the number of common categories in the database pairs and for the *Web*, *TREC4*, and *TREC6* data sets.

Data Set	*numCategories*			
	1	2	3	4
Web	0.50	0.59	0.67	0.69
TREC4	0.52	0.55	0.60	0.70
TREC6	0.52	0.55	0.57	0.62

Table 5c: *Spearman Rank Correlation Coefficient* for pairs of database content summaries, as a function of the number of common categories in the database pairs and for the *Web*, *TREC4*, and *TREC6* data sets.

Data Set	*Sampl. Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.903**	0.745
	QBS	Yes	**0.917**	0.745
	FPS	No	**0.912**	0.827
	FPS	Yes	**0.928**	0.827
Web	QBS	No	**0.962**	0.875
	QBS	Yes	**0.976**	0.875
	FPS	No	**0.989**	0.887
	FPS	Yes	**0.993**	0.887
TREC4	QBS	No	**0.937**	0.918
	QBS	Yes	**0.959**	0.918
	FPS	No	**0.980**	0.972
	FPS	Yes	**0.983**	0.972
TREC6	QBS	No	**0.959**	0.937
	QBS	Yes	**0.985**	0.937
	FPS	No	**0.979**	0.975
	FPS	Yes	**0.982**	0.975

Table 6a: Weighted recall *wr*.

Data Set	*Sampl. Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.589**	0.523
	QBS	Yes	**0.601**	0.523
	FPS	No	**0.623**	0.584
	FPS	Yes	**0.638**	0.584
Web	QBS	No	**0.438**	0.424
	QBS	Yes	**0.489**	0.424
	FPS	No	**0.681**	0.520
	FPS	Yes	**0.711**	0.520
TREC4	QBS	No	**0.402**	0.347
	QBS	Yes	**0.542**	0.347
	FPS	No	**0.678**	0.599
	FPS	Yes	**0.714**	0.599
TREC6	QBS	No	**0.549**	0.475
	QBS	Yes	**0.708**	0.475
	FPS	No	**0.731**	0.662
	FPS	Yes	**0.784**	0.662

Table 6b: Unweighted recall *ur*.

- the number of categories that they share, *numCategories*, where:

$$numCategories = |Path(Ideal(D_i)) \cap Path(Ideal(D_j))|,$$

  and $Ideal(D)$ is the correct classification of D, $Path(Ideal(D)) = \{category\ c\ |c \in Ideal(D)\ or\ c\ is\ an\ ancestor\ of\ some\ n \in Ideal(D)\}$, for $\tau_{es} = 0.25$ and $\tau_{ec} = 10$.

- the *wr*, *ur*, and *SRCC* values of their *correct* content summaries.

Figures 13a, 13b, and 13c report the *wr*, *ur*, and *SRCC* metrics, respectively, over all pairs of databases in the *Controlled* set and discriminated by *numCategories*. The larger the number of common categories between a pair of databases, the more similar their corresponding content summaries tend to be, according to the *wr*, *ur*, and *SRCC* metrics. Tables 5a, 5b, and 5c report the *wr*, *ur*, and *SRCC* metrics, respectively, over all pair of databases in the *Web*, *TREC4*, and *TREC6* data sets, confirming that databases that are classified under similar categories have more similar content summaries than databases under different topics.

6.3 Effect of Shrinkage

We now report experimental results on the quality of the content summaries generated by the shrinkage technique from Section 4.2. To keep our experiments manageable, we use content summaries extracted from *FP-SVM* with specificity threshold $\tau_{es} = 0.25$ and coverage threshold $\tau_{ec} = 10$, which give good classification decisions. We also pick *QBS-Lrd* over *QBS-Ord*, since the former method demonstrates similar performance at substantially smaller cost than the latter. (See Section 6.1 for a justification of this choice.) For conciseness, we now refer to *FP-SVM* as *FPS* and to *QBS-Lrd* as *QBS*. We evaluate the content summaries using the *Controlled*, *Web*, *TREC4*, and *TREC6* data sets.

Recall: We used the weighted recall and unweighted recall metrics to measure the vocabulary coverage of the shrunk content summaries. The shrunk content summaries include (with non-zero probability) every word in any content summary. Most words in any given content summary, however, tend to exhibit a very low probability. Therefore, to not inflate artificially the recall results (and conversely, to not hurt artificially the precision results), we drop from the shrunk content summaries every word w with $round(|D| \cdot \hat{p}_{\mathcal{R}}(w|D)) < 1$ during evaluation. Intuitively, we drop from the content summary all the words that are estimated to appear in less than one document.

Table 6a shows the weighted recall for different content summary construction techniques. Most methods exhibit high weighted recall, which shows that document sampling techniques identify the most frequent words in the database. Not surprisingly, shrinkage increases the (already high) *wr* values and all shrinkage-based methods have close-to-perfect *wr*. This improvement is statistically significant in all cases: a paired t-test [Mar03] showed significance at the 0.01% level. The improvement for the *Web* set is higher compared to that for the *TREC4* and *TREC6* data sets: the *Web* set contains larger databases, and the approximate content summaries are less complete than the respective approximate content summaries of *TREC4* and *TREC6*. Our shrinkage technique becomes increasingly useful for larger databases. To understand whether

Data Set	*Sampl. Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	0.989	**1.000**
	QBS	Yes	0.979	**1.000**
	FPS	No	0.948	**1.000**
	FPS	Yes	0.940	**1.000**
Web	QBS	No	0.981	**1.000**
	QBS	Yes	0.973	**1.000**
	FPS	No	0.987	**1.000**
	FPS	Yes	0.947	**1.000**
TREC4	QBS	No	0.992	**1.000**
	QBS	Yes	0.978	**1.000**
	FPS	No	0.987	**1.000**
	FPS	Yes	0.984	**1.000**
TREC6	QBS	No	0.978	**1.000**
	QBS	Yes	0.943	**1.000**
	FPS	No	0.976	**1.000**
	FPS	Yes	0.958	**1.000**

Table 7a: Weighted precision *wp*.

Data Set	*Sampl. Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	0.932	**1.000**
	QBS	Yes	0.921	**1.000**
	FPS	No	0.895	**1.000**
	FPS	Yes	0.885	**1.000**
Web	QBS	No	0.954	**1.000**
	QBS	Yes	0.942	**1.000**
	FPS	No	0.923	**1.000**
	FPS	Yes	0.909	**1.000**
TREC4	QBS	No	0.965	**1.000**
	QBS	Yes	0.955	**1.000**
	FPS	No	0.901	**1.000**
	FPS	Yes	0.856	**1.000**
TREC6	QBS	No	0.936	**1.000**
	QBS	Yes	0.847	**1.000**
	FPS	No	0.894	**1.000**
	FPS	Yes	0.850	**1.000**

Table 7b: Unweighted precision *up*.

Data Set	*Sampl. Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.723**	0.628
	QBS	Yes	**0.723**	0.628
	FPS	No	**0.765**	0.665
	FPS	Yes	**0.765**	0.665
Web	QBS	No	**0.904**	0.812
	QBS	Yes	**0.904**	0.812
	FPS	No	**0.917**	0.813
	FPS	Yes	**0.917**	0.813
TREC4	QBS	No	**0.981**	0.833
	QBS	Yes	**0.981**	0.833
	FPS	No	**0.943**	0.884
	FPS	Yes	**0.943**	0.884
TREC6	QBS	No	**0.961**	0.865
	QBS	Yes	**0.961**	0.865
	FPS	No	**0.937**	0.905
	FPS	Yes	**0.937**	0.905

Table 8: Spearman Correlation Coefficient.

Data Set	*Sampl. Method*	*Freq. Est.*	*Shrinkage*	
			Yes	*No*
Controlled	QBS	No	**0.364**	0.732
	QBS	Yes	**0.389**	0.645
	FPS	No	**0.483**	0.542
	FPS	Yes	**0.378**	0.503
Web	QBS	No	**0.361**	0.531
	QBS	Yes	**0.382**	0.472
	FPS	No	0.298	**0.254**
	FPS	Yes	0.281	**0.224**
TREC4	QBS	No	**0.296**	0.300
	QBS	Yes	**0.175**	0.180
	FPS	No	0.253	**0.203**
	FPS	Yes	0.193	**0.118**
TREC6	QBS	No	**0.305**	0.352
	QBS	Yes	**0.287**	0.354
	FPS	No	0.223	**0.193**
	FPS	Yes	0.301	**0.126**

Table 9: KL-divergence.

low-frequency words are present in the approximate and shrunk content summaries, we use the unweighted recall metric. Table 6b shows that the shrunk content summaries have higher unweighted recall as well.

Finally, recall is higher when shrinkage is used in conjunction with the frequency estimation technique. This behavior is to be expected: when frequency estimation is enabled, the words introduced by shrinkage are close to their real frequencies, and are used in precision and recall calculations. When frequency estimation is not used, the estimated frequencies of the same words are often below 0.5, and are therefore not used in precision and recall calculations.

Precision: A database content summary constructed using a document sample contains only words that appear in the database. In contrast, the shrunk content summaries may include words not in the corresponding databases. To measure the extent to which "spurious" words are added –with high weight– by shrinkage in the content summary, we use the *weighted precision (wp)* of $A(D)$ with respect to $S(D)$, $wp = \frac{\sum_{w \in W_A \cap W_S} \widehat{df}(w)}{\sum_{w \in W_A} \widehat{df}(w)}$. Table 7a shows that shrinkage decreases weighted precision by just 0.8% to 6%.

We also report the *unweighted precision (up)* metric, defined as $up = \frac{|W_A \cap W_S|}{|W_A|}$. This metric reveals how many words introduced in a content summary do not appear in the complete content summary (or, equivalently, in the underlying database). Table 7b reports the results for the *up* metric, which show that the shrinkage-based techniques have unweighted precision that is usually above 90% and always above 84%.

Word-Ranking Correlation: Table 8 shows that *SRCC* is higher for the shrunk content summaries. In general, *SRCC* is better for the shrunk than for the unshrunk content summaries ($p < 0.001$, according to a paired t-test): not only do the shrunk content summaries have better vocabulary coverage, as the recall figures show, but also the newly added words tend to be ranked properly.

Word-Frequency Accuracy: Our shrinkage-based algorithm modifies the probability estimates $\hat{p}(w|D)$ in the approximate summaries $A(D)$, in order to generate a summary whose probability distribution is

"closer" to that of the original $S(D)$. The *KL-divergence* compares the "similarity" of the $A(D)$ estimates against the real values in $S(D)$: $KL = \sum_{w \in W_A \cap W_S} p(w|D) \cdot \log \frac{p(w|D)}{\hat{p}(w|D)}$, where $p(w|D)$ is defined as $p(w|D) = \frac{tf(w,D)}{\sum_i tf(w_i,D)}$ and $tf(w, D)$ is the total number of occurrences of w in D. The KL metric takes values from 0 to infinity, with 0 indicating that the two content summaries being compared are equal.

Table 9 shows that shrinkage helps decrease large KL values. (Recall that lower KL values indicate higher quality summaries.) This is a characteristic of shrinkage [HTF01]: all summaries are shrunk towards some "common" content summary, which has an "average" distance from all the summaries. This effectively reduces the variance of the estimations and leads to reduced estimation "risk." However, shrinkage (moderately) hurts content-summary accuracy in terms of the KL metric in cases where KL is already low for the unshrunk summaries. We use this observation in our shrinkage-based database selection algorithm in Section 4.2.3, where our algorithm attempts to identify the cases where shrinkage is likely to help general database selection accuracy and avoids applying shrinkage in other cases.

Evaluation Conclusions: The general conclusion from our experiments on content summary quality is that shrinkage drastically improves content summary recall, at the expense of precision. The high *weighted* precision of the shrinkage-based summaries suggests that the spurious words introduced by shrinkage appear with low weight in the summaries, which should reduce any potential negative impact on database selection. Next, we present experimental evidence that the loss in precision ultimately does not hurt, since shrinkage improves overall database selection accuracy.

7 Experimental Results for Database Selection Accuracy

In this section, we evaluate the accuracy of the database selection algorithms that we presented in this article. We first describe our evaluation metric, and then we study the performance of the proposed database selection algorithms under a variety of settings. Just as in Section 6.3, we use *FP-SVM* (for conciseness, *FPS*) with specificity threshold $\tau_{es} = 0.25$ and coverage threshold $\tau_{ec} = 10$ and *QBS-Lrd* (for conciseness, *QBS*) as underlying content summary construction algorithms.

Consider a ranking of the databases $\vec{D} = D_1, \ldots, D_m$ according to the scores produced by a database selection algorithm for some query q. To measure the "goodness" or general "quality" of such a rank, we follow an evaluation methodology that is prevalent in the information retrieval community, and consider the number of documents in each database that are *relevant* to q, as determined by a human judge [SM83]. Intuitively, a good rank for a query includes –at the top– those databases with the largest number of relevant documents for the query. If $r(q, D_i)$ denotes the number of D_i documents that are relevant to query q, then $A(q, \vec{D}, k) = \sum_{i=1}^{k} r(q, D_i)$ measures the total number of relevant documents among the top-k databases in $\vec{D}$. To normalize this measure, we consider a hypothetical, "perfect" database rank $\vec{D}_H = D_{h_1}, \ldots, D_{h_m}$ in which databases are sorted by their $r(q, D_{h_i})$ value. (This is of course unknown to the database selection algorithm.) Then, we define the R_k metric for a query and database rank $\vec{D}$ as $R_k = \frac{A(q,\vec{D},k)}{A(q,\vec{D}_H,k)}$ [GGMT99]. A "perfect" ordering of k databases for a query yields $R_k = 1$, while a (poor) choice of k databases with no relevant content results in $R_k = 0$. We note that when a database receives the "default" score from a database selection algorithm (i.e., when the score assigned to a database for a query is equal to the score assigned to an empty query) we consider that the database is *not* selected for searching. This sometimes results in a database selection algorithm selecting fewer than k databases for a query.

The R_k metric relies on human-generated relevance judgments for the queries and documents. For our experiments on database selection accuracy, we focus on the *TREC4* and *TREC6* data sets, which include queries and associated relevance judgments.[25] We use queries 201-250 from TREC-4 with the *TREC4* data set and queries 301-350 from TREC-6 with the *TREC6* data set. The TREC-4 queries are long, with 8 to 34 words and an average of 16.75 words per query. The TREC-6 queries are shorter, with 2 to 5 words and an average of 2.75 words per query.

[25] We do not consider the *Web* and *Controlled* data sets of Section 3.1 for these experiments because of the lack of relevance judgments for them. In [IG02] we presented a preliminary evaluation of the hierarchical database selection algorithm of Section 4.1 over a subset of the *Web* data set, for a relatively low-scale evaluation. (This evaluation used relevance judgments provided by volunteer colleagues.) The results in [IG02] are consistent with those that we present here over the *TREC* data.

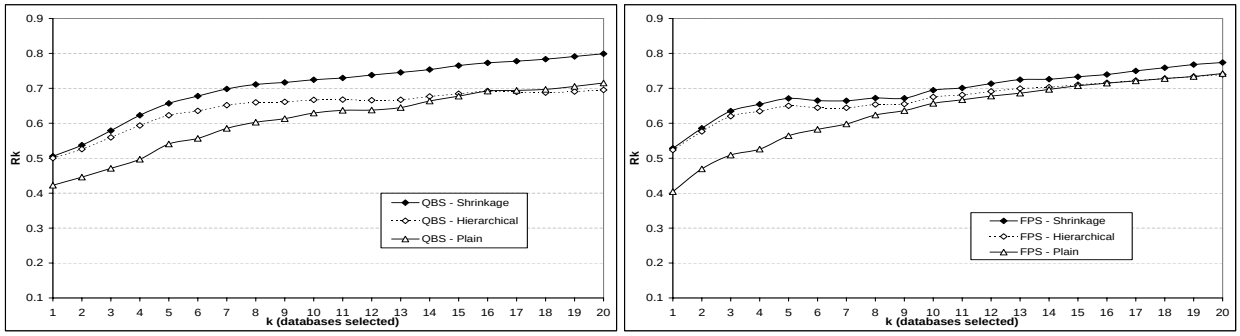

Figure 14a: The R_k ratio for CORI with stemming over the *TREC4* data set.

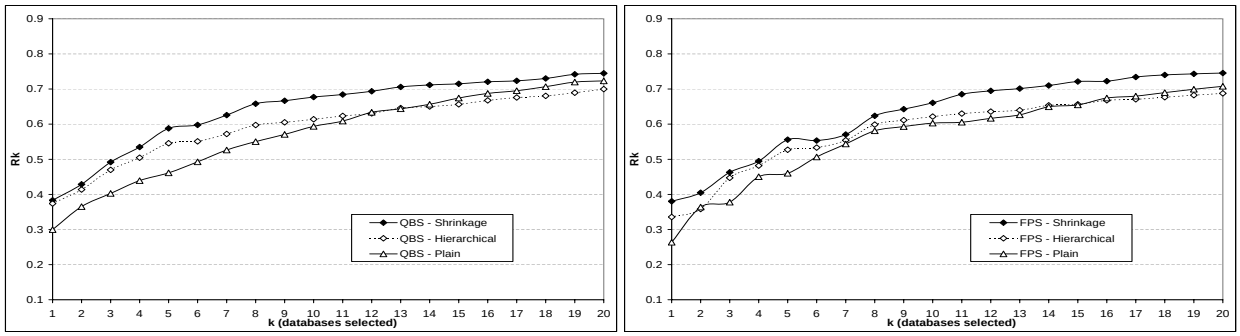

Figure 14b: The R_k ratio for CORI without stemming over the *TREC4* data set.

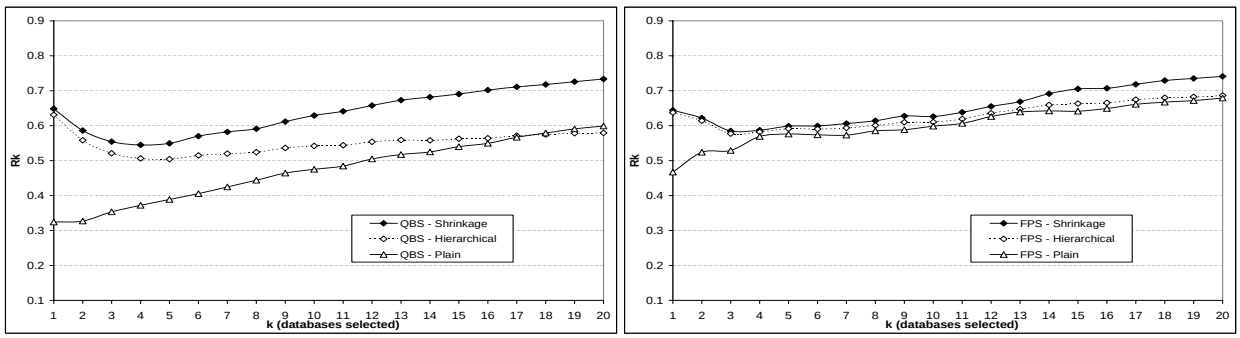

Figure 14c: The R_k ratio for CORI with stemming over the *TREC6* data set.

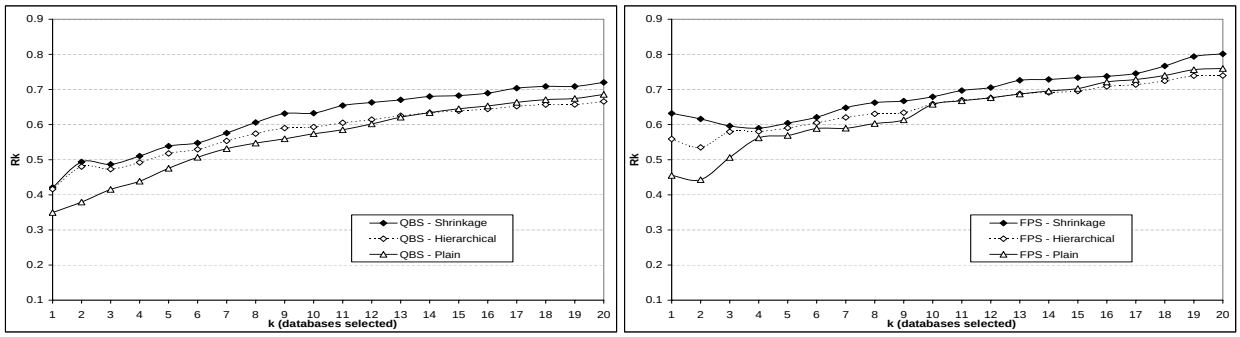

Figure 14d: The R_k ratio for CORI without stemming over the *TREC6* data set.

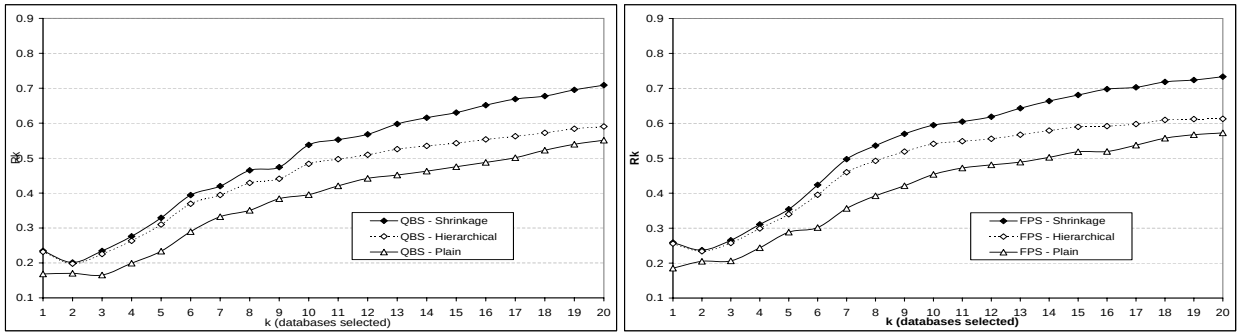

Figure 15a: The R_k ratio for bGlOSS with stemming over the *TREC4* data set.

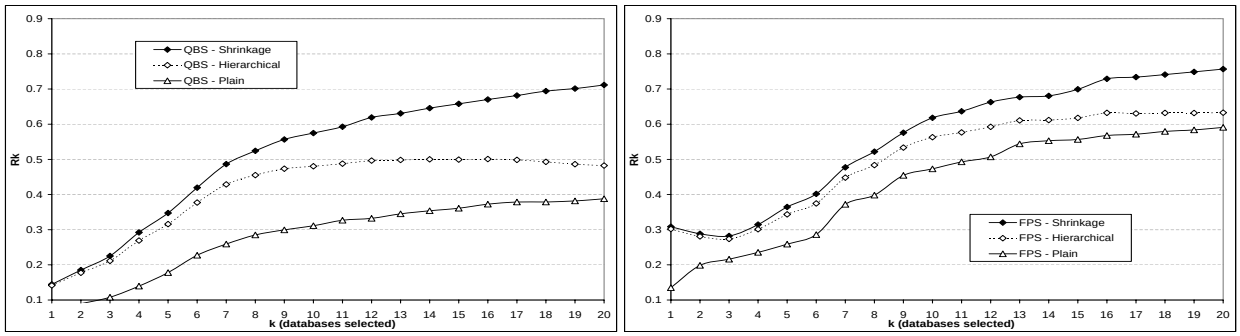

Figure 15b: The R_k ratio for bGlOSS without stemming over the *TREC4* data set.

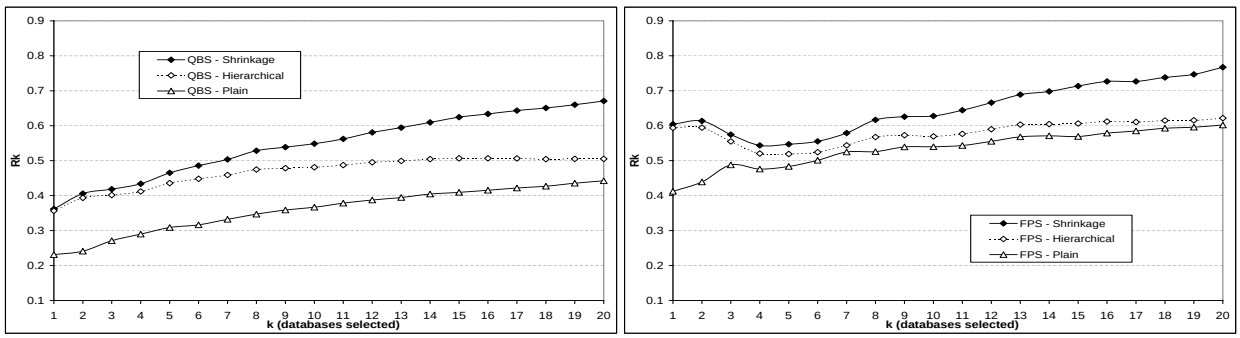

Figure 15c: The R_k ratio for bGlOSS with stemming over the *TREC6* data set.

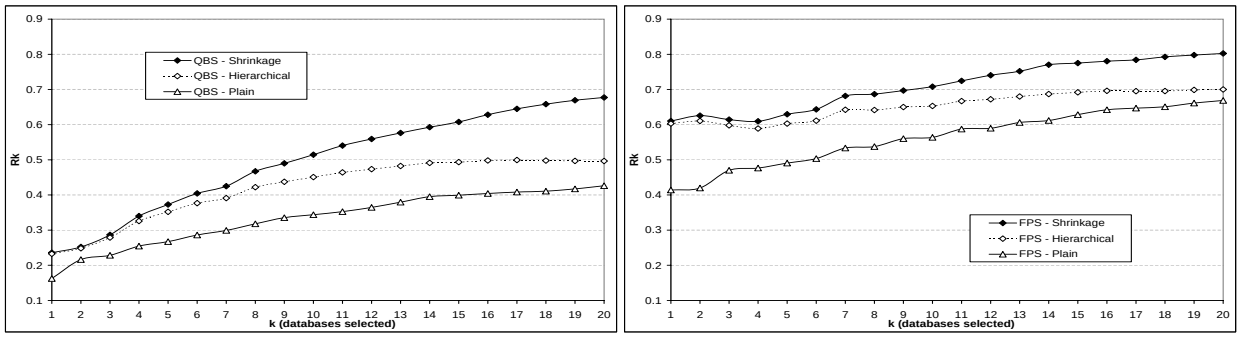

Figure 15d: The R_k ratio for bGlOSS without stemming over the *TREC6* data set.

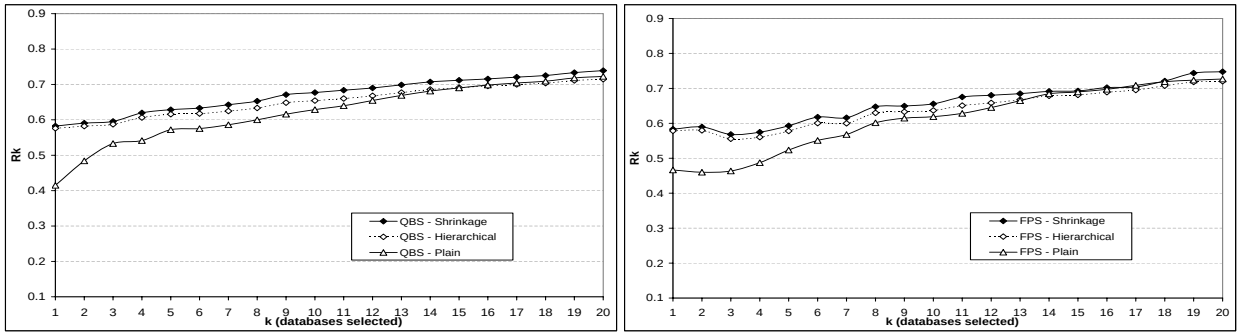

Figure 16a: The R_k ratio for LM with stemming over the *TREC4* data set.

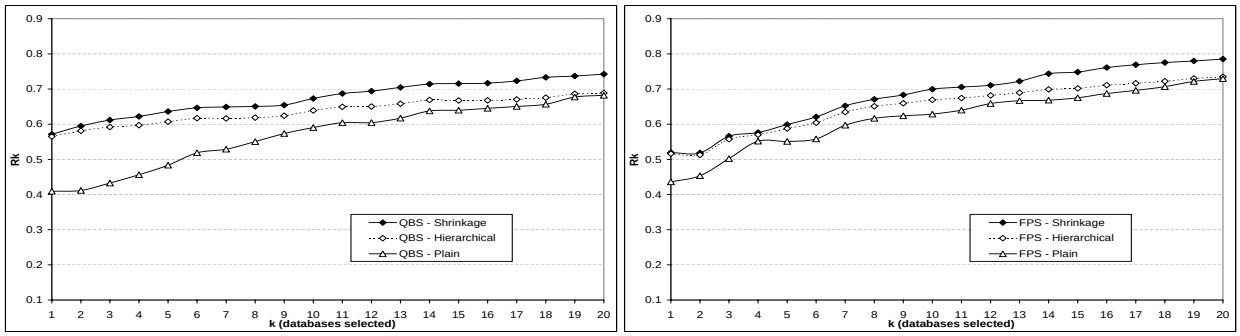

Figure 16b: The R_k ratio for LM without stemming over the *TREC4* data set.

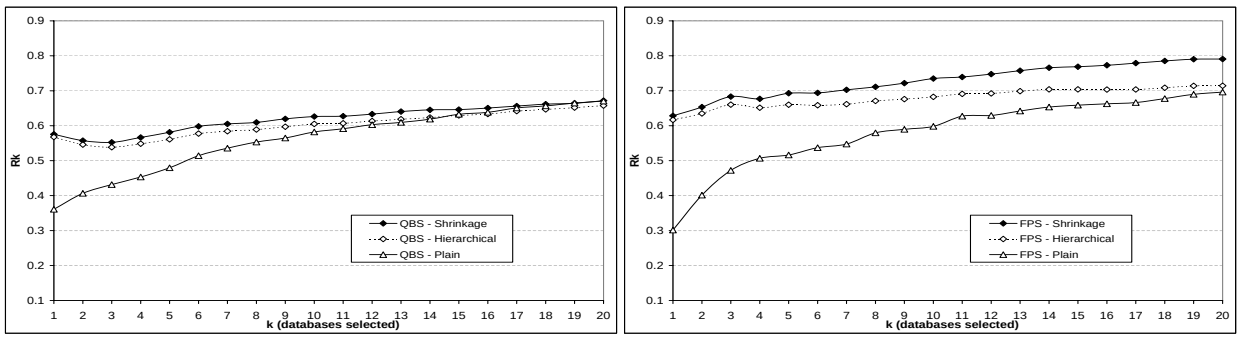

Figure 16c: The R_k ratio for LM with stemming over the *TREC6* data set.

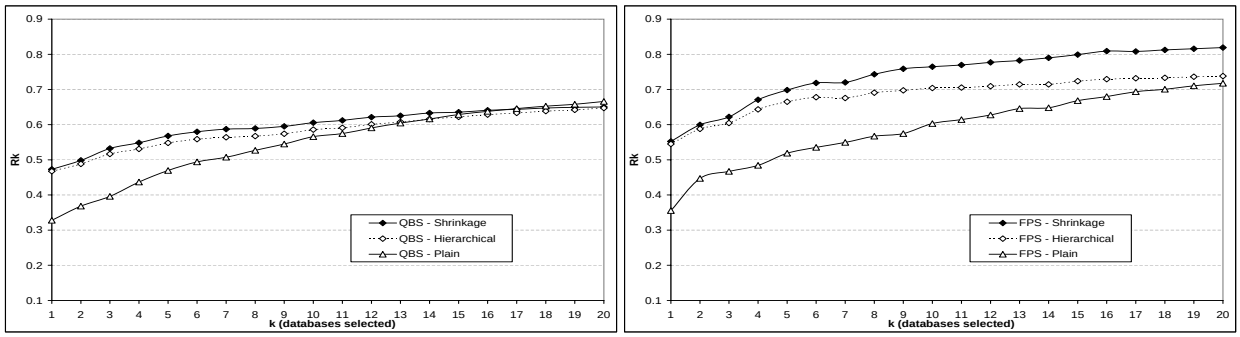

Figure 16d: The R_k ratio for LM without stemming over the *TREC6* data set.

Data Set	*Sampl. Method*	*Database Selection*	*Shrinkage Application*	*Data Set*	*Sampl. Method*	*Database Selection*	*Shrinkage Application*
TREC4	FPS	bGlOSS	35.42%	*TREC6*	FPS	bGlOSS	33.43%
		CORI	17.32%			CORI	13.12%
		LM	15.40%			LM	12.78%
	QBS	bGlOSS	78.12%		QBS	bGlOSS	58.94%
		CORI	15.68%			CORI	14.32%
		LM	17.32%			LM	11.73%

Table 10: Percentage of query-database pairs for which shrinkage was applied.

We considered eliminating stopwords (e.g., "the") from the queries, as well as applying stemming to the query and document words (e.g., so that a query *[computers]* matches documents with word "computing"). While the results improve with stopword elimination, a paired t-test showed that the difference in performance is not statistically significant; therefore, we only report results with stopword elimination. Stemming tends to improve performance for small values of k. The results are mixed when $k > 10$.

Figures 14a, 14b, 14c, and 14d show results for the CORI database selection algorithm. We used both the *TREC4* and *TREC6* data sets and queries, as well as the *QBS* and *FPS* content summary construction strategies (Section 5.2). We consider applying CORI over "unshrunk" content summaries (QBS-Plain and FPS-Plain), using the adaptive shrinkage-based strategy (QBS-Shrinkage and FPS-Shrinkage), and using the hierarchical algorithm (QBS-Hierarchical and FPS-Hierarchical). Figures 15a, 15b, 15c, and 15d show the results for the bGlOSS database selection algorithm, while Figures 16a, 16b, 16c, and 16d show the results for the LM database selection algorithm.

Overall, a paired t-test shows that QBS-Shrinkage improves the database selection performance over QBS-Plain, and this improvement is statistically significant ($p < 0.05$). FPS-Shrinkage also improves the database selection performance relative to FPS-Plain, but this improvement is statistically significant only when $k < 10$. We now describe the details of our findings.

Shrinkage vs. Plain: The first conclusion from our experiments is that QBS-Shrinkage and FPS-Shrinkage improve performance compared to QBS-Plain and FPS-Plain, respectively. Shrinkage helps because new words are added in the content summaries in a *database-* and *category-specific* manner. In Table 10, we report the number of times shrinkage was applied for each database-query pair and for each database selection algorithm. Since the queries for *TREC6* are shorter, shrinkage was applied comparatively fewer times for *TREC6* than for *TREC4*. Also, shrinkage was applied more frequently for bGlOSS than for LM and CORI. bGlOSS does not have any form of smoothing and assigns zero scores to databases whose content summaries do not contain a query word. This results in high variance for the bGlOSS scores, which in turn triggers the application of shrinkage.

Interestingly, Table 10 shows that shrinkage is applied relatively few times overall, yet its impact on database selection accuracy is large, as we have seen. To understand why, note that the Table 10 figures refer to *database-query* pairs. We have observed that the application of shrinkage for even a few critical databases for a given query can sometimes dramatically improve the quality of the database rank that is produced for the query. As a real example of this phenomenon, consider the TREC-6 query [*unexplained highway accidents*] and database *all-2*, which contains 92.5% of all the relevant documents for the query. Using the *LM* algorithm (for both *FPS* and *QBS*) database *all-2* is ranked 16th, resulting in low R_k values for any $k < 16$. Our adaptive shrinkage algorithm decides to use shrinkage for this specific database-query pair, and database *all-2* is ranked 3rd after application of shrinkage. This results in substantially larger R_k values for the shrinkage-based algorithms for $3 \leq k \leq 15$. While our adaptive database selection algorithm applied shrinkage in just 5% of the databases for this query (i.e., for just 5 databases out of 100), the resulting database rank for the query is significantly better than the rank produced with no shrinkage. In general, even limited applications of shrinkage tend to have a significant effect on the R_k ranking: the distribution of relevant documents across databases is typically skewed,[26] and only a small number of databases contain the majority of the relevant documents. Therefore, by ranking the important databases accurately, we can substantially improve the database selection performance.

Shrinkage vs. Hierarchical: QBS-Hierarchical and FPS-Hierarchical generally outperform their "plain" counterparts. This confirms our observation that categorization information helps compensate for incomplete summaries. Exploiting this categorization via shrinkage results in even higher accuracy: QBS-Shrinkage and

[26]This effect is not only limited to the *TREC* data sets, but it is true for the Web at large.

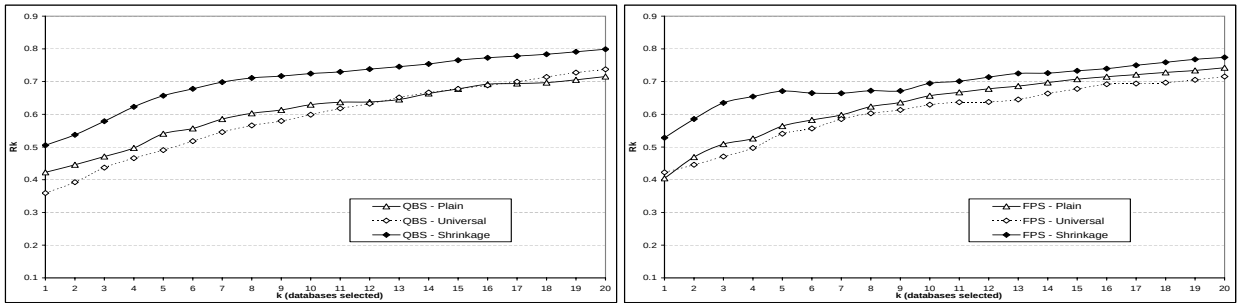

Figure 17a: The R_k ratio for CORI with stemming over the *TREC4* data set, with and without universal application of shrinkage.

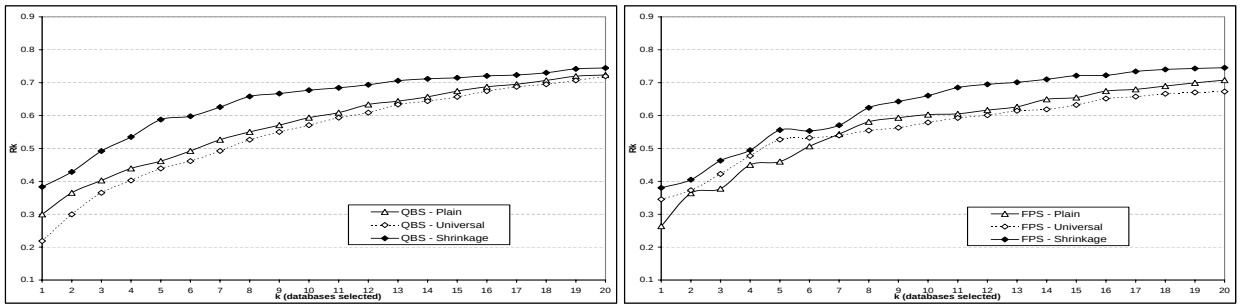

Figure 17b: The R_k ratio for CORI without stemming over the *TREC4* data set, with and without universal application of shrinkage.

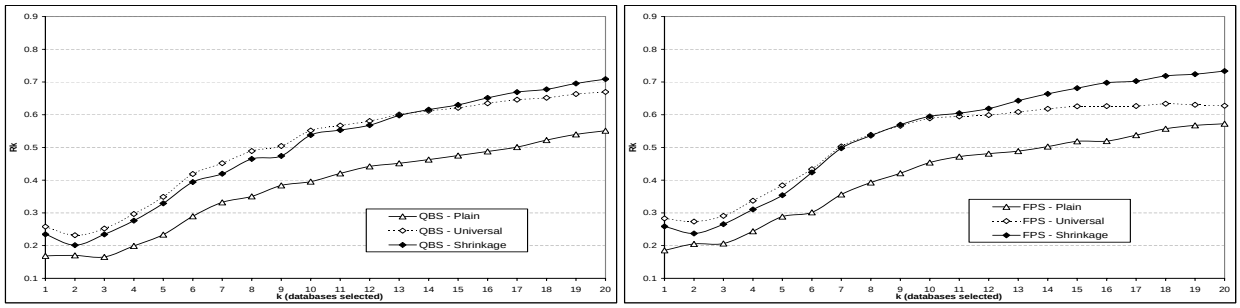

Figure 18a: The R_k ratio for bGlOSS with stemming over the *TREC4* data set, with and without universal application of shrinkage.

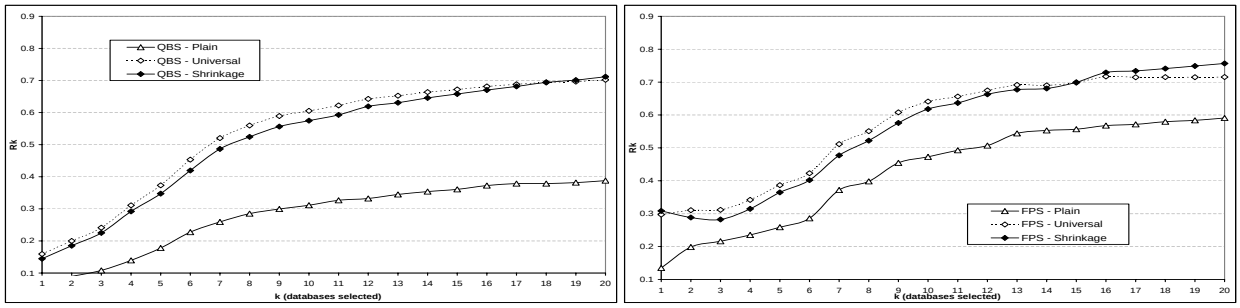

Figure 18b: The R_k ratio for bGlOSS without stemming over the *TREC4* data set, with and without universal application of shrinkage.

FPS-Shrinkage significantly outperform QBS-Hierarchical and FPS-Hierarchical. This improvement is due to the "flat" nature of our shrinkage method: while QBS-Shrinkage and FPS-Shrinkage can rank the databases "globally," QBS-Hierarchical and FPS-Hierarchical have to make irreversible choices at each *category* level of the hierarchy. Even when a chosen category contains only a small number of databases with relevant documents, the hierarchical algorithm continues to select (irrelevant) databases from the (relevant) category. When a query "cuts across" multiple categories, the hierarchical algorithm might fail to select the appropriate databases. In contrast, our shrinkage-based approach can potentially select databases from multiple categories and hence manages to identify the appropriate databases for a query, no matter if they are similarly classified or not.

Adaptive vs. Universal Application of Shrinkage: The strategy in Section 4.2.3 dynamically decides when to apply shrinkage for database selection. To understand whether this decision step is necessary, we evaluated the performance of the algorithms when we *always* decide to use shrinkage (i.e., when the $\widehat{\mathcal{R}}(D_i)$ content summary is always chosen in Figure 10). Figures 17a, 17b, 18a, and 18b show the *TREC4* results for CORI and bGlOSS, with QBS-Universal and FPS-Universal denoting universal application of shrinkage.[27] The only case where QBS-Universal and FPS-Universal are better than QBS-Plain and FPS-Plain, respectively, is for bGlOSS (Figures 18a and 18b): unlike CORI and LM, bGlOSS does not have any form of "smoothing" already built in, so if a query word is not present in a content summary, bGlOSS assigns a zero score to the database. Unlike bGlOSS, CORI and LM perform worse when we apply shrinkage universally than when we do so adaptively. The only exception is for content summaries created without the use of stemming, and only for small values of k, but even in this case the small improvement is not statistically significant. This result indicates that CORI and LM handle incomplete content summaries in a more graceful way than bGlOSS does, since both CORI and LM have a form of smoothing already embedded.

Frequency Estimation: We also examined the effect of frequency estimation (Section 3.2) on database selection. Figures 19a, 19b, 19c, and 19d show the results for CORI over *TREC4* and *TREC6*. In general, frequency estimation affected only the performance of the CORI database selection algorithm, and had only a small effect on the performance of bGlOSS and LM, so we do not show plots for these two techniques. The reason is that bGlOSS and LM rely on *probabilities* that remain virtually unaffected after the frequency estimation step. In contrast, CORI relies on document frequencies. Figures 19a, 19b, 19c, and 19d show that, when shrinkage is used, frequency estimation improves the performance of CORI, by 10% to 30% for small values of k, with respect to the case where the raw word frequencies for the document sample are used. Interestingly, frequency estimation alone, without shrinkage, does not improve database selection as much, hinting that more accurate frequency estimates only improve database selection accuracy substantially when the underlying content summaries are sufficiently complete.

Evaluation Conclusions: A general conclusion from the experiments is that the *adaptive* application of shrinkage significantly improves database selection when selection decisions are based on sparse content summaries. An interesting observation is that the *universal* application of shrinkage is not always beneficial, indicating that for cases where selection decisions are already accurate, shrinkage negatively affects the selection process. Another conclusion is that stemming-based summaries are typically better than their non-stemmed counterparts, since stemming reduces data sparseness. The difference is significant for small numbers of selected databases, which indicates that stemming results in better database rankings.

8 Related Work

This section addresses the literature relevant to the topics covered in this article. Portions of this article appeared in [IG02] and [IG04]. First, Section 8.1 reviews work on database selection. Then, Section 8.2 discusses related work for content summary construction. Finally, Section 8.3 summarizes various applications of query probing, a technique that we used extensively in this article.

8.1 Database Selection

A large body of work has been devoted to distributed information retrieval, or metasearching, over text databases. As we discussed, a crucial task for a metasearcher is database selection, which requires that the

[27] The results for *TREC6* are similar, while the results for LM are analogous to those for CORI.

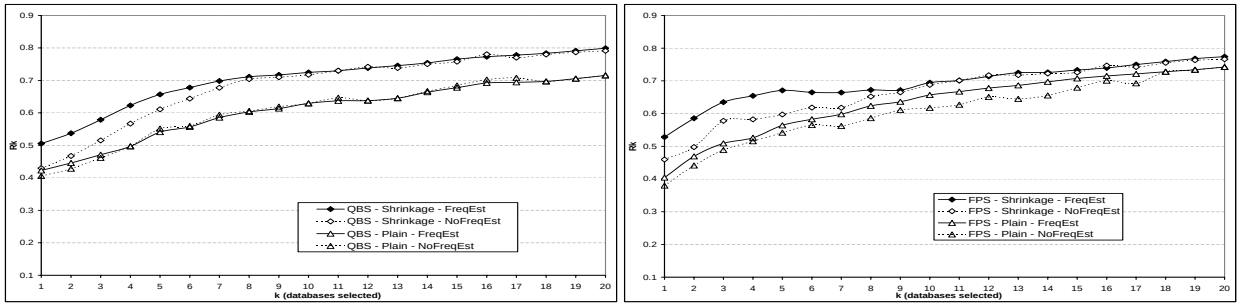

Figure 19a: The R_k ratio for CORI with stemming over the *TREC4* data set, for summaries generated with ("-FreqEst") and without ("-NoFreqEst") the use of frequency estimation.

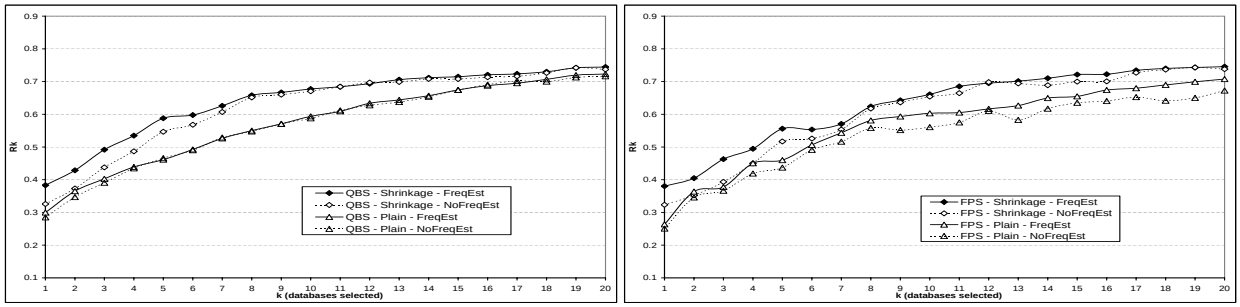

Figure 19b: The R_k ratio for CORI without stemming over the *TREC4* data set, for summaries generated with ("-FreqEst") and without ("-NoFreqEst") the use of frequency estimation.

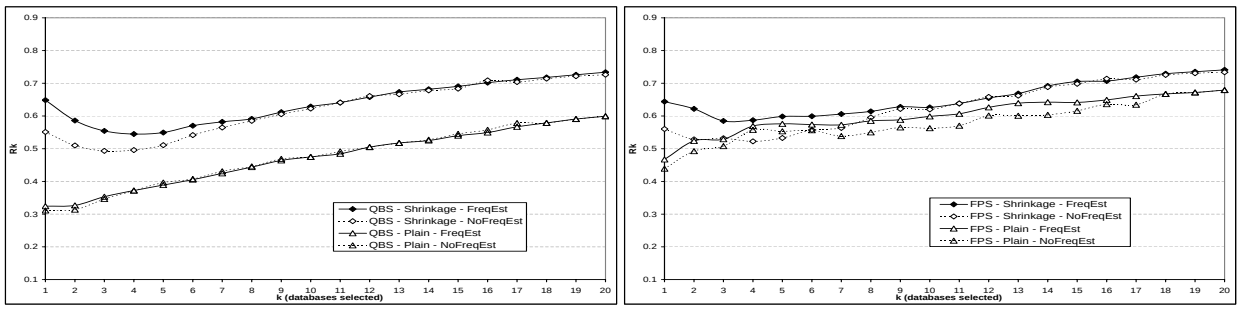

Figure 19c: The R_k ratio for CORI with stemming over the *TREC6* data set, for summaries generated with ("-FreqEst") and without ("-NoFreqEst") the use of frequency estimation.

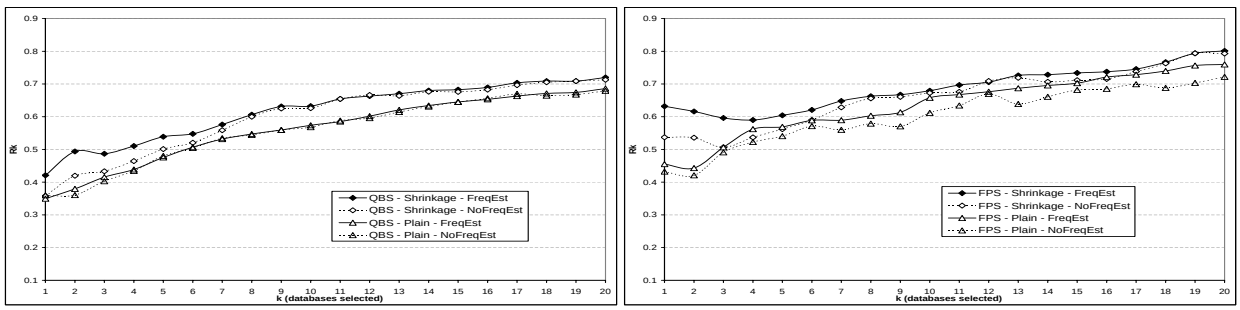

Figure 19d: The R_k ratio for CORI without stemming over the *TREC6* data set, for summaries generated with ("-FreqEst") and without ("-NoFreqEst") the use of frequency estimation.

metasearcher have summaries of the database contents.

Early database selection techniques relied on human-generated database descriptions. WAIS [KMG+93] uses such descriptions and ranks databases according to their similarity to the queries. In Search Broker [MB97], each database is manually tagged with two or three category index descriptors. At query time, users specify the query category and then Search Broker selects the appropriate databases. Chakravarthy and Haase [CH95] use Wordnet [Fel98] to complement the manually assigned keywords that are used to describe each database for database selection.

More robust database selection approaches rely on statistical metadata about the contents of the databases, generally following the type of content summaries used in this article. CORI [CLC95, XC98] uses inference networks together with this kind of content summaries to select the best databases for a query. (We used CORI for our experiments in Section 7.) GlOSS [GGMT99] uses content summaries and selects databases for a query according to some notion of *goodness* for a query. GlOSS can choose among a variety of definitions of *goodness*, some of which depend on the retrieval model supported by the databases. (We used bGlOSS, a variant of GlOSS originally introduced for boolean databases, for our experiments in Section 7.) Yuwono and Lee [YL97] use content summaries and rank databases according to the *cue validity* of the query words: a query word w has high cue validity for a database D if the probability of observing w in D is comparatively higher than in other databases. Meng et al. [MLY+98, YML+99] also rely on content summaries to identify the databases that contain the highest number of documents similar to a query, and similarity is computed using the cosine similarity metric. Meng et al. use a variety of methods to estimate the weight of the words in the database, and propose to keep significant covariance statistics for word pairs that appear often together. The storage requirements for the content summaries in [MLY+98] are much higher compared to the other methods that ignore the covariance statistics, such as [CLC95, XC98, GGMT99, YL97], which we described above. In a similar approach, Yu et al. [YMWL01] rank the databases for a query according to the highest similarity of any document in each database and the query. Baumgarten [Bau97, Bau99] proposes a probabilistic framework for database selection and uses content summaries to derive the $\hat{p}(w|D)$ probability estimates that are used during querying. Most approaches that use content summaries rely either on access to all documents or on metadata directly exported by the databases, using, for example, a protocol like STARTS [GCGMP97].

French et al. [FPV+98, FPC+99, PFC+00, PF03] present experimental evaluations of database selection algorithms. Their main conclusion is that CORI is robust and performs better than other database selection algorithms for a variety of data sets. Results by Xu and Croft [XC99] and Si and Callan [SJCO02] indicate that a language modeling (LM) approach for database selection works better than CORI for topically focused databases. (We used the LM algorithm for our experiments in Section 7.) Xu and Croft [XC99] and Larkey et al. [LCC00] show that organizing documents by topic helps improve database selection accuracy. Our results in Section 7 are consistent with these findings, since they show that classification-aware database selection algorithms perform better than algorithms that ignore the classification information.

Our database selection techniques in Section 4 are built on top of an arbitrary "base" database selection algorithm. We reported experiments using CORI, bGlOSS, and LM. Our experimental results show that our techniques improve database selection –in the face of sparse data– when used in conjunction with a variety of existing "flat" algorithms. In the future, our techniques can continue to leverage new database selection algorithms that rely on content summaries to make the selection decisions. An example of a new algorithm that we might use in the future is Si and Callan's ReDDE algorithm [SC03].

Other database selection algorithms rely on hierarchical classification schemes –mostly *for efficiency*– to direct queries to appropriate categories of the hierarchy [Dol98, She95, GGMT99, CY01, YML+99]. The hierarchical database selection algorithm in [She95] uses intentionally small content summaries that contain only the high-frequency terms that characterize each category. The *hGlOSS* system [GGMT99] focuses on the efficiency of selection and does not exploit any topic similarity of the databases. Similarly, the hierarchical organization in [Dol98] focuses on efficiency and does not exploit the clustering of similar databases under the same categories. Fuhr [Fuh99] briefly discusses the hierarchical database selection problem, but no special clustering of similar databases is considered to improve the hierarchical selection task. The above hierarchical algorithms also need access to all documents or metadata directly exported by the databases. Our hierarchical database selection algorithm in Section 4.1 first appeared in [IG02]; this algorithm uses a topic hierarchy not only for efficiency, but also for improving the quality of the database selection decisions in the presence of sparse content summaries.

Other approaches rely on users providing relevance judgments to create a profile of each database. Voorhees et al. [VGJL95] use a set of training queries to learn the usefulness of each database and decide how many documents to retrieve from each. ProFusion [GWG96] and SavvySearch [DH97] also exploit historic data to learn the performance of each database for various types of queries. Then, databases that exhibit higher performance for a query are preferred over others that tend to return worse results. Fuhr [Fuh99] uses a decision-theoretic model to decide whether to use a database and to determine how many documents to retrieve from a selected database. The method in [Fuh99] tries to minimize the cost of retrieval and assumes that the precision-recall curves of the underlying retrieval system either are known or can be estimated.

8.2 Constructing Database Content Summaries

Unfortunately, hidden-Web text databases do not usually export any metadata about their contents and do not offer immediate access to their contents. Callan et al. [CCD99, CC01] probe databases with semi-random queries to extract content summaries from autonomous databases. (See Section 2 for a detailed discussion of this technique.) We used Callan et al.'s algorithm extensively in our experiments of Sections 6 and 7. Monroe et al. [MFP02] present and evaluate small variations of the algorithm from [CCD99, CC01]. Craswell et al. [CBH00] compared database selection algorithms in the presence of incomplete content summaries, extracted using document sampling, and observed that the performance of the algorithms deteriorated with respect to their behavior over complete summaries. Sugiura and Etzioni [SE00] proposed the *Q-Pilot* technique, which uses query expansion to route Web queries to the appropriate search engines and characterizes databases using words that appear in the Web pages that host the search interfaces, as well as words that appear in other Web pages that link to the databases. We used an adaptation of *Q-Pilot* for content summary generation in a preliminary experimental evaluation [IG02] of our hierarchical database selection algorithm of Section 4.1. Our experiments showed that the *Q-Pilot* content summaries are not sufficient for accurate database selection. Hawking and Thistlewaite [HT99] used query probing to perform database selection by ranking databases by similarity to a given query. Their algorithm assumed that the query interface to the database can handle normal queries and query probes differently, and that the cost to handle query probes is smaller than that for normal queries.

A preliminary version of the algorithm of Section 3 for constructing content summaries appeared in [IG02]. The frequency estimation algorithm in [IG02] managed to produce relatively accurate frequency estimates for database words that appeared in sample-based content summaries. However, a problem with the algorithm in [IG02] is the assumption that the rank of a word in a database coincides with the word's rank in a document sample extracted from the database. Unfortunately, this assumption does not hold in general, and is largely false for words that appear in a database only a relatively small number of times. In Section 3.2, we presented an improved frequency estimation algorithm that addresses this problem and produces significantly more accurate word frequency estimates for database words that appear only a relatively small number of times.

Along a related research direction, Si and Callan [SC03] show that database selection performance can be improved by considering database size estimates within their ReDDE database selection algorithm. ReDDE retains the documents retrieved during content summary construction and uses this document sample to estimate the distribution of relevant documents across databases. Further studies by Si and Callan [SC04] show that CORI and LM are only marginally affected when used in conjunction with the database size estimation method from [SC03]. This result is consistent with the behavior that we observed for CORI (without use of shrinkage) with our frequency estimation method.

Our content summary construction technique in Section 4.2 appeared in [IG04] and is based on the work by McCallum et al. [MRMN98], who introduced a shrinkage-based approach for hierarchical document classification in the face of sparse data. Shrinkage is a form of *smoothing* and smoothing has been used extensively in the area of speech recognition [Jel99] to improve probability estimates in language models. Language modeling has also been used for information retrieval [CL03]. Notably, smoothing is present in recent language modeling approaches to information retrieval [ZL01, ZL02, ZL04]. An interesting direction for future work is to examine the performance of smoothing models other than shrinkage for database selection, especially in the presence of database classification information.

Liu et al. [LLCC04] estimate the potential inaccuracy of the database rank produced for a query by a database selection algorithm. If this inaccuracy is unacceptably large, then the query is dynamically

evaluated on a few carefully chosen databases to reduce the uncertainty associated with the database rank. This work does not take content-summary accuracy into consideration. In contrast, in Section 4.2.3, we addressed the scenario where summaries are derived from document samples –and are hence incomplete– and decide dynamically whether shrinkage should be applied, without actually querying databases during database selection. The bulk of Section 4.2.3 appeared originally in [IG04], where we described a generic method for computing the mean and the variance of a database score distribution when we use sample-based content summaries. The method in [IG04] did not use the fact that most database selection algorithms assume independence of the query words. We now exploit this property in Appendix B to substantially improve both the accuracy and the runtime performance of the mean-variance computation, which, in turn, improves substantially the runtime performance of our database selection algorithm of Section 4.2.3.

8.3 Miscellaneous Applications of Query Probing

In this article, we used query probing for the extraction of content summaries from text databases. Query probing has helped in other related tasks. Gravano et al. [GIS03] use a small number of query probes derived using state-of-the-art machine learning techniques to categorize a text database in a topic hierarchy. (We briefly reviewed this algorithm in Section 2.3 and used it extensively in subsequent sections.) Perkowitz et al. [PDEW97] use it to automatically understand query forms and extract information from Web databases to build a comparative shopping agent. New forms of crawlers [RGM01] use query probing to automatically interact with Web forms and crawl the contents of hidden-Web databases. Cohen and Singer [CS96] use RIPPER to learn queries that retrieve mainly documents about a specific category. The queries are used at a later time to retrieve new documents about this category. Flake et al. [FGLG02] extract rules from non-linear SVMs that identify documents with a common characteristic (e.g., "calls for papers"). The generated rules are used to modify queries sent to a search engine, so that the queries retrieve mostly documents of the desired kind. Grefenstette and Nioche [GN00] use query probing to determine the use of different languages on the Web. The query probes are words that appear only in one language. The number of matches generated for each probe is subsequently used to estimate the number of Web pages written in each language. Ghani et al. [GJM01] automatically generate queries to retrieve documents written in a specific language. Meng et al. [MYL99] used guided query probing to determine sources of heterogeneity in the algorithms used to index and search locally at each text database. Bergholz and Chidlovskii [BC04] probe a database with a carefully selected set of queries to identify the characteristics of the query language. Finally, the *QXtract* system [AG03] automatically generates queries to improve the efficiency of a given information extraction system such as Snowball [AG00] or Proteus [YG98] over large text databases. Specifically, *QXtract* learns queries that tend to match those database documents that are useful for the extraction task at hand. The information extraction system can then focus only on these documents, which results in large performance improvements.

9 Conclusion

Database selection is critical to building efficient metasearchers that interact with potentially large numbers of databases. Exhaustively searching all available databases to answer each query is impractical (or even not possible) in increasingly common scenarios. Current database selection algorithms rely on statistical summaries about the contents of the databases to select the best databases for a given query but, unfortunately, the databases accessible through the Web do not generally export these statistics. In this article, we presented efficient algorithms for constructing content summaries for such databases. Our algorithms create content summaries of higher quality than alternate approaches and, additionally, categorize databases in a classification scheme. We also presented a shrinkage-based technique that further improves the quality of the generated content summaries. The shrinkage-based content summaries are more complete than their "unshrunk" counterparts. Our shrinkage-based technique achieves this performance gain efficiently, without requiring any increase in the size of the document samples.

We also presented techniques for improving the performance of database selection algorithms in the case where database content summaries are derived from relatively small document samples. Such summaries are typically incomplete, and this can hurt the performance of database selection algorithms. We showed that classification-aware database selection algorithms can significantly improve the accuracy of the selection

decisions in the face of incomplete content summaries. Both the hierarchical database selection algorithm of Section 4.1 and the adaptive, shrinkage-based database selection algorithm of Section 4.2 perform better than their counterparts that do not exploit database classification. Furthermore, we showed that the shrinkage-based strategy outperforms the hierarchical database selection algorithm: the hierarchical algorithm initially selects databases under a single subtree of the classification hierarchy, so the hierarchical algorithm fails to select the appropriate databases for queries that "cut across" multiple categories. Shrinkage, on the other hand, embeds the category information in the content summaries. Therefore, a "flat" database selection algorithm can exploit the classification information without being constrained by the classification hierarchy.

References

[Ada02] Lada Ariana Adamic. Zipf, power-laws, and Pareto – A ranking tutorial. Online at `http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html`, 2002.

[AG00] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL 2000)*, 2000.

[AG03] Eugene Agichtein and Luis Gravano. Querying text databases for efficient information extraction. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE 2003)*, 2003.

[Baa06] R. Harald Baayen. *Word Frequency Distributions*. Springer, 2006.

[Bau97] Christoph Baumgarten. A probabilistic model for distributed information retrieval. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'97*, pages 258–266, 1997.

[Bau99] Christoph Baumgarten. A probabilistic solution to the selection and fusion problem in distributed information retrieval. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'99*, pages 246–253, 1999.

[BC04] Andre Bergholz and Boris Chidlovskii. Using query probing to identify query language features on the web. In *Distributed Multimedia Information Retrieval, SIGIR 2003 Workshop on Distributed Information Retrieval, Revised Selected and Invited Papers (LNCS 2924)*, pages 21–30, 2004.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[Bri00] BrightPlanet.com LLC. The Deep Web: Surfacing hidden value. Available at `http://www.completeplanet.com/Tutorials/DeepWeb/index.asp`, July 2000.

[CBH00] Nick Craswell, Peter Bailey, and David Hawking. Server selection on the World Wide Web. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL 2000)*, pages 37–46, 2000.

[CC01] James P. Callan and Margaret Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001.

[CCD99] James P. Callan, Margaret Connell, and Aiqun Du. Automatic discovery of language models for text databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, pages 479–490, 1999.

[CH95] Anil Srinivasa Chakravarthy and Kenneth William Haase, Jr. Netserf: Using semantic knowledge to find Internet information archives. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, pages 4–11, 1995.

[CL03] William Bruce Croft and John Lafferty. *Language Modeling for Information Retrieval*. Kluwer Academic Publishers, July 2003.

[CLC95] James P. Callan, Zhihong Lu, and William Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, pages 21–28, 1995.

[Coh96] William Weston Cohen. Learning trees and rules with set-valued features. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), Eighth Conference on Innovative Applications of Artificial Intelligence (IAAI-96)*, pages 709–716, 1996.

[CS96] William Weston Cohen and Yoram Singer. Learning to query the web. In *AAAI Workshop on Internet-Based Information Systems*, pages 16–25, 1996.

[CY01] Yong S. Choi and Suk I. Yoo. Text database discovery on the Web: Neural net based approach. *Journal of Intelligent Information Systems*, 16(1):5–20, January 2001.

[DH97] Daniel Dreilinger and Adele E. Howe. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems*, 15(3):195–222, 1997.

[DHS00] Richard Oswald Duda, Peter Elliot Hart, and David G. Stork. *Pattern Classification*. Wiley, 2nd edition, 2000.

[DLR77] Arthur Pentland Dempster, Nan McKenzie Laird, and Donald Bruce Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B(39):1–38, 1977.

[Dol98] Ron A. Dolin. *Pharos: A scalable distributed architecture for locating heterogeneous information sources*. PhD thesis, University of California, Santa Barbara, 1998.

[Fel98] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, May 1998.

[FGLG02] Gary Flake, Eric Glover, Steve Lawrence, and Clyde Lee Giles. Extracting query modifications from nonlinear SVMs. In *Proceedings of the 11th International World Wide Web Conference (WWW11)*, 2002.

[FPC+99] James Cornelius French, Allison Lane Powell, James P. Callan, Charles Lowell Viles, Travis Emmitt, Kevin J. Prey, and Yun Mou. Comparing the performance of database selection algorithms. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'99*, pages 238–245, 1999.

[FPV+98] James Cornelius French, Allison Lane Powell, Charles Lowell Viles, Travis Emmitt, and Kevin 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, SIGIR'98*, pages 121–129, 1998.

[Fuh99] Norbert Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, May 1999.

[GCGMP97] Luis Gravano, Kevin Chen-Chuan Chang, Héctor García-Molina, and Andreas Paepcke. *STARTS*: Stanford proposal for Internet meta-searching. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD'97)*, pages 207–218, 1997.

[GGMT99] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. *GlOSS*: Text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2):229–264, June 1999.

[GIS03] Luis Gravano, Panagiotis G. Ipeirotis, and Mehran Sahami. QProber: A system for automatic classification of hidden-web databases. *ACM Transactions on Information Systems*, 21(1):1–41, January 2003.

[GJM01] Rayid Ghani, Rosie Jones, and Dunja Mladenic. Using the web to create minority language corpora. In *Proceedings of the 2001 ACM Conference on Information and Knowledge Management (CIKM 2001)*, pages 279–286, 2001.

[GN00] Gregory Grefenstette and Julien Nioche. Estimation of English and non-English language use on the WWW. In *Recherche d'Information Assistée par Ordinateur (RIAO 2000)*, 2000.

[GWG96] Susan Gauch, Guijun Wang, and Mario Gomez. ProFusion*: Intelligent fusion from multiple, distributed search engines. *The Journal of Universal Computer Science*, 2(9):637–649, September 1996.

[Har96] Donna Harman. Overview of the fourth Text REtrieval Conference (TREC-4). In *NIST Special Publication 500-236: The Fourth Text REtrieval Conference (TREC-4)*, pages 1–24, 1996.

[HT99] David Hawking and Paul B. Thistlewaite. Methods for information server selection. *ACM Transactions on Information Systems*, 17(1):40–76, January 1999.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Harold Friedman. *The Elements of Statistical Learning*. Springer Verlag, August 2001.

[IG02] Panagiotis G. Ipeirotis and Luis Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002)*, pages 394–405, 2002.

[IG04] Panagiotis G. Ipeirotis and Luis Gravano. When one sample is not enough: Improving text database selection using shrinkage. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, pages 767–778, 2004.

[Jel99] Frederick Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, 1999.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML-98, 10th European Conference on Machine Learning*, pages 137–142, 1998.

[KMG+93] Brewster Kahle, Harry Morris, Johnathan Goldman, Thomas Erickson, and John Curran. Interfaces for distributed systems of information servers. *Journal of the American Society for Information Science*, 44(8):453–467, September 1993.

[LCC00] Leah Sue Larkey, Margaret E. Connell, and James P. Callan. Collection selection and results merging with topically organized U.S. patents and TREC data. In *Proceedings of the 2000 ACM Conference on Information and Knowledge Management (CIKM 2000)*, pages 282–289, 2000.

[LLCC04] Zhenyu Liu, Chang Luo, Junghoo Cho, and Wesley Chu. A probabilistic approach to metasearching with adaptive probing. In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE 2004)*, pages 547–559, 2004.

[Man88] Benoit B. Mandelbrot. *Fractal Geometry of Nature*. W. H. Freeman & Co., 1988.

[Mar03] Joaquim P. Marques De Sá. *Applied Statistics*. Springer Verlag, 2003.

[MB97] Udi Manber and Peter Alfred Bigot. The Search Broker. In *1st USENIX Symposium on Internet Technologies and Systems (USITS 1997)*, 1997.

[MFP02] Gary Anthony Monroe, James Cornelius French, and Allison Lane Powell. Obtaining language models of web collections using query-based sampling techniques. In *35th Annual Hawaii International Conference on System Sciences (HICSS 2002)*, pages 67–73, 2002.

[MLY+98] Weiyi Meng, King-Lup Liu, Clement Tak Yu, Xiaodong Wang, Yuhsi Chang, and Naphtali Rishe. Determining text databases to search in the Internet. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB'98)*, pages 14–25, 1998.

[MRMN98] Andrew McCallum, Ronald Rosenfeld, Tom Michael Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 359–367, 1998.

[MYL99] Weiyi Meng, Clement Tak Yu, and King-Lup Liu. Detection of heterogeneities in a multiple text database environment. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS 1999)*, pages 22–33, 1999.

[PDEW97] Mike Perkowitz, Robert B. Doorenbos, Oren Etzioni, and Daniel Sabey Weld. Learning to understand information on the Internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, March 1997.

[PF03] Allison Lane Powell and James Cornelius French. Comparing the performance of collection selection algorithms. *ACM Transactions on Information Systems*, 21(4):412–456, October 2003.

[PFC+00] Allison Lane Powell, James Cornelius French, James P. Callan, Margaret Connell, and Charles Lowell Viles. 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, SIGIR 2000*, pages 232–239, 2000.

[Qui92] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1992.

[RGM01] Sriram Raghavan and Héctor García-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pages 129–138, 2001.

[SC03] Luo Si and James P. Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2003*, pages 298–305, 2003.

[SC04] Luo Si and James P. Callan. The effect of database size distribution on resource selection algorithms. In *Distributed Multimedia Information Retrieval, SIGIR 2003 Workshop on Distributed Information Retrieval, Revised Selected and Invited Papers (LNCS 2924)*, pages 31–42, 2004.

[SE00] Atsushi Sugiura and Oren Etzioni. Query routing for web search engines: Architecture and experiments. In *Proceedings of the Ninth International World Wide Web Conference (WWW9)*, 2000.

[She95] Mark A. Sheldon. *Content Routing: A Scalable Architecture for Network-Based Information Discovery*. PhD thesis, M.I.T., 1995.

[SJCO02] Luo Si, Rong Jin, James P. Callan, and Paul Ogilvie. A language modeling framework for resource selection and results merging. In *Proceedings of the 2002 ACM Conference on Information and Knowledge Management (CIKM 2002)*, pages 391–397, 2002.

[SM83] Gerald A. Salton and Michael John McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.

[VGJL95] Ellen Marie Voorhees, Narendra Kumar Gupta, and Ben Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'95*, pages 172–179, 1995.

[VH98] Ellen Marie Voorhees and Donna Harman. Overview of the sixth Text REtrieval Conference (TREC-6). In *NIST Special Publication 500-240: The Sixth Text REtrieval Conference (TREC-6)*, pages 1–24, 1998.

[XC98] Jinxi Xu and James P. Callan. Effective retrieval with distributed collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'98*, pages 112–120, 1998.

[XC99] Jinxi Xu and William Bruce Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'99*, pages 254–261, 1999.

[YG98] Roman Yangarber and Ralph Grishman. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[YL97] Budi Yuwono and Dik Lun Lee. Server ranking for distributed text retrieval systems on the Internet. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA'97)*, pages 41–50, 1997.

[YML+99] Clement Tak Yu, Weiyi Meng, King-Lup Liu, Wensheng Wu, and Naphtali Rishe. Efficient and effective metasearch for a large number of text databases. In *Proceedings of the 1999 ACM Conference on Information and Knowledge Management (CIKM'99)*, pages 217–224, 1999.

[YMWL01] Clement Tak Yu, Weiyi Meng, Wensheng Wu, and King-Lup Liu. Efficient and effective metasearch for text databases incorporating linkages among documents. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, 2001.

[Zip49] George Kingsley Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

[ZL01] Chengxiang Zhai and John David Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2001*, pages 334–342, 2001.

[ZL02] Chengxiang Zhai and John David Lafferty. Two-stage language models for information retrieval. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2002*, pages 49–56, 2002.

[ZL04] Chengxiang Zhai and John David Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, 22(2):179–214, April 2004.

A Estimating Score Distributions

Section 4.2.3 discussed how to estimate the "uncertainty" associated with a database score for a query. Specifically, this estimate relied on the probability P of the different possible query keyword frequencies. To compute P, we assume independence of the words in the sample:

$$P = \prod_{k=1}^{n} p(d_k|s_k)$$

where $p(d_k|s_k)$ is the probability that w_k occurs in d_k documents in D given that it occurs in s_k documents in S. Using the Bayes rule, we have:

$$p(d_k|s_k) = \frac{p(s_k|d_k)p(d_k)}{\sum_{i=0}^{|D|} p(i)p(s_k|i)}$$

To compute $p(s_k|d_k)$, we assume that the presence of each word w_k follows a binomial distribution over the S documents, with $|S|$ trials and probability of success $\frac{d_k}{|D|}$ for every trial. Then,

$$p(s_k|d_k) = \binom{|S|}{s_k} \left(\frac{d_k}{|D|}\right)^{s_k} \left(1 - \frac{d_k}{|D|}\right)^{|S|-s_k}$$

$$p(d_k|s_k) = \frac{p(d_k)\left(\frac{d_k}{|D|}\right)^{s_k}\left(1-\frac{d_k}{|D|}\right)^{|S|-s_k}}{\sum_{i=0}^{|D|}\left(p(i)\left(\frac{i}{|D|}\right)^{s_k}\left(1-\frac{i}{|D|}\right)^{|S|-s_k}\right)}$$

Finally, to compute $p(d_k)$ we use the well known fact that the distribution of words in text databases tends to follow a power law [Man88]: approximately cf^{γ} words in a database have frequency f, where c and γ are database-specific constants ($c > 0, \gamma < 0$). Then,

$$p(d_k) = \frac{cd_k^{\gamma}}{\sum_{i=1}^{|D|} ci^{\gamma}} = \frac{d_k^{\gamma}}{\sum_{i=1}^{|D|} i^{\gamma}}$$

Interestingly, $\gamma = \frac{1}{B} - 1$, where B is a parameter of the frequency-rank distribution of the database [Ada02] and can be computed as described in Section 3.2.

B Estimating Score Variance

The adaptive algorithm in Figure 10 computes the mean and the variance of the query score distribution for a database to decide whether to use shrinkage for the database content summary. In Section 4.2.3, we outlined a method for computing the mean and variance relatively efficiently for any arbitrary database selection algorithm. This computation can be made even faster for the large class of database selection algorithms that assume independence of the query words. For example, bGlOSS, CORI, and LM, the database selection algorithms that we used in our experiments, belong to this class. For these algorithms, we can calculate the mean and variance of the subscore associated with each query word separately, and then combine these word-level mean and variance values to compute the final score mean and variance for the query. We show the derivation of variance[28] for bGlOSS and CORI. The computation of variance for LM is similar to the one for bGlOSS.[29]

Estimating Score Variance for bGlOSS

bGlOSS defines the score $s(q, D)$ of a database D for a query q as:

$$s(q, D) = |D| \cdot \prod_{w \in q} \hat{p}(w|D)$$

By definition of variance we have:

$$\begin{aligned} Var(s(q,D)) &= E\left[s(q,D)^2\right] - \left(E\left[s(q,D)\right]\right)^2 \\ &= E\left[\left(|D| \cdot \prod_{w\in q} p(w|D)\right)^2\right] - \left(E\left[|D| \cdot \prod_{w\in q} p(w|D)\right]\right)^2 \\ &= |D|^2 \cdot E\left[\left(\prod_{w\in q} p(w|D)\right)^2\right] - |D|^2 \cdot \left(E\left[\prod_{w\in q} p(w|D)\right]\right)^2 \end{aligned}$$

Since the $p(w|D)$'s are assumed to be independent:

$$\begin{aligned} E\left[\left(\prod_{w\in q} p(w|D)\right)^2\right] &= \prod_{w\in q} E\left[p(w|D)^2\right] \\ \left(E\left[\prod_{w\in q} p(w|D)\right]\right)^2 &= \left(\prod_{w\in q} E\left[p(w|D)\right]\right)^2 \end{aligned}$$

[28] The computation of the mean score is simpler and the derivation is analogous to the variance computation presented here.

[29] In the computation of mean and variance for LM, we treat the values of $\hat{p}(w|G)$ as constants, since the variance of the random variable $\hat{p}(w|G)$ is negligible compared to the variance of $\hat{p}(w|D)$.

Therefore:

$$Var\left(s(q,D)\right) \quad = \quad |D|^2 \cdot \left(\prod_{w \in q} E\left[p(w|D)^2\right] - \left(\prod_{w \in q} E\left[p(w|D)\right] \right)^2 \right)$$

The mean values of the distributions of $p(w|D)$ and $p(w|D)^2$ can be computed using the results from Appendix A. Since $p(w|D)$ depends only on the frequency s_w of the word w in $\hat{S}(D)$ we have:

$$\begin{aligned} E\left[p(w|D)\right] &= \sum_{i=1}^{|D|} \frac{i}{|D|} p(i|s_w) \\ E\left[p(w|D)^2\right] &= \sum_{i=1}^{|D|} \left(\frac{i}{|D|}\right)^2 p(i|s_w) \end{aligned}$$

We can see that the variance $Var\left(s(q,D)\right)$ can be computed efficiently, because there is no need to consider frequency combinations, unlike the case for a generic database selection algorithm (see Section 4.2.3).

Estimating Score Variance for CORI

CORI defines the scores(q,D) of a database D for a query q as:

$$s(q,D) = \sum_{w \in q} \frac{0.4 + 0.6 \cdot T_w \cdot I_w}{|q|} = 0.4 + 0.6 \cdot \sum_{w \in q} \frac{T_w \cdot I_w}{|q|}$$

$$T_w = \frac{p(w|D) \cdot |D|}{p(w|D) \cdot |D| + 50 + 150 \cdot \frac{cw(D)}{mcw}} \quad , \quad I_w = \log\left(\frac{m + 0.5}{cf(w)}\right) / \log\left(m + 1.0\right)$$

where $cf(w)$ is the number of databases containing w, m is the number of databases being ranked, $cw(D)$ is the number of words in D, and mcw is the mean cw among the databases being ranked. For simplicity in our calculations below, we assume that $cf(w)$ and $cw(D)$ are constants, since the variance of their values is small compared to the other components of the CORI formula. In that case, I_w is also constant. Then, by definition of variance we have:

$$\begin{aligned} Var(s(q,D)) &= E\left[\left(s(q,D)\right)^2\right] - \left(E\left[s(q,D)\right]\right)^2 \\ &= E\left[\left(0.4 + 0.6 \cdot \sum_{w \in q} \frac{T_w \cdot I_w}{|q|}\right)^2\right] - \left(E\left[0.4 + 0.6 \cdot \sum_{w \in q} \frac{T_w \cdot I_w}{|q|}\right]\right)^2 \\ &= \frac{0.36}{|q|^2} \cdot E\left[\left(\sum_{w \in q} T_w \cdot I_w\right)^2\right] - \frac{0.36}{|q|^2} \cdot \left(E\left[\sum_{w \in q} T_w \cdot I_w\right]\right)^2 \\ &= \frac{0.36}{|q|^2} \left(\sum_{w \in q} E\left[T_w^2 \cdot I_w^2\right] + \sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i} \cdot T_{w_j} \cdot \cdot I_{w_j}\right] \right. \\ &\qquad \left. - \sum_{w \in q} \left(E\left[T_w \cdot I_w\right]\right)^2 - \sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i}\right] \cdot E\left[T_{w_j} \cdot I_{w_j}\right] \right) \end{aligned}$$

By assuming independence of the words w in the query, the variables T_{w_i} and T_{w_j} are independent if $i \neq j$, and we have:

$$\sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i} \cdot T_{w_j} \cdot \cdot I_{w_j}\right] = \sum_{w_i, w_j \in q, i \neq j} E\left[T_{w_i} \cdot I_{w_i}\right] \cdot E\left[T_{w_j} \cdot I_{w_j}\right]$$

Therefore:

$$Var(s(q,D)) = \frac{0.36}{|q|^2}\left(\sum_{w\in q} I_w^2 \cdot \left(E\left[T_w^2\right] - \left(E\left[T_w\right]\right)^2\right)\right)$$

Again, the distribution of the random variables T_w and T_w^2 can be computed using the results from Appendix A. The mean values of the distributions can be computed efficiently, since there is no need to consider frequency combinations, unlike the case for a generic database selection algorithm (see Section 4.2.3).