A Probabilistic Approach to Metasearching with Adaptive Probing

Zhenyu Liu, Chang Luo, Junghoo Cho, Wesley W. Chu
Computer Science Department
University of California, Los Angeles, CA 90095
{vicliu, lc, cho, wwc}@cs.ucla.edu

Abstract

An ever-increasing amount of valuable information is stored in Web databases, "hidden" behind search interfaces. To save the user's effort in manually exploring each database, metasearchers automatically select the most relevant databases to a user's query [2, 5, 16, 21, 27, 18]. In this paper, we focus on one of the technical challenges in metasearching, namely database selection. Past research uses a pre-collected summary of each database to estimate its "relevancy" to the query, and in many cases make incorrect database selection. In this paper, we propose two techniques: probabilistic relevancy modelling and adaptive probing. First, we model the relevancy of each database to a given query as a probabilistic distribution, derived by sampling that database. Using the probabilistic model, the user can explicitly specify a desired level of certainty for database selection. The adaptive probing technique decides which and how many databases to contact in order to satisfy the user's requirement. Our experiments on real Hidden-Web databases indicate that our approach significantly improves the accuracy of database selection at the cost of a small number of database probing.

1. Introduction

An ever increasing amount of information on the Web is available through search interfaces. This information is often called the *Hidden Web* or *Deep Web* [1, 6] because traditional search engines cannot index them using existing technologies [10, 24]. The Hidden Web is estimated to be significantly larger and contain much more valuable information than the "Surface Web" that is indexed by traditional search engines [1, 6]. However, accessing the Hidden Web is often frustrating and time-consuming for average Internet users who must manually query all potentially relevant *Hidden Web databases*[1] and investigate the query results.

To help users better access the Hidden Web, recent efforts have focused on building a *metasearcher* that mediates many Hidden-Web databases and provides a single access point for the user [2, 5, 14, 15, 16, 18, 21, 25, 26, 27]. Given a user's query

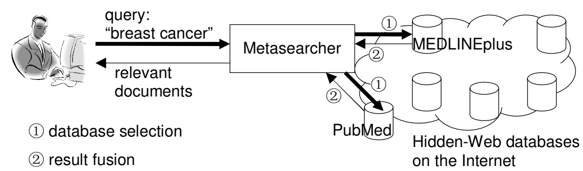


Figure 1. The metasearching process

(e.g. "breast cancer" as shown in Figure 1), the metasearcher determines which databases are the most likely to be relevant, directs the user's query to those databases and collects the search results back to the user. Given this scenario, we note that an effective metasearcher needs to accomplish two challenging tasks:

1. Based on the user's query, the metasearcher has to identify a few databases that are the most relevant, so that it can direct the query to those databases (the arrows labelled ① in Figure 1). This task is often referred to as *database selection* or *database discovery*. In Figure 1, the metasearcher directs the query "breast cancer" to two databases: "PubMed" and "MEDLINEplus.[2]"
2. The metasearcher gathers the query results from the selected databases, and selectively presents the results from multiple sources to the user (The arrows labelled ② in Figure 1). This task is also known as *result merging* or *result fusion*.

In this paper, we mainly focus on the database selection problem (task 1). Note that the ultimate goal of the metasearcher is to return a set of most relevant documents to the user. In order to achieve this goal, a good solution to database selection is essential for the *efficiency* and the *effectiveness* of the overall process: First, without database selection, the metasearcher will have to issue every query to all databases and merge the returned results. This approach cannot scale to the hundreds of thousands of Hidden-Web databases on the Internet [1, 6]. Database selection makes the metasearching process scalable by prescreening all the Hidden-Web databases and excluding the non-relevant ones, so that the metasearcher only queries a few databases in the end. Second, the accuracy of database selection directly affects the quality of the result of metasearching. Excluding a valuable data source in the selection list may cause some of the most informative documents missing in the final result.

Further, in this paper we are specially interested in selecting databases that are free-text repositories and provide keyword-search interfaces. Such free-text databases will proliferate in the

1 We call a collection of documents accessible through a Web search interface as a Hidden-Web database. PubMed (`http://www.ncbi.nlm.nih.gov/entrez/query.fcgi`) is one example.

2 `http://www.nlm.nih.gov/medlineplus/`

future because of the easy availability of free-text information (e.g. Web pages, emails, news articles, research papers, etc). Therefore, to be able to accurately locate the most relevant free-text databases for a keyword search has a great value.

To select the right set of databases given a query, we need to identify the relevancy of each database and pick the databases most relevant to the query (detailed definitions of *database relevancy* are discussed in Section 2). In previous research, the metasearcher uses a statistical summary of each database to *estimate* its relevancy to the given query [21, 26, 14, 15, 18]. For example, Gravano et al. [14, 15, 18] use (keyword, number of appearances) pairs to estimate the number of potentially relevant documents in each database. One of the main weaknesses of the existing approaches is that the estimation methods usually make strong assumptions about the database statistics, and often introduce *errors* into the relevancy estimation. As a result, the database selection fails to pick up the most relevant databases and the user ends up wasting a significant amount of time on the irrelevant ones. Recent study shows that investigating irrelevant Web pages is a major cause for users to waste time on the Web [19].

In this paper, we develop a ***probabilistic relevancy model*** in which the relevancy of each database follows a *relevancy distribution*, or *RD*. Using this relevancy distribution, we capture the error of an estimation method for each database, so that we can take this error into account when we select the final databases. As our later experiment shows, probabilistic relevancy model improves the "correctness" of database selection significantly (more than 38% in certain cases). Section 3.1 explains our relevancy model in more detail.

Further, we propose an ***adaptive probing*** technique (issuing the user query to the databases on the fly) in order to improve the "correctness" of database selection, so that the final quality of our answer may satisfy the user's anticipation. Intuitively, our adaptive probing is based on the following idea: When the metasearcher is not sure exactly what databases to select for a particular query, the metasearcher issues the query to some databases to learn their exact relevancy values. Based on this "improved knowledge" on database relevancy, the metasearcher can select databases more accurately. Since adaptive probing increases the overall processing cost of database selection, the main challenge is minimizing the number of databases to probe while maximally improving the accuracy of database selection. The answer to this challenge is presented in Section 5.

The rest of this paper is organized as follows: We review the background of the database selection problem in Section 2. In Section 3, we present our framework that incorporates both the probabilistic relevancy model and adaptive probing. Section 4 focuses on sampling a database to generate error distributions. Section 5 presents the adaptive probing technique. Our experimental results are presented in Section 6 and related works are discussed in Section 7. Section 8 concludes the paper and outlines future research.

2. Database selection background

In this section, we first present various definitions of database relevancy to a given query. We then review existing techniques that use a statistical summary to estimate a database's relevancy. We also discuss the weaknesses of such estimation-based methods, which motivate us to investigate probabilistic relevancy model and adaptive probing.

$\|db_1\|$: 20,000	
term t	$r(db_1, t)$
breast	2,000
cancer	10,000
...	...

$\|db_2\|$: 20,000	
term t	$r(db_2, t)$
breast	2,600
cancer	5,000
...	...

Figure 2. Term vs. # of appearances table.

2.1. Database relevancy

Intuitively, we consider a database relevant to a query if the database contains enough documents pertinent to the query topic. Different definitions have been proposed to formalize this notion of relevancy.

- *Document-frequency-based.* A database's relevancy is *the number of relevant documents* inside that database [21, 18]. The notion of "relevant document," however, is a subjective matter and hard for a program to decide. As a result, researchers have been using *the number of matching documents* (documents that contain all the query keywords) as a surrogate to the original definition [18]. The premise is that databases tend to have comparable fraction of relevant documents among the matching ones.
- *Document-similarity-based.* A database's relevancy is *the relevancy of the most relevant document* in that database [26]. Facing the same difficulty of measuring document relevancy, researchers usually use the query-document similarity computed by the *cosine similarity* of *tf·idf* vectors [22] as a surrogate.

The techniques that we develop in this paper can be used for both relevancy definitions. In the remainder of this paper, we refer to the ***relevancy*** of database db to query q as $r(db, q)$.

2.2. Relevancy estimation

Contacting all the Hidden-Web databases for their exact relevancy values incurs too much network traffic and processing overhead. In the past, researchers proposed to keep a local statistical summary of each database, and consult only the summary to estimate the $r(db, q)$ value for each database. Note that this estimation is typically different from the actual relevancy of the database. We refer to the ***estimated relevancy*** of database db to q as $\tilde{r}(db, q)$.

We now briefly illustrate how we may compute $\tilde{r}(db, q)$ under the document-frequency-based relevancy definition (the first item in Section 2.1). We use the following simple example to make our discussion concrete. Our techniques can also be used for other database-relevancy definitions and estimators.

In [14, 18], Gravano et al. compute $\tilde{r}(db, q)$ (the estimated number of matching documents in db) by assuming that the query terms $t_1, ..., t_m$ of q are independently distributed.

Example 1 A metasearcher is mediating two databases, db_1 and db_2, and keeps a statistical summary for each database (Figure 2). For example, the first row in db_1's table shows that the word "breast" appears in 2,000 documents of db_1. We assume that db_1 and db_2 each contain 20,000 documents. Given a user query "breast cancer," the metasearcher can estimate the number of matching documents in each database as follows: From the summary we know that $\frac{2,000}{20,000}$ of the documents in db_1 contain the word "breast" and $\frac{10,000}{20,000}$ of them contain "cancer." Then, assuming that the words "breast" and "cancer" are independently distributed, db_1 will have $20,000 \cdot \frac{2,000}{20,000} \cdot \frac{10,000}{20,000} = 1,000$ documents using both "breast" and "cancer." Thus $\tilde{r}(db_1, q)$=1,000. Similarly, $\tilde{r}(db_2, q)$=$20,000 \cdot \frac{2,600}{20,000} \cdot \frac{5,000}{20,000}$=650. □

In general, when we use this *term-independence estimator*, $\tilde{r}(db, q)$ is computed as:

$$\tilde{r}(db, q) = |db| \cdot \prod_{t_i \in q} \frac{r(db, t_i)}{|db|} \quad (1)$$

where $|db|$ is the size of db, t_i is a key term in q and $r(db, t_i)$ is the number of documents in db that use the term t_i.[3]

Note that the term-independence assumption is often incorrect and the estimated relevancy of a database can be very different from its actual relevancy. In the following section, we discuss this issue further.

2.3. Weaknesses of estimation-based database selection

Existing relevancy estimators make strong assumptions on the database statistics (like the "term-independence assumption" in Eq. 1, which is also used in [21, 26]), and often introduce errors into relevancy estimation. This error, however, may or may not affect the correctness of database selection. We illustrate this point using a scenario similar to Example 1. As in the example, we assume that the term-independence estimator predicts that the relevancy of db_1 and db_2 to the query "breast cancer" is 1000 and 650, respectively.

- *Uniform error*: When the error of the estimator is uniform (or nearly uniform) on all databases, we can select correct databases even using the erroneous estimation.

  For example, let us assume that the estimator uniformly underestimates the relevancy of db_1 and db_2 by 100% (Figure 3(a)). The actual relevancy of db_1 is 2000 while its estimated relevancy is 1000. The white bars in the figure represent the estimated relevancy and the shaded bars represent the actual relevancy.

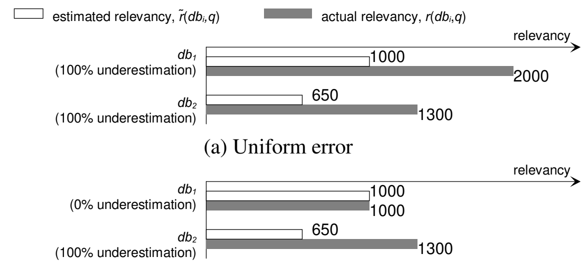
  Note that when the errors are uniform, we can still correctly select db_1 as the most relevant database based on the estimation, because db_1's estimated relevancy is still higher than that of db_2.

- *Non-uniform error*: When the errors of the estimator are not uniform among the databases, we may select a wrong database.

  For example, let us consider the scenario of Figure 3(b). The estimator has no error on db_1's relevancy, while it underestimates db_2's by 100%. Thus, we will incorrectly select db_1 as the most relevant database because db_1's estimated relevancy is higher than that of db_2.

Our experimental results show that relevancy estimators often have non-uniform errors for different databases. Because of this non-uniformity, database selection can be wrong for as many as 50% of the queries under a typical setting when we use the term-independence estimator. In the following section, we explain how we can compensate non-uniform errors using a probabilistic relevancy model.

3 Reference [18] explains in detail how we may obtain this value from Hidden-Web databases.



(a) Uniform error

(b) Non-uniform error

Figure 3. The errors of relevancy estimation

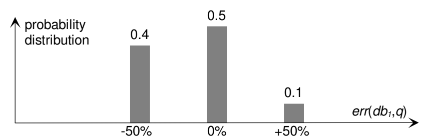


Figure 4. The ED of db_1

3. A framework for probabilistic metasearching with adaptive probing

In this section, we develop a high-level framework for the techniques that we propose. We will present the details of each technique in later sections.

3.1. Probabilistic relevancy model

In the previous section, we illustrated that non-uniform errors of a relevancy estimator may lead to incorrect database selection. Intuitively, if we can somehow predict estimation errors for each database and take the errors into account when we select the top-k databases, we may get more accurate database selection. In this subsection, we first formalize the notion of "error," and then propose a probabilistic model to capture the different errors of different databases.

The error made by a relevancy estimator on a database db_i and a query q is defined as:

$$err(db_i, q) = \frac{r(db_i, q) - \tilde{r}(db_i, q)}{\tilde{r}(db_i, q)} \quad (2)$$

For example, in Figure 3(b), the error made by the term-independence estimator on db_2 and query "breast cancer" is $\frac{1300-650}{650} = 100\%$. Note that we can compute the error value, $err(db_i, q)$, only when we know the actual relevancy $r(db_i, q)$, which is often unavailable. In the following paragraph, we explain how we can use a probabilistic distribution to predict the error without knowing the actual relevancy.

Error Distribution. Given a user query q, our goal is to use a probabilistic distribution to model the $err(db_i, q)$ made by a relevancy estimator on db_i. The basic idea is to use a small number of sample queries to observe the error distribution before we handle the user query q, and use the observed distribution to predict the errors for q. We illustrate our idea using the following example.

Example 2 We want to predict the estimation error on db_1 for the user query q. For this purpose, we sample the database db_1 with

100 sample queries (randomly chosen from, say, previous query traces). For each sample query q_s, we repeat the following procedure:

- Compute the estimated relevancy of q_s to db_1, $\tilde{r}(db_1, q_s)$ (using, say, Eq. 1).
- Obtain the actual relevancy $r(db_1, q_s)$ by issuing q_s to db_1 and analyzing the returned result.
- Compute the error $err(db_1, q_s)$ using Eq. 2.

After we have accumulated the errors for the 100 sample queries on db_1, we may observe that 40 queries have an error of -50%, 50 queries have an error of 0% and the rest 10 queries have $+50\%$. We can then summarize this into a histogram type of distribution shown in Figure 4: For 0.4 fraction of queries (40 out of 100), we get -50% error, 0.5 of queries 0% error and 0.1 of queries 50% error. We refer to this observed distribution as an *error distribution*, or *ED* of db_1.

Given this ED obtained from a random sample, we may expect that the error for a future user query q follows a similar distribution and has, say, 0.4 chance to be -50%. □

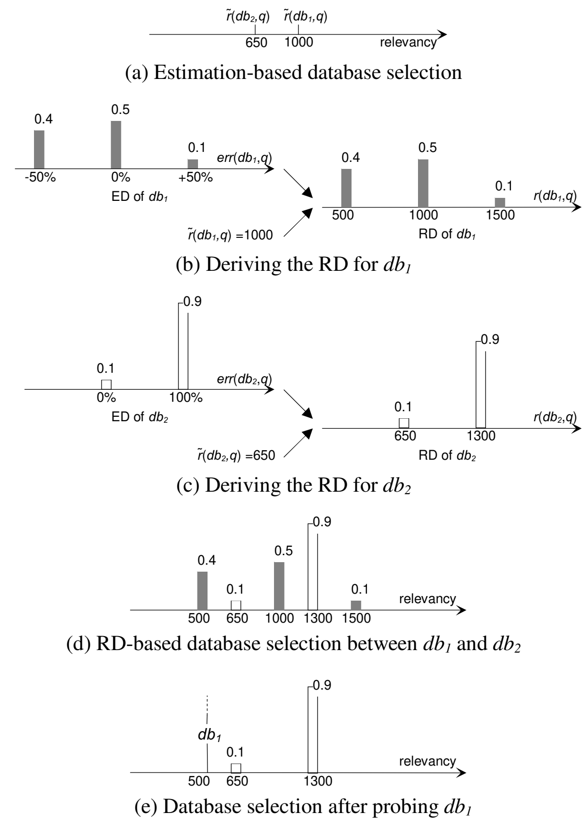
When the errors that db_1 exhibits are quite different from those of db_2, the obtained ED's on the two databases will be very different. Therefore, by obtaining a separate ED for every database db_i, we can capture non-uniform error behaviors that databases exhibit.

When we obtain an ED from random samples to model the future errors for user queries, the following questions may arise: Shall all queries exhibit similar error distribution? Wouldn't a query on topic A exhibit different error behavior from another query on topic B? If so, how can we identify different "types" of queries that exhibit different error behaviors and use a separate error distribution for each "query type?" In Section 4 we investigate this issue more carefully. Essentially, we study various criteria by which we can classify queries into different "types." We maintain a separate ED for each "query type." We assume that the error behavior of a future query, albeit unseen, is the same as the sample queries of the same type. This enables us to apply the ED learned from the sample queries in that type to the new query. Our experimental results have shown that this assumption generally holds when we focus on queries within a specific domain, e.g. medicine and health care.

In summary, our exact error modelling is as follows: Before we accept user queries, we obtain ED for each query type. When the user issues a new query q (e.g. "breast cancer"), we classify it into the appropriate query type and use the ED of that type to predict the error of the estimated relevancy $\tilde{r}(db_i, q)$. Note that the classification of query types can be database dependent, as we shall discuss in Section 4.

Relevancy Distribution. Since we eventually need to compare the relevancy of various databases for the selection purpose, we use the error distribution (ED) as a "stepping-stone" to predict the actual relevancy of those databases. More specifically, after we have obtained the error distribution of a query q on db_i, we can derive a distribution for the actual relevancy value $r(db_i, q)$, referred to as the *relevancy distribution*, or *RD*. Consider the following example.

Example 3 We have obtained the ED of q = "breast cancer" on db_1 as shown on the left side of Figure 5(b). We now want to derive the RD for db_1. To derive RD, we first estimate $\tilde{r}(db_1, q)$ using, say, the term-independence estimator. Suppose that the estimator indicates that $\tilde{r}(db_1, q)$ is 1000. From this estimation, we can derive RD as follows:



(a) Estimation-based database selection

(b) Deriving the RD for db_1

(c) Deriving the RD for db_2

(d) RD-based database selection between db_1 and db_2

(e) Database selection after probing db_1

Figure 5. RD-based database selection

From the middle bar of ED, we expect that there is a 0.5 probability to get a 0% error. Therefore, the actual relevancy $r(db_1, q)$ will be the same as $\tilde{r}(db_1, q)$ =1000 with a 0.5 probability. Similarly, we can derive that $r(db_1, q)$ has a 0.4 probability to be 500, and a 0.1 probability to be 1500, which correspond to the other bars of the RD. The derived RD is shown on the right side of Figure 5(b). □

As we will explain later, we may select the top-k databases more "correctly," using the RD's that we just obtained. To compare the effectiveness of various database selection methods, we need to define the correctness of the results returned by each method. In the next section, we first formalize the notion of correctness. We then discuss how we may use the RD's to improve the correctness.

3.2. Correctness metric for database selection

Generally speaking, the goal of database selection is to find k databases that are the most relevant to a query. Here k is a number given by the user, e.g. 1 or 3. We use $DB \equiv \{db_1, db_2, ..., db_n\}$ to represent the entire set of Hidden-Web databases, and use DB^{topk} to represent the set of k databases that are actually the most relevant. Note that the correct answer DB^{topk} is unknown to a database selection method. We refer to the set of k databases selected by a particular method as DB^k, and evaluate the effectiveness of that method by comparing the DB^k it generates with DB^{topk}. The correctness of DB^k can be defined in one of the following ways.

Symbol	Meaning
DB	$\{db_1,...,db_n\}$, the entire set of Hidden-Web databases
q	The user's query
$r(db_i, q)$	The actual relevancy of db_i for q
$\tilde{r}(db_i, q)$	The estimated relevancy of db_i for q
$err(db_i, q)$	The estimator's error on db_i for q
k	The number of databases we need to select
DB^{topk}	The actual set of top-k relevant databases
DB^k	A set of k databases selected by a metasearching method
$Cor_a(DB^k)$	Absolute correctness metric for DB^k
$Cor_p(DB^k)$	Partial correctness metric for DB^k
$E[Cor(DB^k)]$	The expected correctness of DB^k, here Cor can be Cor_a or Cor_p
t	The answer's certainty level required by the user

Figure 6. Notation used throughout the paper

- *Absolute correctness, $Cor_a(DB^k)$*: We consider DB^k is "correct" only when it contains all DB^{topk}.

$$Cor_a(DB^k) = \begin{cases} 1 & \text{if } DB^k = DB^{topk} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

- *Partial correctness, $Cor_p(DB^k)$*: We give "partial credit" to DB^k if it contains some of DB^{topk}.

$$Cor_p(DB^k) = \frac{|DB^k \cap DB^{topk}|}{k} \quad (4)$$

  For example, if an answer set DB^3 contains 2 of the 3 most relevant databases, its partial correctness is 0.67.

To help readers, the notations are summarized in Figure 6. Some of the symbols will be discussed later.

3.3. RD-based database selection

We now explain how we may use relevancy distributions (RD) to improve the correctness of database selection. The basic idea is that the RD gives us more information on how much error we may get from estimation. To illustrate, we use the examples shown in Figure 5.

Example 4 The term-independence estimator predicts that the relevancy of db_1 and db_2 to the query "breast cancer" is 650 and 1000, respectively (Figure 5(a)).

From our sampling queries, we obtained the ED's of the two databases shown on the left side of Figures 5(b) and (c). From these ED's, we can see that the term-independence estimator predicts relevancy of db_1 reasonably well (we get zero error 50% of the time), while it consistently underestimates the relevancy of db_2 (the relevancy is underestimated by 100% for 90% of the time).

Now suppose that we simply use the term-independence estimator for database selection (Figure 5(a)). In this case, we will select db_1 as the most relevant database, because $\tilde{r}(db_1, q)$ is higher than $\tilde{r}(db_2, q)$. However, note that the estimator tends to underestimate the relevancy of db_2 by 100%. Therefore, with a high probability, the relevancy of db_2 can be in fact 1300 (=650 $\cdot$ 200%) and can be larger than that of db_1 (since $\tilde{r}(db_1, q) = 1000$).

We can accommodate this fact by considering RD's of db_1 and db_2. We show the RD of db_1 on the right side of Figure 5(b) (the shaded bars) and that of db_2 in Figure 5(c) (the white bars). Figure 5(d) shows both RD's together. From these RD's, we can see that db_2 has a higher chance to be more relevant than db_1. That is, db_2 will be more relevant than db_1 in the following cases:

- $r(db_2, q) = 1300$, and $r(db_1, q) = 1000$ or $r(db_1, q) = 500$
- $r(db_2, q) = 650$ and $r(db_1, q) = 500$

The first case occurs with 0.81($= 0.9 \cdot (0.4 + 0.5)$) probability and the second case occurs with 0.04 ($= 0.1 \cdot 0.4$) probability. The sum of these probabilities is 0.85 which represents a high chance of db_2 to be more relevant. Thus we conclude that db_2 is the most relevant database (with 0.85 probability) and return it. □

We refer to the database selection based on RDs as *RD-based database selection* method. Note that the RD-based method may generate quite different results from the term-independence estimator. (In the above example, the independence estimator selects db_1 while RD-based method selects db_2.) This difference is because RD-based method takes into account the errors that the estimator typically makes on each database.

Also note that we may consider the probability computed in the above example as the "certainty level" of the RD-based method: With a 0.85 probability, the answer from RD-based method is "correct." This certainty is an indicator for the eventual correctness of our answer. For example, suppose we select the top-1 database for 100 queries each with 0.85 certainty. In the end if we check our answer against the correct answer, we can expect that for around 85 queries we have got the correct answer.

3.4. Adaptive probing

Our second technique is to use database probing to improve the certainty level that we have described in the previous subsection. Probing is an operation that issues the user's query q to database db_i and gathers the returned information in order to evaluate the *exact relevancy* of db_i to q.

We note that implementing database probing under both of the relevancy definitions (discussed in Section 2.1) is relatively easy. Under the document-frequency-based definition, for instance, many databases report the number of matching documents in their answer page. Therefore, by issuing the user query q to db_i and obtaining the number of matching documents, we can learn the exact relevancy of q to db_i. Similarly, under the document-similarity-based relevancy definition, after we issue q to db_i, we may download the top returned documents and analyze their content to compute the query-document similarity.

After we probe a database db_i, we know the exact relevancy of db_i to query q, i.e., $r(db_i, q)$. Thus, the RD for $r(db_i, q)$ changes from a regular distribution to an impulse function. For example, assuming that we probe db_1 in Figure 5(d) and get the actual relevancy as $r(db_1, q) = 500$, then the RD for db_1 becomes the impulse function shown in Figure 5(e).

Probing a few databases raises the certainty level of database selection. For example, before probing (Figure 5(d)), we only have a 0.85 certainty level to return db_2 as the top-1 database. After the probing (Figure 5(e)), however, we know that the relevancy of db_2 is always higher than db_1, and the certainty of returning db_2 becomes 1. Therefore, we have increased the certainty of our answer by 0.15.

We may consider this certainty level as a "knob" that the user can turn in order to control the database selection "quality." For the above example, for instance, when the user requires that the answer to his query should meet the certainty level of 0.7 (i.e., the answer should be correct 70% of the time), then we can simply return db_2 as the top-1 database without probing db_1. However, if the user-required certainty level is 0.9, we need to probe db_1 in order to increase our certainty level from 0.85 to above 0.9.

Given the user-required certainty level, we *adaptively* decides which and how many databases to probe in order to meet the user's requirement. Since probing can be an expensive operation during database selection, one important challenge is how we can maximally increase the certainty level with a minimal number of probing. Later in Section 5, we discuss our answer to this challenge.

4. Deriving error distribution (ED) via sampling

In the framework section, we have explained that we should separate queries into several types according to their error behaviors on a database, and sample the database to get an error distribution (ED) for each query type. This section explains the criteria that we use to identify these query types, and studies how many sampling queries should be drawn from each type.

4.1. Specialized ED's for different types of queries

In this subsection, we study how to separate queries that exhibit different error behaviors into different types. The following are a few criteria that may be used for query separation. In this part of discussion, we use the document-frequency-based relevancy definition (Section 2.1), and the term-independence estimator as our estimation method.

- **Number of terms in the query.** When estimating the number-of-matching-documents in a database, the term-independence estimator (Eq. 1) tends to make larger errors on queries that have more terms. For example, the estimator has larger errors on 3-term queries than it does on 2-term queries. Therefore the error distribution of 3-term queries differs from that of 2-term queries. Hence, we should explicitly separate queries with different number of terms.
- **Value of the initial estimation $\tilde{r}(db_i, q)$.** Intuitively, a query that is related to the topic of database db_i may exhibit quite different error behavior from a query that is not related to db_i. Note that understanding the semantics of the query and the databases is a difficult task. In our current study, we have found that the value of $\tilde{r}(db_i, q)$ (the estimated number-of-matching-documents) provides a reasonably good heuristics to decide whether the topic of query q is related to db_i. If $\tilde{r}(db_i, q)$ is below a certain threshold, the database might not cover that query topic at all, and thus the actual number-of-matching-documents $r(db_i, q)$ is typically 0. In this case, the error tends to be negative (according to Eq. 2). On the other hand, if $\tilde{r}(db_i, q)$ is above the threshold, the database might have reasonable coverage on that query topic. The actual number $r(db_i, q)$ may turn out to be much larger than $\tilde{r}(db_i, q)$, because the query terms are in fact correlated with each other in the database. In this case, the error tends to be positive. In this paper, we set this threshold as 1. That is, on database db_i, we separate queries with $\tilde{r}(db_i, q) < 1$ from those with $\tilde{r}(db_i, q) \geq 1$. Our experimental results suggest that an empirical threshold of 1 gives us reasonably good separation of different queries. We have studied other thresholds for $\tilde{r}(db_i, q)$ and include the results in [20].

  Notice that this criterion is database dependent, because a query with $\tilde{r}(db_i, q) \geq 1$ on one database db_i may have $\tilde{r}(db_j, q) < 1$ on another database db_j.

In summary, to sample a database db_i, we classify our sample queries into the following types: 2-term queries with $\tilde{r}(db_i, q) < 1$, 2-term queries with $\tilde{r}(db_i, q) \geq 1$, 3-term queries with $\tilde{r}(db_i, q) < 1$, etc. We generate a separate ED for each type of queries via sampling. When a future query arrives, we first classify that query into one of these types, and apply the corresponding ED.

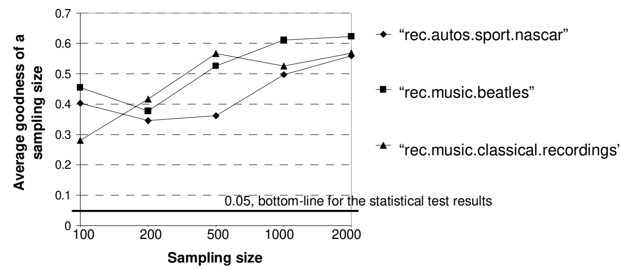


Figure 7. The average goodness of various sampling sizes on a few databases

4.2. Sampling size

After we have classify queries into different types, we sample a database with each type and generate an error distribution (ED) for that type. We want to study the following questions: First, how many samples should we draw from each query type to obtain a reasonable ED? If we decide to draw, say 500 queries from each type, how well does the ED generated on this sample size predict the ED of a future query in the same type? To answer these questions, we design the following experiment to test out various sampling sizes. The basic idea is, we first sample a database db_i using *all* the queries that we can gather in one type, and generate an "ideal" ED. Second, for a sampling size S (e.g. 500), we sample db_i with S queries to get a sample ED. Because the "ideal" ED is the best we can use for any future queries, we use a standard statistical test to compare the sample ED against the "ideal" ED. If the two are reasonably close, then the sample ED is a reasonably good approximation to the ED of a future query. The details of this experiment are as follows.

First, we construct 20 synthetic databases using newsgroup articles. We download all the articles in the 20 largest newsgroups on the UCLA news server during May, 2003, and group the articles in each newsgroup as one database. The sizes of these databases range from 28,900 articles to 81,400 articles. Note that we cannot use real Hidden-Web databases in this experiment, because issuing the entire set of queries (in hundreds of thousands) to a Hidden-Web database to get the "ideal" distribution is too expensive and time-consuming.

Second, we create a comprehensive query set from all the Web queries submitted to a search engine in one month. This set contains 4.7 million queries. For each database db_i ($1 \leq i \leq 20$), we classify these queries into the following types: 2-term queries with $\tilde{r}(db_i, q) < 1$, 2-term queries with $\tilde{r}(db_i, q) \geq 1$, etc. In the following discussion we focus on the second type, i.e., 2-term queries with $\tilde{r}(db_i, q) \geq 1$. To facilitate the discussion we refer to all the queries in this type as Q_{total}. The size of Q_{total} is typically 50,000 - 60,000 depending on db_i. We issue all the queries in Q_{total} to db_i and generate an "ideal" error distribution called ED_{total}.

Third, on the same db_i we try out five different sampling sizes, 100, 200, 500, 1000 and 2000. For one sampling size S, we randomly pick S queries from Q_{total}, issue them to db_i and generate an error distribution called ED_S. We then use the stan-

Sampling size S	100	200	500	1000	2000
Avg goodness of S	0.48	0.46	0.52	0.55	0.53

Figure 8. Average goodness of different sampling sizes

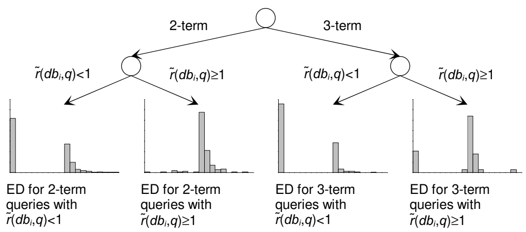


Figure 9. Separate EDs for four types of queries on a given database

dard Pearson-χ^2 test [23] (10 bins and degree of freedom as 9) to compare ED_S with ED_{total}. The test result is between 0 and 1. The closer the test result is to 0, the less likely that ED_S equals ED_{total}. Typically we *accept* the hypothesis that ED_S is the same as ED_{total} if the test result is above 0.05. Intuitively, we can consider the test result as a "goodness" measure of sampling size S. We repeat the above procedure 10 times for each sampling size S, and in the end compute an average goodness of S. Figure 7 shows the average goodness of various sampling sizes on a few databases. For example, on the particular database "rec.music.classical.recordings," the average goodness of sampling size 100 is 0.28.

Fourth, to summarize the results, we further average the goodness of each sampling size S over the 20 databases, and present the numbers in Figure 8. For example, the first column shows the average goodness of sampling size 100 is 0.48, averaged over the 20 databases.

There are two interesting observations on this result. First, for all sampling sizes, the goodness is much higher the bottom line of the statistical test (0.05). In fact, we can get reasonably good distributions using only 100 or 200 sample queries. Second, the goodness of sampling gets slightly higher when we use larger sampling sizes.

Recall that the results in Figure 8 are generated on 2-term queries with $\tilde{r}(db_i, q) \geq 1$. We have observed similar results on other query types. In our experiments, we choose to be a little bit conservative about the sampling size and use 500 queries to sample each query type. The following example summarizes our discussion in this section.

Example 5 On the synthetic database "rec.music.artists.springsteen," we generate an ED for each of the four query types: 2-term queries with $\tilde{r}(db_i, q) < 1$, 2-term queries with $\tilde{r}(db_i, q) \geq 1$, 3-term queries with $\tilde{r}(db_i, q) < 1$ and 3-term queries with $\tilde{r}(db_i, q) \geq 1$, as shown in the bottom part of Figure 9. For a new query, we use a decision tree (the top part of Figure 9) to decide which ED to use. For example, for the query "breast cancer," we first identify it as a 2-term query and narrow down to the left branch. Further, this query has an estimated relevancy less than 1 on the current database, so we will use the leftmost ED. □

5. Adaptive probing

Using the distributions we have learned in the previous section, we can select k databases DB^k for a query q with a certainty level. If this certainty is below a user-required level t, we will probe a few databases so that we can eventually return a DB^k with a certainty higher than t. This section presents the adaptive probing algorithm.

5.1. Expected correctness of DB^k

Before discussing the algorithm, we need to formally define the "certainty" of DB^k so that the algorithm can use it in the stopping condition. In this paper, we propose to use the *expected correctness* of DB^k as this certainty measure. Section 3.2 defined two correctness metrics, i.e., the absolute and the partial correctness of DB^k. In the following, we explain the expected correctness of DB^k for both metrics.

Expected absolute correctness. The absolute correctness of DB^k, $Cor_a(DB^k)$ is 1 if DB^k equals the correct answer DB^{topk}, and 0 otherwise. Thus, the expectation of $Cor_a(DB^k)$, denoted as $E[Cor_a(DB^k)]$, can be computed as:

$$\begin{aligned} & E[Cor_a(DB^k)] \\ & = 1 \cdot Pr(Cor_a(DB^k) = 1) + 0 \cdot Pr(Cor_a(DB^k) = 0) \\ & = Pr(Cor_a(DB^k) = 1) \end{aligned} \quad (5)$$

The probability $Pr(Cor_a(DB^k) = 1)$ in Eq. 5 is the probability that DB^k equals the correct answer, DB^{topk}. For example, in Figure 5(d), db_1 has a 0.85 probability to be higher than db_2 and to be the actual top-1. Hence, $Pr(Cor_a(db_1) = 1) = 0.85$ and consequently $E[Cor_a(db_1)] = 0.85$.

Generally, for any k value, $Pr(Cor_a(DB^k) = 1)$ can be computed using the relevancy distributions (RD) of all the databases. Therefore, $E[Cor_a(DB^k)]$, which equals $Pr(Cor_a(DB^k) = 1)$, is a function of all the RDs: $f(\{RD_i; i = 1, ..., n\})$. Interested readers can refer to [20] for the detailed formula of this f function.

Expected partial correctness. If DB^k has l ($0 \leq l \leq k$) databases overlapping with the correct answer DB^{topk}, then the partial correctness of DB^k, $Cor_p(DB^k)$ is $\frac{l}{k}$. Thus, the *expected partial correctness* of DB^k, $E[Cor_p(DB^k)]$ is:

$$E[Cor_p(DB^k)] \quad = \sum_{0 \leq l \leq k} \frac{l}{k} \cdot Pr(Cor_p(DB^k) = \frac{l}{k}) \quad (6)$$

For example, let us consider an answer set of two databases, DB^2. Suppose the probability that DB^2 equals DB^{top2} is 0.5, the probability that DB^2 has one database overlapping with DB^{top2} is 0.4, and the probability that DB^2 has no overlap with DB^{top2} is 0.1. Thus, $Pr(Cor_p(DB^2) = \frac{2}{2}) = 0.5$, $Pr(Cor_p(DB^2) = \frac{1}{2}) = 0.4$ and $Pr(Cor_p(DB^2) = \frac{0}{2}) = 0.1$. The $E[Cor_p(DB^2)]$ is $\frac{2}{2} \cdot 0.5 + \frac{1}{2} \cdot 0.4 + \frac{0}{2} \cdot 0.1 = 0.7$. In other words, a 0.7 fraction of DB^2 is expected to be correct.

The probability $Pr(Cor_p(DB^k) = \frac{l}{k})$ in Eq. 6 can also be computed using the RDs of all the databases. Thus, $E[Cor_p(DB^k)]$ can be another function of all the RDs: $g(\{RD_i; i = 1, ..., n\})$. Due to space limit, we will not present the complex equation here. For readers who are interested in this subject please refer to [20].

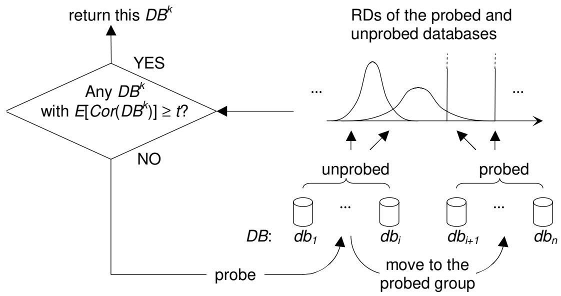


Figure 10. The adaptive probing process

Our adaptive probing algorithm can be used for both the absolute and the partial correctness metrics. Therefore, we use $E[Cor(DB^k)]$ to generally denote the expected correctness of DB^k, where the correctness metric is either absolute or partial.

5.2. Probing cost

Probing a database is costly because it incurs network traffic and query processing loads. In order to scale to many Hidden-Web databases, the adaptive probing algorithm should incur little probing cost to reach the user-required certainty level. To simplify the discussion, we assume an equal probing cost for all databases. Thus, minimizing the probing cost is the same as minimizing the total number of probing. Our methods, proposed later, can be extended to scenarios where different databases have different probing costs.

5.3. The adaptive probing algorithm

The algorithm takes four inputs: the entire set of databases DB, a query q, a number k, and the user-required certainty level t. The goal is to use a minimum number of probing to find k databases DB^k ($DB^k \subseteq DB$) with $E[Cor(DB^k)] \geq t$. Figure 10 illustrates this adaptive probing process. At any intermediate step, DB is divided into two groups: the probed group and the unprobed group. Based on the impulse RDs of the probed databases and the regular RDs of the unprobed ones, we check whether there is a DB^k with $E[Cor(DB^k)] \geq t$. If there is (the "YES" branch), the adaptive probing halts and returns this DB^k; otherwise it probes one more database (the "NO" branch), moves it from the unprobed group to the probed group, and checks the condition again.

Figure 11 provides the pseudo-code of the algorithm. In Steps (1) to (4), we first derive the RD for a query q and every database db_i based on the ED and the estimated $\tilde{r}(db_i, q)$ value. At each iteration, we check if there is a qualified DB^k that exceeds the user threshold (Step (5)). If no such DB^k exists, we pick an unprobed database (Step (6)), probe it (Step (7) and (8)) and check the condition again. When we find a qualified DB^k, we exit the **WHILE** loop and return it in the end (Step (9)).

The key issue in this algorithm is how the function *SelectDb* (Step (6)) picks the next database to probe, so that the algorithm minimizes the total number of probing. In the extended version of this paper [20], we describe the optimal policy that minimizes the total number of database probing (in the probabilistic sense). However, the optimal policy is computationally too expensive and not practical for real applications (its complexity is $O(n!)$ where n is the number of databases that the metasearcher mediates). In the following section, we describe our greedy policy that has less computational cost while selecting reasonably good databases for probing.

Algorithm 5.1 $APro(DB, q, k, t)$
Input:
DB: the entire set of databases, $\{db_1,...,db_n\}$
q: a given query
k: the number of databases to select
t: the user-required level for $E[Cor(DB^k)]$
Output:
DB^k with $E[Cor(DB^k)] \geq t$
Procedure
(1) **FOR** each $db_i \in DB$
(2) Compute $\tilde{r}(db_i, q)$
(3) Choose db_i's ED for q
(4) Derive db_i's RD from $\tilde{r}(db_i, q)$ and the ED
(5) **WHILE** $E[Cor(DB^k)] < t$ for all $DB^k \subseteq DB$
(6) $db_i \leftarrow SelectDb(DB)$
(7) Probe db_i with query q
(8) Change the RD of db_i from regular to an impulse
(9) **RETURN** the DB^k with $E[Cor(DB^k)] \geq t$

Figure 11. The adaptive probing algorithm $APro$

5.4. A greedy probing policy for $SelectDb$

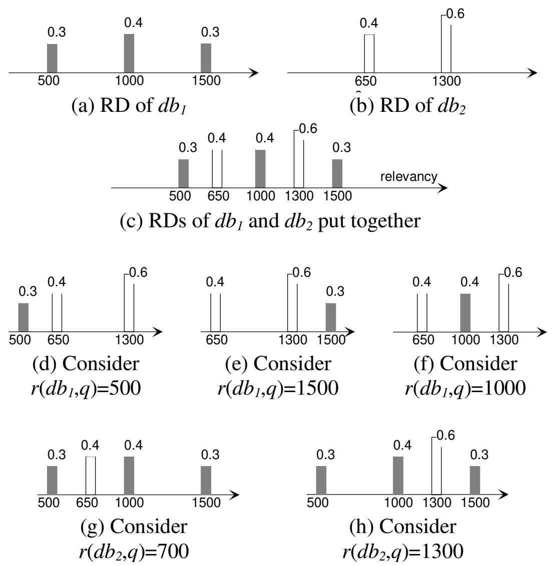
Our greedy policy is based on the following idea: The adaptive probing algorithm (Figure 11) stops as soon as the expected correctness of a DB^k, $E[Cor(DB^k)]$, exceeds the user-required threshold t. Therefore, we can make the algorithm finish early (thus reducing the probing cost) by increasing $E[Cor(DB^k)]$ most in each step. That is, we probe the database that is likely to increase $E[Cor(DB^k)]$ the most.

More precisely, we note that our algorithm stops as soon as *one* DB^k satisfies: $E[Cor(DB^k)] \geq t$. Therefore, we can only investigate the maximum $E[Cor(DB^k)]$ among all the possible $DB^k \subseteq DB$ after each probing. To help our following discussion, we refer to this maximum $E[Cor(DB^k)]$ among all the DB^k after probing db_i as the *usefulness* of probing db_i. Under this terminology, our greedy policy probes the database that yields the highest usefulness. We illustrate our greedy policy using the following example.

We need to select the top-1 database (k=1) among $\{db_1, db_2\}$. The user requires that the expected correctness of the answer to be 0.8 ($t = 0.8$). From our initial estimate of $\tilde{r}(db_i, q)$ and the ED's of the databases, we have obtained the RDs of the two databases in Figure 12(a) and (b). For clarity, we represent each RD as a simple histogram. The shaded bars represent db_1's RD and the white bars represent db_2's RD. For example, db_1's RD in Figure 12(a) indicates that $r(db_1, q)$ has a 0.3 probability to be 500, a 0.4 probability to be 1000 and a 0.3 probability to be 1500.

In order to pick the next database to probe, we need to analyze and compare the usefulness of probing db_1 and db_2. We first consider db_1. Given its RD, the probing of db_1 will lead to one of the following three outcomes:

- Case 1: $r(db_1, q) = 500$ (Figure 12(d)). When the relevancy of db_1 is 500, db_2 is definitely more relevant than db_1, be-



Example 6 **Figure 12. Picking the first database to probe between db_1 and db_2**

α	$Pr(r(db_1,q)=\alpha)$	$E[Cor(db_1)]$	$E[Cor(db_2)]$	*usefulness*
500	0.3	0	1	1
1500	0.3	1	0	1
1000	0.4	0.4	0.6	0.6

(a) Possible outcomes of probing db_1

β	$Pr(r(db_2,q)=\beta)$	$E[Cor(db_1)]$	$E[Cor(db_2)]$	*usefulness*
650	0.4	0.7	0.3	0.7
1300	0.6	0.3	0.7	0.7

(b) Possible outcomes of probing db_2

Figure 13. Considering all the outcomes of probing db_1 and db_2 under the greedy policy

cause $r(db_2, q)$ is either 650 or 1300. Therefore, the expected correctness of returning db_2 is 1 ($E[Cor(db_2)] = 1$) and the expected correctness of returning db_1 is 0 ($E[Cor(db_1)] = 0$).[4] This result is summarized in the first row of Figure 13(a). Column 2 is the probability of this case. Columns 3 and 4 show the expected correctness of returning db_1 or db_2. The maximum between column 3 and 4 is highlighted, and copied to the last column as the usefulness of probing db_1. Remember that the usefulness of probing db_1 is defined as the maximum expected correctness after probing it.

- Case 2: $r(db_1, q) = 1500$ (Figure 12(e)). When the relevancy of db_1 is 1500, we know for sure that db_1 is more relevant than db_2: $E[Cor(db_1)] = 1$ and $E[Cor(db_2)] = 0$. Thus, the usefulness is again 1 in this case. The second row of Figure 13(a) summarizes this result.
- Case 3: When $r(db_1, q) = 1000$ (Figure 12(f)), db_1 can be more relevant than db_2 with a 0.4 probability and db_2 can be more relevant than db_1 with a 0.6 probability. Thus, $E[Cor(db_1)] = 0.4$ and $E[Cor(db_2)] = 0.6$. The usefulness is 0.6 in this case. The result is summarized in the last row of Figure 13(a). □

4 When we select the top-1 database, i.e. $k = 1$, Cor_a and Cor_p are the same by definition. Thus in this example we use Cor to denote both absolute and partial correctness of a database.

Database	URL	Size
MedWeb	www.medweb.emory.edu	~14,000
PubMed Central	www.pubmedcentral.nih.gov	~60,000
NIH	www.nih.gov	163,799
Science	www.sciencemag.org	29,652

Figure 14. Sample Web databases used in our experiment

In summary, the usefulness of probing db_1 (the maximum expected correctness after probing db_1) is either 1, 1, or 0.6 with probability 0.3, 0.3, and 0.4, respectively. Thus, the *expected usefulness* of probing db_1 is $1 \cdot 0.3 + 1 \cdot 0.3 + 0.6 \cdot 0.4 = 0.84$. We can similarly analyze the expected usefulness of probing db_2 (Figure 13(b)) and get 0.7. Since the expected usefulness of probing db_1 is higher, the greedy policy picks db_1 to probe next: By probing db_1, we are likely to increase $E[Cor(DB^k)]$ more.

6. Experiments

This section reports our experimental results. Section 6.1 describes the experimental setup and the evaluation metric. Section 6.2 shows the improvement on the correctness of database selection using the relevancy distributions (RD) only, without any probing. Sections 6.3 and 6.4 show the improvement of using probing.

6.1. Experimental setup

Databases and query sets. We simulate a metasearching application that mediates 20 medical or health-related databases. First, from the "Health & Medicine" category of CompletePlanet,[5] we select 13 databases which contain at least 3000 articles. We put this empirical requirement on database size because smaller databases typically return zero documents for most of the queries, and we hypothesize that such small databases are of little interest to the user. Second, we include 4 database on broader science topics (e.g. Science and Nature), and 3 daily news websites that have constant update on health-related topics (e.g. CNN and NYTimes). We show some sample databases and their sizes[6] in Figure 14. The complete list of our databases can be found in [20].

We build our query sets using real Web query traces (provided by `inventory.overture.com`) collected during one month period. Since our experiment focuses on the "health-care" subject, we use the following procedure to ensure that our queries are also "health-care" related. We start by building a health-care vocabulary using single terms extracted from the health topic pages in MedLinePlus (`www.medlineplus.org`, an authoritative medical information website). Then from the Web query trace, we randomly pick multiple-term queries that use at least two terms from our health-care vocabulary.

To learn the distributions of the 20 databases, we prepare a query set Q_{train} with 1,000 2-term queries and 1,000 3-term

5 `http://www.completeplanet.com`

6 For databases that do not export their sizes, we roughly estimate the size by issuing a query with common terms, e.g. "medical OR health OR cancer ..."

queries using the above procedure.[7] Note that Q_{train} is a *training set* only to learn the error distributions (ED) for each database (Section 4). It is not used for performance comparison.

To evaluate the effectiveness of our proposed techniques, we prepare another query set Q_{test} as our *test set*, also with 1,000 2-term queries and 1,000 3-term queries. We compare the performance of our approach with past estimation-based approaches on this set. In addition, Q_{test} does not overlap with Q_{train}, so that we can study how well the EDs learned on the training set apply to the test set.

Evaluation metric. In our experiments, we use the document-frequency-based relevancy definition, and use the term-independence estimator (Eq. 1) to compute an initial relevancy estimation (Section 2.1). To check the correctness of a database selection method, we build the golden standard on Q_{test} as follows: For each query in Q_{test}, we issue it to the 20 databases, get the number-of-matching-documents of each database, and record the top-k databases DB^{topk} as the correct answer. We will use this golden standard throughout the rest of this section.

To evaluate the correctness of a specific database selection method $\mathcal{M}$, for every query in Q_{test}, we use $\mathcal{M}$ to select k databases DB^k. We check DB^k against the golden standard DB^{topk}, and use Eq. 3 and Eq. 4 to compute the absolute and partial correctness of DB^k. Thus, we can compute the absolute and partial correctness of $\mathcal{M}$ on each query in Q_{test}. By averaging over the 2000 queries in Q_{test}, we get the average absolute correctness $Avg(Cor_a)$ and the average partial correctness $Avg(Cor_p)$ of $\mathcal{M}$.

Term-independence estimator as the baseline. To see how much we can improve over the existing database selection methods, we apply the term-independence estimator on the Q_{test} queries and generate a baseline. The $Avg(Cor_a)$ and $Avg(Cor_p)$ of the term-independence estimator are shown in Figure 15. Note that when $k = 1$, the absolute and partial correctness are the same by definition.

For example, in the first row of Figure 15, when $k = 3$, the term-independence estimator has an $Avg(Cor_a) = 0.301$. This means in the absolute sense, the estimator correctly selects the top-3 databases for 602 ($= 0.301 \times 2000$) test queries. Also in the same row when $k = 3$, $Avg(Cor_p) = 0.699$. This means for an average query, roughly 2 ($\approx 0.699 \times 3$) out of the 3 databases selected by the estimator are in the correct answer set DB^{topk}.

6.2. Correctness improvement by the RD-based database selection

In this section, we study the effectiveness the RD-based database selection method. In Section 3.3 when we present the method, we explain that this method returns the DB^k that has the highest certainty. Since we have formalized the notion of "certainty" as the expected correctness, in the implementation of this method we select the DB^k that has the highest $E[Cor(DB^k)]$. Note that no probing is involved in this process.

We test the RD-based method on the 2000 queries of Q_{test}, compute the $Avg(Cor_a)$ and $Avg(Cor_p)$ of this method and list the results in the second row of Figure 15. For example, when $k = 1$, the $Avg(Cor_a)$ of the RD-based method is 0.651, which means the method is correct for 1302 ($= 0.651 \times 2000$) of the queries in Q_{test}. This represents a 38.2% improvement over the baseline (0.471 by the term-independence estimator). Similar improvements can be observed in other columns under different settings. These results indicate that RD effectively captures the error of the initial estimation, and thus improves the correctness of database selection.

$\mathcal{M}$	$k=1$	$k=3$	
	$Avg(Cor_a)$, $Avg(Cor_p)$	$Avg(Cor_a)$	$Avg(Cor_p)$
Term-independence estimator (baseline)	0.471	0.301	0.699
RD-based, no probing	0.651	0.478	0.815

Figure 15. RD-based database selection vs. the term-independence estimator

6.3. Correctness improvement by probing

In this experiment, we study the impact of adaptive probing on the correctness of database selection. Our goal is to see how much improvement we achieve right after 1 probing, 2 probing, etc. More specifically, for each test query, we ask the $APro$ (Figure 11) to report the DB^k with the highest $E[Cor(DB^k)]$ after *each* probing (even if the algorithm has not halted yet), and measure the correctness of the returned DB^k. We then compute the average correctness over all the queries in Q_{test}, and summarize the results in Figure 16.

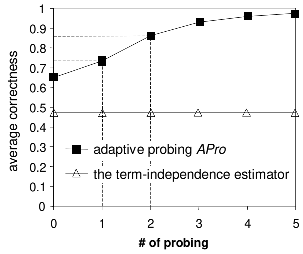
In the figure, the horizontal axis shows the number of probing that *APro* has performed so far. The vertical axis shows the average correctness over the 2000 queries. For example in Figure 16(a) when $k = 1$ (selecting the top-1 database), the average correctness of *APro* is 0.735 after one probing. This means *APro* correctly selects the top-1 database for 1470 (=0.735 $\times$ 2000) queries after only one probing. We also include the baseline produced by the term-independence estimator. Since the estimator is not affected by probing, its average correctness remains constant.

Note that at the point of no probing (# of probing = 0), *APro* is identical to the RD-based method in Section 6.2. For example, in Figure 16(a), the average correctness of zero probing is 0.651, the same as in the second row of Figure 15 ($k = 1$). In Figure 16(a), after the greedy policy has probed two databases the average correctness reaches beyond 0.80. We can observe similar increases of the average correctness in Figure 16(b) and Figure 16(c). These results have two indications. First, adaptive probing significantly improves the correctness of database selection. Second, the greedy probing policy performs reasonably well in deciding which database to probe.

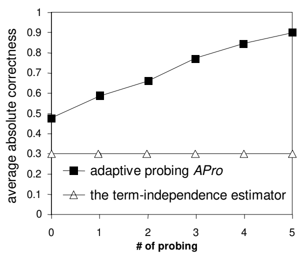
6.4. Adaptive probing under different user-required threshold t

In this subsection, we study how many probings *APro* uses to return a DB^k with $E[Cor(DB^k)] \geq t$, where t is the user-required correctness level. We experiment on six t values: {0.7, 0.75, 0.8, 0.85, 0.9, 0.95}. The number of probing increases as t increases, as shown in Figure 17. The x-axis shows the different t values. The y-axis is the number of probing used by $APro$ to reach each level of t, averaged over the 2,000 queries in Q_{test}.

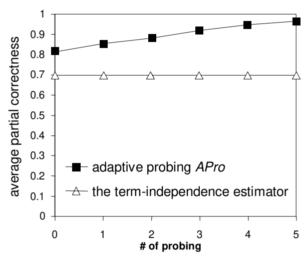
7 Note that Web queries contain 2.2 terms on average [19]. Therefore, in this paper we focus on 2 and 3-term queries, which correspond to the typical scenario of metasearching.



(a) The improvement of correctness by probing, k = 1

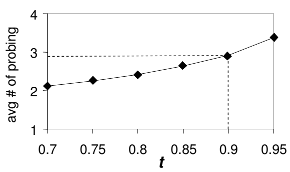


(b) The improvement of absolute correctness by probing, k = 3

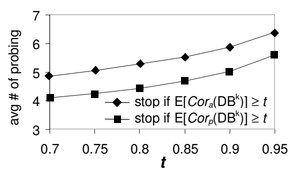


(c) The improvement of partial correctness by probing, k = 3

Figure 16. Improvement of average correctness by $APro$



(a) k = 1



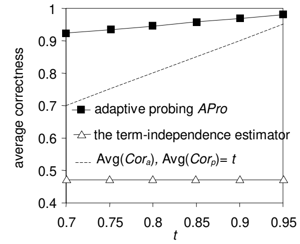
(b) k = 3

Figure 17. The average number of probing used to reach the user-required correctness level t

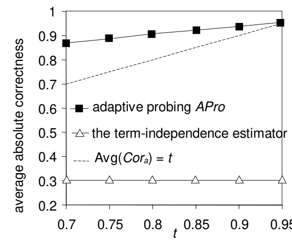
For example, when $k = 1$ (Figure 17(a)), on average *APro* uses 3 probing to reach $t = 0.9$.

In Figures 17(b) we run *APro* under two stopping conditions. The first condition requires that the expected *absolute* correctness of DB^k, $E[Cor_a(DB^k)]$ reach t for every query. The second condition requires the expected *partial* correctness to reach t for every query. Note that less probing is required under the partial correctness stopping condition, because when $k > 1$, $Cor_p(DB^k)$ is always larger than or equal to $Cor_a(DB^k)$ on any DB^k. As a result, *APro* reaches the correctness threshold faster under the partial correctness stopping condition and terminates earlier.

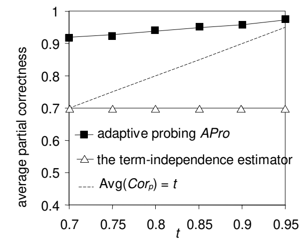
In the last set of data, we examine whether $APro$ guarantees to reach the user-required level t. Figure 18 presents the actual correctness of *APro* under different t settings. The x-axis shows the different t values, and the y-axis is the average correctness over the 2000 test queries. In Figure 18(a), for example, when we set $t = 0.7$, adaptive probing achieves an average correctness of 0.92. In the case of $k = 3$, we again run $APro$ using the two stopping conditions, $E[Cor_a(DB^k)] \geq t$ and $E[Cor_p(DB^k)] \geq t$, and show the results in Figure 18(b) and (c) respectively. The dot-



(a) k = 1



(b) k = 3, E[$Cor_a(DB^k)$] ≥ t as the stopping condition



(c) k = 3, E[$Cor_p(DB^k)$] ≥ t as the stopping condition

Figure 18. The average correctness of $APro$ vs. the user-required correctness level t

ted lines in the figures represent the user's requirement on the answers' correctness, $Avg(Cor_a) = t$ or $Avg(Cor_p = t)$. As we can see in all of the three cases, the correctness of *APro* is always higher than the dotted lines, which means the user's requirement is satisfied in all cases.

7. Related work

Database selection is a critical step in the metasearching process. Past research mainly focused on estimating how relevant a database is to the user's query. The databases with the highest estimated relevancy are selected and presented to the user. The quality of database selection is highly dependent on the accuracy of the estimation method. In the early work of bGlOSS [14] that mediates databases with boolean search interfaces, a metasearcher estimates the relevancy of each database by assuming query terms appear independently. vGlOSS [15] extends bGloss to support databases with vector-based search interfaces, and uses a *high-correlation assumption* or a *disjoint assumption* on query terms to estimate the relevancy of a database under the vector-space-model. [21] uses term covariance information to model the dependency between each pair of terms, and achieve better estimation than vGlOSS. An even better estimation is reported in [26] by incorporating document linkage information. There have been parallel research in the distributed information retrieval context. In [2, 5, 25] the relevancy of a database is modelled by the probability of the database containing similar documents to the query. In [4], various estimation methods discussed above are compared on a common basis. Our method is orthogonal to this body of research in that we are not proposing a new estimation method to improve the correctness of database selection. Instead, we use a probabilistic model to quantify the expected correctness of our answer, and use probing to increase this correctness to a user-required level.

Database selection is related to a broader research area called *top-k query answering*. Past research [11, 7, 8, 9] largely focused on relational data, and use deterministic methods to find the absolutely correct top-k answers. While in our context of Hidden-Web-database-selection, enforcing the deterministic approach would end up probing almost all the Hidden-Web databases. In our probabilistic approach, we only probe the databases that would maximally increase our certainty of the top-k answers.

Mediating heterogenous databases to provide a single query interface has been studied for years [17, 12]. While the existing research focused on integrating relational data sources, in this paper we study mediating Hidden-Web databases with unstructured textual data.

8. Conclusion and future work

In this paper, we have presented a probabilistic approach to the database selection problem together with adaptive probing. In our approach, we leverage on the existing relevancy estimation methods. We first use a probabilistic model to capture the errors of the relevancy estimation on different databases, and use the error distribution (ED) to derive the relevancy distribution (RD) of each database to a given query. Our experimental results reveal that RD compensates the error in the original relevancy estimation, and leads to more correct database selection.

Further, RD enables us to quantify the quality of our answer using the expected correctness measure. Thus, the user can explicitly control the answer's quality by specifying a desired level of expected correctness. Our adaptive probing technique will then based on the query and the user's requirement dynamically decide which and how many databases to probe. Our experimental results reveal that adaptive probing can satisfy the user's correctness requirement using a reasonably small number of probing.

In the experiments of this paper, we use the document-frequency-based relevancy definition (the first item in Section 2.1). As future work, we plan to study the effectiveness of the probabilistic relevancy model and adaptive probing on other relevancy definitions, e.g. document-similarity-based. We also plan to extend our dataset to include more Hidden-Web databases and testify our method on a larger scale.

Acknowledgements

We thank Panagiotis G. Ipeirotis and Luis Gravano at Columbia University for sharing their past research experiences in Hidden-Web database selection and commenting on the draft of this paper.

References

[1] M.K. Bergman. The Deep Web: Surfacing Hidden Value. Accessible at `www.brightplanet.com/ deepcontent/tutorials/ DeepWeb`, 2000

[2] C. Baumgarten. A Probabilistic Solution to the Selection and Fusion Problem in Distributed Information Retrieval. In *Proc. of ACM SIGIR '99*, CA, 1999

[3] J.A. Borges, I. Morales, N.J. Rodrguez. Guidelines for Designing Usable World Wide Web Pages. In *Proc. of ACM SIGCHI '96*, `http://sigchi.org/chi96/`, 1996

[4] Craswell, P. Bailey, and D. Hawking. Server Selection on the World Wide Web. In *Proc. of ACM Conf. Digital Library '00*, TX, 2000

[5] J. P. Callan, Z. Lu, and W. Croft. Searching Distributed Collections with Inference Networks. In *Proc. of ACM SIGIR '95*, WA, 1995

[6] K. Chang, B. He, C. Li, Z. Zhang. Structured Databases on the Web: Observations and Implications. Technical report, UIUCDCS-R-2003-2321, Department of Computer Science, UIUC, February 2003

[7] S. Chaudhuri and L. Gravano. Optimizing Queries over Multimedia Repositories. In *Proc. of ACM SIGMOD '96*, Canada, 1996

[8] S. Chaudhuri and L. Gravano. Evaluating Top-k Selection Queries. In *Proc. of VLDB '99*, Scotland, 1999

[9] K. Chang and S.Hwang. Minimal Probing: Supporting Expensive Predicates for Top-k Queries. In *Proc. of ACM SIGMOD '02*, WI, 2002

[10] A. Clyde. The Invisible Web. *Teacher Librarian* 29(4), `http://www.teacherlibrarian.com`, 2002

[11] R. Fagin. Combining fuzzy information from multiple systems. In *Proc. of ACM PODS '96*, Canada, 1996

[12] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *J. Intelligent Information System* 8(2):117-132, 1997

[13] M.R. Garey, D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1990

[14] L. Gravano, H. Garcia-Molina, A. Tomasic. The Effectiveness of GlOSS for the Text Database Discovery Problem. In *Proc. of SIGMOD 94*, MN, 1994

[15] L. Gravano, and H. Garcia-Molina. Generalizing GlOSS to Vector-Space databases and Broker Hierarchies. In *Proc. of VLDB '95*, Switzerland, 1995

[16] L. Gravano, H. Garcia-Molina, A. Tomasic. GlOSS: Text-Source Discovery over the Internet. *ACM TODS* 24(2):229-264, 1999

[17] A.Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal* 10(4):270-294, 2001.

[18] P.G. Ipeirotis, L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. In *Proc' of VLDB 02*, China, 2002

[19] S. Kirsch. The Future of Internet Search: Infoseek's Experiences Searching the Internect. *ACM SIGIR Forum* 32(2):3-7, 1998

[20] V.Z. Liu, R.C. Luo, J. Cho, W.W. Chu. A Probabilistic Framework for Hidden Web Database Selection Using Adaptive Probing. Technical report, Computer Science Department, UCLA, 2003

[21] W. Meng, K.L. Liu, C. Yu, et al. Determining Text Databases to Search in the Internet. In *Proc. of VLDB 98*, NY, 1998

[22] G. Salton, M.J. McGill. *Introduction to Modern Information Retrieval*, 1983

[23] D.D. Wackerly, W. Mendenhall III, R.L. Scheaffer. *Mathematical Statistics with Applications. Sixth Edition.* Duxbury Press, 2002

[24] K. Wiseman. The Invisible Web: Searching the Hidden Parts of the Web. *Learning Technology Review*, Fall 1999 / Winter 2000, `http://www.apple.com/education/LTReview/fall99`, 1999

[25] J. Xu and J. Callan. Effective Retrieval with Distributed Collections. In *Proc. of ACM SIGIR '98*, Australia, 1998

[26] C. Yu, W. Meng, W. Wu, K. Liu. Efficient and Effective Metasearch for Text Databases Incorporating Linkages among Documents. In *Proc. of SIGMOD 01*, CA, 2001.

[27] B. Yuwono, D.L. Lee. Server Ranking for Distributed Text Retrieval System on the Internet. In *Proc. of the 5th Int'l Conf. on Database System for Advanced Applications*, Australia, 1997