Relevance and Overlap Aware Text Collection Selection

Thomas Hernandez & John (Wes) Dyer & Subbarao Kambhampati*

Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85282
Paper Id: 427

Abstract

In an environment of distributed text collections, the first step in the information retrieval process is to identify which of all available collections are more relevant to a given query and should thus be accessed to answer the query. Collection selection is difficult due to the varying relevance of sources as well as the overlap between these sources. Previous collection selection methods have considered relevance of the collections but have ignored overlap among collections. They thus make the unrealistic assumption that the collections are all effectively disjoint. In this paper, we describe two new approaches: (i) COSCO which handles collection overlap, and (ii) ROSCO, which builds on COSCO to handle both collection relevance and collection overlap. We start by developing methods for estimating the statistics concerning size, relevance, and overlap that are necessary to support collection selection. We then explain how COSCO and ROSCO select text collections based upon these statistics. Finally, we demonstrate the effectiveness of COSCO and ROSCO by comparing them to major text collection selection algorithms (CORI and RDDE) under a variety of scenarios. Our evaluation is based on a set of 8 testbeds drawn from online scientific paper collections that vary systematically across relevance, overlap and size.

1 Introduction

Traditional information retrieval techniques concentrate on solving the problem of finding which documents within a source could be relevant to a user query. A slightly more complicated scenario occurs when a user wishes to query several collections simultaneously (e.g. news metasearchers and bibliography search engines). Here, in addition to the standard information retrieval issues, we have the additional challenge of deciding which collections to access. Unless the retrieval system intends to search every information source at hand – which of course would not be particularly efficient – it must decide which collection or subset of collections to call to answer a given query. This particular process is generally referred to as *collection selection*. This is particularly important because redundant or irrelevant calls are expensive in terms of query execution cost and post-query processing (i.e. duplicate removal and results merging), network load, source load, etc. Naturally, as the number of collections increase, effective collection selection becomes essential for the performance of the overall retrieval system.

The general trend in the existing approaches for collection selection essentially requires term frequency statistics about each collection in order to select the sources they deem relevant to the query (c.f. [16]). This general strategy works fairly well when the collections do not overlap (as was the case in the collections made from the TREC corpus [16]). However, many real world collections have significant overlap. For example, multiple bibliography collections (e.g. ACMDL, IEEE XPlore, DBLP etc.) may store some of the same papers, and multiple news archives (e.g. New York Times, Washington Post etc.) may store very similar news stories. Since the existing approaches fail to take into account overlap between collections when determining their collection order, they may decide to call a collection which has no new documents when considering the documents which have already been retrieved at that point in time (e.g. in the case of two mirror collections). This leads to significant loss of performance in query processing.

These problems which arise in collection selection in a distributed group of overlapping collections have not previously been addressed in the literature, as it has usually been assumed that the group of collections constitutes a perfect

*Work reported in this paper is based on an M.S. thesis by Thomas Hernandez [8, 9] and a follow-up honors thesis [6] by John Dyer. This research is supported in part by ASU Prop 301 grant ECR A601. The authors would like to thank Ullas Nambiar for the many helpful discussions throughout this research.

partition of all documents available. Our objective in this paper is to design a system able to order the collections such that when a collection is accessed, it is the collection which would provide the most new relevant results (documents). To do so, our system must be capable of making two types of predictions:

- How likely is it that a given collection has documents relevant to the query, and
- Whether a collection is useful given the ones already selected.

We present two algorithms for achieving these aims: COSCO which develops an effective way of considering coverage and overlap among collections, and ROSCO which builds on COSCO and combines the overlap with top-k relevance.

1.1 Overview of the Problem and our Solution

The problem: Formally, the general text collection selection problem can be stated as follows. Given a set S_n of n text collections with unknown and possibly overlapping document contents, a keyword query Q, and two integers c and k, pick a subset S_c of size c from S_n such that accessing the collections in S_c will result in the highest percentage recall of top-k relevant documents for the query Q. The set $\mathcal{DK}$ of top-k relevant documents is taken to be the k most relevant documents that would have been returned if the query Q was applied to a single collection that contained the union of all documents in S_n. If the collections in S_c each return the document sets D_i, then the percentage recall provided by S_c is defined as:

$$R^*_{S_c} = 100 \times \frac{|(\cup_i D_i) \cap \mathcal{DK}|}{k} \quad (1)$$

Notice that the formula considers the intersection between the union of all results D_i and the set of top-k results $\mathcal{DK}$. Thus, returning same results multiple times (as might be done when the collections are overlapping) doesn't increase the percentage recall R^*.

It should be noted that the percentage recall, as defined above, is *normative*, as it assumes complete information about the distribution of relevant documents over collections. The implemented methods need to *estimate* this information. Such estimates are often made through statistical sampling of the collections.

Various methods in the literature focus on different approximations of R^* (see Section 2). Most well known methods assume that the collections are non-overlapping (i.e. disjoint), and thus $|(\cup_i D_i)|$ is equal to $\sum_i |D_i|$. In this paper, we are interested in relaxing this assumption. In the following, we describe (i) COSCO which learns and uses query specific collection overlap statistics, and (ii) ROSCO which combines the overlap statistics with relevance statistics. In terms of our discussion above, both COSCO and ROSCO account for the of overlap between collections (i.e., they recognize that $|(\cup_i D_i)|$ may not be equal to $\sum_i |D_i|$). COSCO assumes that all documents are equally relevant (and thus $\cup_i D_i \subseteq \mathcal{DK}$), while ROSCO avoids that assumption and aims for collections that will give the most relevant documents.

Overview of COSCO: COSCO stores coverage and overlap statistics with respect to queries. Doing so ensures that when a new query comes in, the system is able to find statistics relevant to that particular query. However, since it is infeasible to keep statistics with respect to every query, we actually store them with respect to query classes. Query classes are defined in terms of frequent keyword sets, which are identified among past queries for which we have coverage and overlap statistics. Any new query could then be mapped to a set of known keyword sets. The benefit of using frequent item sets in place of exact queries is that previously unseen queries can also be mapped to some item sets.

The coverage statistics are straightforward to obtain, as they are related to the number of results returned by a collection for a specific query. That number is usually readily available from collections at query time. The overlap statistics are more challenging to estimate, and we propose to compute the overlap between two collections by looking at their respective result sets, as opposed to the individual results. This approach simplifies greatly the overlap computation and yet seems to be an effective approximation, as will be shown in this paper. Given a new query, COSCO uses its learned statistics to estimate the *coverage and residual coverage* of the individual collections, and uses these estimates to access the best sources.

Overview of ROSCO: ROSCO builds on COSCO approach to take both overlap and relevance into account. Specifically, instead of estimating coverage and residual coverage of the collections, ROSCO uses its learned statistics to estimate relevance and residual relevance (both of which are measured in terms of the number of new documents in $\mathcal{DK}$ returned by a collection). To this end, ROSCO first builds a representative of each collection via query sampling. The sampling serves three purposes. The first purpose is to provide a basis upon which to estimate the relevance of a collection with respect to a query. The second purpose is to determine the overlap between collections. The third purpose is to estimate the size of each collection. The first and second purposes are accomplished together by using random sampling whereas the third purpose specifically samples from frequent queries (using the COSCO approach). This phase constitutes the offline portion of the solution.

Evaluation: To evaluate the effectiveness of our approaches in comparison to the existing techniques, we developed

eight testbeds. These testbeds, drawn from online scientific bibliographic sources such as ACMDL, vary across three fundamental attributes: *relevance, overlap, and size*. The testbeds thus allow a collection selection method to be tested under a variety of conditions, thereby providing an understanding of the effect that these factors have on the method's performance. Using these testbeds, we present a detailed experimental evaluation of COSCO and ROSCO as well as two other major methods–CORI [3] and ReDDE [17]. Our experiments demonstrate that COSCO and ROSCO algorithms outperform existing methods in the testbeds that have overlapping collections, while being competitive in the others.

Organization: The paper is organized as follows. Existing work related to the particular problems presented above is discussed in Section 2. The COSCO approach for computing and using collection overlap statistics is presented in Section 3. The ROSCO approach for combining overlap *and* top-k relevance is described in Section 4. The experimental setup is described in Section 5, followed by the results in Section 6. Finally, we conclude in Section 7.

2 Related Work

Several approaches have been taken to solve the collection selection problem, all of which can be seem as aiming at maximizing some approximation of perecentage recall R^* as defined in Equation 1. As mentioned by Powell and French in their systematic comparison of collection selection algorithms [16], the main idea until recently has been to try to create a representative for each collection based on term and document frequency information, and then use that information at query-time to determine which collections are most promising for the incoming query. This is the case for gGlOSS [7], the CVV ranking method [22], CORI [3], SavvySearch [10], and many other approaches [20, 14, 19, 11, 13, 5]. In their survey, Powell and French [16] show that CORI is among the most effective of these approaches, but that it tends to be sensitive to the size of the collections–picking larger collections over smaller ones. A recent system called ReDDE [17] is not as susceptible to variations in size. Furthermore, ReDDE seeks to locate the top-k documents which is a stronger form of relevance based ranking. Most of these methods seek to approximate a relevance based ranking of the collections and assume that the collections are all non-overlapping. In contrast, COSCO/ROSCO approaches explicitly take collection overlaps into account. While COSCO focuses solely on overlap, ROSCO combines relevance *and* overlap.

Using coverage and overlap statistics for source selection has been explored by Nie and Kambhampati [15]. Our work, while inspired by theirs, differs in many significant ways. Their approach addresses the relational data model, in which overlap can be identified among tuples in a much more straightforward way. They use a large set of past queries with their associated coverage and overlap statistics and cluster them based on these statistics and the frequency of the training queries. Unlike in a relational environment, there are no attributes to classify queries on in a text collection environment. Finally, while relational databases use a binary notion of relevance (i.e., a tuple either is or is not an answer for a query), text collections have to use the non-binary notion of "relevance" [16]. Others have suggested using coverage information [11, 21] or overlap information [21, 19] in multi-collection scenarios, but none have actually learned and used both coverage and overlap statistics for the specific purpose of collection selection.

3 The COSCO Approach

COSCO is composed of an offline component which gathers the statistics and an online component which determines at runtime the collection ranking for a new incoming query. In the following, we describe both these components in detail.

3.1 Gathering and Computing the Statistics: the Offline Component

The purpose of the offline component in the collection selection system is to gather coverage and overlap information about collections for particular queries, and compute relevant statistics for the online component to use. More precisely, the offline component addresses three subproblems: (i) it must obtain the appropriate coverage and overlap information from the collections for a set of training queries (ii) it must then identify frequent item sets among previously asked queries to better map new queries at runtime and finally, (iii) it must compute new statistics corresponding to each of these item sets.

3.1.1 Measuring overlap and gathering statistics

As was mentioned earlier, the overlap between two collections evaluates the degree to which one collection's results are in common with another's. The ideal overlap measure would therefore capture the number of results in a collection C_a that have a similarity[1] higher than a predetermined threshold with a result in a collection C_b. Unfortunately, using thresholds implies that collection overlap is non-symmetric, in that a single result in C_a could very well be highly similar to several results in C_b. An even more problematic situation arises when considering overlap between several collections. In fact, since a document D_1 in

[1] Using a similarity measure to evaluate overlap instead of a strict identity detection mechanism is needed for this environment, as the overall purpose of the system is to avoid retrieving redundant – as opposed to identical – documents.

C_1 may be highly similar to documents D_2 in C_2 and D_3 in C_3 without D_2 being highly similar to D_3, document overlap is non-transitive. Quantifying the overlap between several collections can thus become too expensive.

To address the issues mentioned above, COSCO uses an overlap approximation which amounts to considering the set of result documents for a keyword query over a particular collection as a single document instead of a set of documents. Overlap between two collections for a particular keyword query would thus be calculated as the overlap between the union of the results of the two collections for that query. The motivation for this approach is that it is much cheaper than considering individual results for overlap computation and can still be effective enough in determining to what degree the results of two collections overlap for an individual query. Furthermore, we choose to store statistics for overlaps between pairs of collections only, as the online component will approximate the overlap between several collections using only these *pairwise* overlaps. Ignoring the actual overlap between sets of more than two collections will naturally cause some imprecisions, but, as will be shown, this approximation remains effective and much more efficient than an approach which would require overlap to be computed for all potential sets of collections.

More specifically, we first define a document to be a bag of words (i.e. a set of $(term, occurrence)$ pairs). Following the approach described above, overlap between collections C_1 and C_2 for a keyword query q is computed as the size of the intersection[2] $\mathcal{R}_{1q} \cap \mathcal{R}_{2q}$, where $\mathcal{R}_{iq}$ is the bag corresponding to the union of the top k documents for query q from collection C_i. In other words, $\mathcal{R}_{iq} = \bigcup_{j=1}^{k} results_{C_i,q}(j)$. where $results_{C_i,q}(j)$ refers to j^{th} document returned by collection C_i for keyword query q. Finally, the definition for overlap – as our approach suggests – is the following: $overlap_q(C_i, C_j) = |\mathcal{R}_{iq} \cap \mathcal{R}_{jq}|$ Notice that collection overlap as defined above is now a symmetric relation since $overlap_q(C_i, C_j) = overlap_q(C_j, C_i)$.

In addition to the overlap information, other necessary statistics include query frequency and collection coverage. Both are much easier to collect for individual queries. The query frequency simply refers to the number of times a particular query has been asked in the past, and we will define it as $freq_q$. Collection coverage for a query is the number of results a collection returns for that query. Note that once the coverage of each collection is known for a single query, the absolute coverage becomes irrelevant; instead we can consider coverage as a relative measure in terms of all results available, making a closed-world assumption. The following definition will be used for the coverage of a collection C_i with respect to a query q:

$$coverage_q(C_i) = \frac{|results_{C_i,q}|}{\sum_{j=1}^{n} |results_{C_j,q}|} \quad (2)$$

where $results_{C_i,q}$ is the set of all documents returned by collection C_i for keyword query q and n is the total number of collections being considered by the collection selection engine. Notice that the denominator $\sum_{j=1}^{n} |results_{C_j,q}|$ in Formula 2 may actually be counting some results multiple times because of the overlap between collections, but this does not affect the relative coverage measure of each collection for a particular query q since the sum would remain constant for q.

In summary, the statistics stored for each query can be considered as a vector of statistics, defined as $\overrightarrow{stats_q}$. The components of $\overrightarrow{stats_q}$ are the following:

$$\begin{bmatrix} coverage_q(C_i), & \text{for all } i \text{ from 1 to } n \\ overlap_q(C_i, C_j), & \text{for all } i, j \text{ from 1 to } n, \text{ with } i < j \\ |\mathcal{R}_{iq}|, & \text{for all } i \text{ from 1 to } n. \end{bmatrix}$$

Note that in addition to coverage and overlap statistics, we also store $|\mathcal{R}_{iq}|$ statistics. The size of the results bag $\mathcal{R}_{iq}$ is necessary and its usage will be clarified in Section 3.2.3 when describing the collection selection algorithm.

3.1.2 Identifying frequent item sets

With an overlap criterion now in hand and a statistics vector available for each training query, the next point to investigate relates to how to make use of the statistics. In fact, keeping statistics with respect to each individual query would not only be costly, but also of limited use since the statistics could only be used for the exact same query. In contrast, queries can be clustered in terms of their keywords as well as their corresponding coverage and overlap statistics with the objective of limiting the amount of statistics stored, yet keeping enough information for the online component to handle any incoming query.

Essentially, the method consists in using the Apriori algorithm [1] to discover frequently occurring keyword sets among previously asked queries. For example, the query "*data integration*" contains three item sets: {*data*}, {*integration*}, and {*data*, *integration*}. All, some, or none of these item sets may be frequent, and statistics will be stored only with respect to those which are. While keeping the number of statistics relatively low, this method also improves the odds of having some partial statistics available for new queries, as we would possibly be able to map previously unseen queries to some item sets. Using the previous example, even though the query "*data*" may not have been asked as such, the idea is to use the statistics from the query "*data integration*" – if it is frequent enough – to estimate those for "*data*". The purpose of identifying the

[2]Recall that the intersection $D_1 \cap D_2$ between two bags of words D_1 and D_2 is simply a bag containing each word and its minimum frequency across D_1 and D_2. .

frequent items sets among the queries is to avoid having to store statistics for each query, and instead store statistics with respect to frequently asked keyword sets, which are more useful for the online component, as will be explained in Section 3.2.

3.1.3 Computing statistics for frequent item sets

Once the frequent item sets are identified, statistics for each of them need to be computed. The statistics of an item set are computed by considering the statistics of all the queries that contain the item set. Let $\mathcal{Q}_{IS}$ denote the set of previously asked queries that contain the item set IS. The statistics for an item set IS are defined as the weighted average of the statistics of all the queries in $\mathcal{Q}_{IS}$, according to the following formula:

$$\overrightarrow{stats_{IS}} = \sum_{q_i \in \mathcal{Q}_{IS}} \frac{freq_{q_i}}{\sum_{q_j \in \mathcal{Q}_{IS}} freq_{q_j}} \times \overrightarrow{stats_{q_i}} \quad (3)$$

As apparent in Formula 3, the statistics of the queries are weighted by the frequency of each query, which was collected in the previous phase in addition to $\overrightarrow{stats_q}$. Using $\frac{freq_q}{\sum_{q_j \in \mathcal{Q}_{IS}} freq_{q_j}}$ as the weight ensures that the statistics for the item set would be closer to those of the most frequent queries containing the item set. The statistics should thus be more accurate more often.[3] Notice that $\overrightarrow{stats_{IS}}$ will contain estimated statistics for each of these components: $coverage_{IS}(C_i)$, $overlap_{IS}(C_i, C_j)$, and $|R_{iIS}|$.

A special case must also be dealt with when computing the statistics vectors of the frequent item sets, and that is for the *empty* item set, IS_{empty}. It is necessary to have statistics for the empty set in order to have statistics for entirely new queries (i.e. those which contain none of the frequent item sets identified by the offline component). The statistics for the empty set, $\overrightarrow{stats_{IS_{empty}}}$, are computed after having obtained all $\overrightarrow{stats_{IS}}$ vectors. $\overrightarrow{stats_{IS_{empty}}}$ is calculated by averaging the statistics of all frequent item sets. Let us denote as $item\_sets$ the set of all frequent item sets. The formula we use is then:

$$\overrightarrow{stats_{IS_{empty}}} = \frac{\sum_{IS \in item\_sets} \overrightarrow{stats_{IS}}}{|item\_sets|} \quad (4)$$

The intuition behind this formula is that the statistics for the empty set should try to reflect the general coverage and overlap information of all collections, so that a query that cannot be mapped to any stored keyword set would be assigned some average statistics which are representative of all collections. With that reasoning in mind, the statistics vector for the empty set is computed as the average of the statistics of all stored item sets.

[3]This assumes that the new queries will follow a distribution close to that of the previously asked queries.

3.2 Collection Selection at Runtime: the Online Component

The online component of the collection selection system is the component in charge of determining which is the best set of collections to call for a given user query. This requires essentially three phases. First the incoming query must be mapped to a set of item sets for which the system has statistics. Second, statistics for the query must be computed using the statistics of all mapped item sets. Finally, using these estimated query statistics, the system must determine which collections to call and in what order.

3.2.1 Mapping the query to item sets

The system needs to map the user query to a set of item sets in order to obtain some pre-computed statistics and estimate the coverage and overlap statistics for the query. More specifically, the goal is to find which group of item sets covers most, if not all, of the query. When several sets compete to cover one term, the set(s) with the most terms is(are) chosen. Consider for example the query "*data integration mining*", and suppose that only the item sets {{*data*}, {*mining*}, {*integration*}, {*data, mining*}, {*data, integration*}} are frequent. In that case, the query will be mapped to the two frequent two-term sets. Furthermore, if the item set {*data, integration, mining*} was frequent, then clearly the query would only be mapped to this three-term set.

The algorithm used to map the query to its frequent item sets is given in Algorithm 1. Practically speaking, the query

Algorithm 1 mapQuery(query Q, frequent item sets FIS) $\rightarrow IS_Q$

```
1: IS_Q ← {}
2: freqQTerms ← {}
3: for all terms t ∈ Q such that t ∈ FIS do
4:     freqQTerms ← freqQTerms ∪ t
5: IS_Q ← PowerSet(freqQTerms)
6: for all IS_i ∈ IS_Q such that IS_i ∉ FIS do
7:     Remove IS_i from IS_Q
8: for all IS_i ∈ IS_Q do
9:     if IS_i ⊂ IS_j for some IS_j ∈ IS_Q then
10:        Remove IS_i from IS_Q
11: Return IS_Q
```

q is mapped by first taking all frequent item sets that are contained in the query (lines 3 to 7). Among these selected item sets, those that are subsets of another selected item set are removed (lines 8 to 10) on the grounds that the statistics of a subset would be less accurate. The resulting set, which we call IS_q, is the set of mapped item sets for the query q.

3.2.2 Computing statistics for the query

Once the incoming user query has been mapped to a set of frequent item sets, the system computes coverage and overlap estimates by using the statistics of each mapped item set. For example, if $IS_{q_{new}} = \{\{data, integration\}, \{mining\}\}$ then the system would use the statistics of both item sets $\{$*data, integration*$\}$ and $\{$*mining*$\}$ for its statistics estimates. The query statistics for q_{new}, noted as $\overrightarrow{stats_{q_{new}}}$, are calculated by averaging each of the mapped item set statistics. When the query q_{new} was not mapped to any item set (i.e. $IS_{q_{new}} = \{\} = IS_{empty}$), then we approximate $\overrightarrow{stats_{q_{new}}}$ as being equal to $\overrightarrow{stats_{IS_{empty}}}$. In summary, we can write the following definition for $\overrightarrow{stats_{q_{new}}}$:

$$\overrightarrow{stats_{q_{new}}} = \begin{cases} \frac{\sum_{IS \in IS_{q_{new}}} \overrightarrow{stats_{IS}}}{|IS_{q_{new}}|}, & \text{if } IS_{q_{new}} \neq IS_{empty} \\ \overrightarrow{stats_{IS_{empty}}}, & \text{if } IS_{q_{new}} = IS_{empty}. \end{cases} \quad (5)$$

3.2.3 Determining the collection order

The aim of our collection selection system is to make sure that for any given k, the system would return a set of k collections which would result in the most number of distinct results of all sets of k collections. Another way to consider this is that every time a new collection (from the order suggested by our system) is called, we would like to ensure that it provides the most *new* results, taking into account the collections that have already been called. By taking into account coverage of collections with respect to item sets, our strategy would thus avoid calling collections that contain very few if any relevant documents. Moreover, by taking into account overlap among collections, it would avoid calling redundant collections which would not return any new documents.

Once the query statistics $\overrightarrow{stats_{q_{new}}}$ have been computed, the collection selection process is the following. The first collection selected is simply the one with highest coverage $coverage_{q_{new}}(C_i)$. The next collections are selected by determining which one would lead to the largest remaining result set document. More formally, the collection selection process is done according to Formula 6. At each step k, we select collection C_l such that

$$l = \begin{cases} \text{for } k = 1: \quad \underset{i}{argmax} \left[coverage_{q_{new}}(C_i) \right] \\ \text{for } k > 1: \\ \quad \underset{i}{argmax} \left[|\mathcal{R}_{iq_{new}}| - \sum_{C_j \in \mathcal{S}} overlap_{q_{new}}(C_i, C_j) \right] \end{cases} \quad (6)$$

where $\mathcal{S}$ is the set of already selected collections.

Notice that for $k > 1$, the formula is approximating the remaining result set document size by looking at pairwise overlaps only. As was explained in Section 3.1.1, we are essentially assuming that higher-order statistics (i.e. overlaps between more than two collections) are absent. This could obviously cause some inaccuracies in the statistics estimation, but as will be shown in section 6, the approximation presented here is quite effective.

4 ROSCO: Combining Relevance and Overlap

Although COSCO deals with overlap, it does not address top-k relevance–assuming instead that all documents exported by a collection are equally relevant (which amounts to the assumption that $\cup_i D_i \subseteq \mathcal{DK}$ in Equation 1). ROSCO relaxes this assumption by combining relevance and overlap estimations of the collections. It builds on COSCO by adapting and extending ideas from ReDDE [17] and

As in COSCO, ROSCO also has an offline statistics gathering component and an online query processing one. In addition to statistics about the overlap between collections (which it borrows from COSCO), ROSCO's statistics learning component also aims to build an accurate representation of the collections. To this end, it learns an accurate representation of each collection using sampling techniques [4] to help it estimate its size and relevance to a given query. Given a new query, ROSCO uses these statistics to estimate for each collection the relevance as well as the residual relevance (given other selected collections).

The offline component uses query based sampling [4] to acquire the necessary sample for building the representative. Each collection's size is estimated as well in order to normalize later computations by the collection's size. An inverted term index is built for the union of the collection samples while the source of each document in this index is noted. Finally, training queries are used to find frequently jointly occurring query terms called frequent item sets. As in COSCO, overlap statistics are then computed and stored in relation to these frequent item sets. At this point the system is ready to answer queries.

The second component of the system is the online component. When a new query arrives, ROSCO queries the centralized inverted term index. Using the results from this sample index as well as the estimated collection sizes, an estimate of the number of top-k documents in each collection is made. The collection with the largest number of top-k documents is selected first. At this point the number of top-k documents is adjusted for each remaining collection by the estimated overlap with regard to the query. The collection with the highest residual number of new top-k documents is selected next. This continues until all of the collections which are estimated to have at least one top-k

documents are selected. At this point, ROSCO loosens its notion of relevance to allow all answers to a query. It will then use the COSCO method to select the remaining collections. In the following subsections, the two components are described in greater detail.

4.1 Gathering Size and Relevance Statistics

Collection Representation through Query Based Sampling: To construct a sampled representation of each collection (for use in relevance judgements), a number of random queries are sent to each collection and a portion of the results are kept for the sample. The queries that are chosen can easily be randomly picked from the training queries. It has been shown that a relatively small number of queries is required to obtain an accurate representation of each collection [4]. Furthermore, a refinement can be made by using only the first few queries from the training data and obtaining subsequent query terms from the documents which are returned. During this exploration phase, the documents from each collection are separately stored. An inverted index is built for each collection sample to provide single source text retrieval from the sample.

Estimating Collection Size: Once a sample from each collection is available, collection size estimates are made. The sample-resample method [17] is used to estimate collection size. It is assumed that the sample is representative of the real collection. Then a query is randomly selected from the training queries. This query is sent to both the real collection and the sample. Note that the sample collection size is known and is denoted as N_{sample}. The number of results in the sample is denoted as R_{sample} whereas the number of results from the real collection is denoted as R. Let N be the size of the collection. So the probability, $P(A)$ that the real collection contains the query term is $\frac{R}{N}$ and the probability $P(B)$ that the sample contains the query term is $\frac{R_{sample}}{N_{sample}}$. Now since, $P(A) \approx P(B)$, we have $\hat{N} = \frac{R \cdot N_{sample}}{R_{sample}}$

Si and Callan showed that when the mean of several estimates is used, the absolute error ratio of the size estimate is small [17]. Note that the size of the union of all of the collections can now be estimated. It is just the sum of all of the individual estimates; however, this is not accurate in the presence of overlap. *The sample-resample method does not allow for overlap and thus requires an extension.* Let $\hat{N}$ be the sum of the estimated collection sizes, let $\hat{N}_{sample}$ be the total number of documents sampled, and let $\hat{N}'_{sample}$ be the total number of distinct documents sampled. Then the size of the union of all the collections, $\hat{N}'$, can be estimated as $\hat{N}' = \frac{\hat{N} \cdot \hat{N}'_{sample}}{\hat{N}_{sample}}$.

These estimates are stored for each collection and for the union of the collections. The estimates are used in the online component for normalization purposes.

Finally, all of the documents that have been sampled are indexed together while noting from which sources each document has been obtained. It cannot be assumed that each document came from exactly one source, as it may have been sampled from multiple sources. Furthermore, it is possible to not only consider duplicates as exact replicas but also as documents which are similar enough.

4.2 Answering Queries

When a query is posed to the ROSCO mediator, it will first use the relevance and size statistics to find the collection with the most top-k documents. Then the mediator will combine the relevance, size, and overlap estimates to find the collections with the most remaining top-k documents. This continues until the relevance estimates have been exhausted at which point the result sizes are used instead of top-k relevance estimates. The ROSCO collection selection algorithm is described in Algorithm 2 below.

Algorithm 2 $CollectionSelection(query) \rightarrow OrderedCollectionList$

```
Load Overlap and Result Size statistics for the query
Query the total sample collection
Count ← 0
for all results r in the query results in descending rank do
    r.Document.Score ← Count
    Count ← Count + mean(r.EstimatedSize/r.SampleSize)
for all collections c do
    c.Score ← 0
    for all documents d in c do
        if d.Score < Threshold then
            c.Score ← c.Score + 1
    c.Score ← (c.Score·c.EstimatedSize) / c.SampleSize
while exists a collection c with c.Score > 0 do
    Pick a collection with argmax{ResidualRelevance(Collection)}
while exists a collection c not yet selected do
    Pick a collection with argmax{ResidualCoverage(Collection)}
Return Order of Collections
```

4.2.1 Finding the Most Relevant Collections

As mentioned, ROSCO aims to estimate the relevance and residual relevance of each individual collection given a query. The (residual) relevance of a collection is defined as the fraction of top-k (new) result documents that it is expected to give. The idea of algorithm 2 is to find the collections with the highest number of remaining top-k documents first and then find the collections with the most re-

maining results. It accomplishes this by assigning each document a score equal to the number of documents which are estimated to be more relevant than itself. Each collection is then assigned a score which is the estimated number of top-k documents in the collection. Finally, those collections with the most remaining top-k documents are chosen and then it selects the rest of the collections by choosing which collection has the most remaining results. The algorithm is described in more detail below.

As in COSCO, the algorithm begins by computing all non-empty subsets of the query and finding the corresponding frequent item sets. If no frequent item sets are found then the empty set statistics are used. Otherwise, the statistics are the mean of the frequent item sets that are found. The query is then sent to the complete sample collection, which is the union of the individual collection samples. The complete sample collection returns a ranked list of documents which are relevant to the query. Next, the $Count$ is initialized to zero. This count indicates the estimated number of relevant documents encountered thus far.

After this initialization, the algorithm then iterates through all of the results with the most relevant results being visited first. The document that corresponds to the result has its score set to $Count$ which is the number of relevant documents encountered. Therefore, the score of each document is the estimated number of documents that are more relevant than it in the entire collection. To see why, note that $Count$ is incremented by the mean of the ratio of each collection's estimated size to its sample size. The collections that are included in this computation are those in which the result can be found. The mean of this ratio is the number of documents in the real union of collections that the sample result is representing.

In the next step, each collection is examined and its score is initially set to zero. Then for all the documents which are in the sample collection and have a score less than some threshold, the collection will receive one more point. The documents that contribute represent the documents which are in the top-k documents overall where k is the threshold. Finally, the collection's score is scaled by the ratio of the estimated collection size to the sample size.

At this point, each collection's score is an estimate of the number of documents in the top-k documents overall. The algorithm then proceeds to select the collection with the highest residual relevance while there exist collections with a score greater than zero. Thus all of the collections that originally were thought to contain documents in the top-k documents are selected before any of the collections thought to not contain such documents. The equation for computing residual relevance is included below.

$$ResidualRelevance(C) = C.Score \times (1 - \frac{Overlap(C)}{C.EstimatedSize}) \quad (7)$$

The overlap component is the number of documents in the collection that overlap with documents in the previously selected collections. Therefore, this essentially reduces the estimated number of relevant documents in the collection. The overlap equation is: $Overlap(C) = \sum Overlap(C, C_i)$

Each C_i is a previously selected collection and the statistics for this have been computed at the first step of the algorithm.

4.2.2 Using Overlap to Find New Results

Once all of the collections which probably contain top-k documents have been selected, then the notion of relevance is expanded to include all results instead of just top-k documents. Since this is the goal of the COSCO algorithm, ROSCO will now switch into COSCO mode where it will continue to pick the collection with the highest residual coverage until all collections have been picked. Residual coverage is computed as follows.

$$ResidualCoverage(C) = C.ResultSize - \sum Overlap(C) \quad (8)$$

Now that all of the collections have been selected then the order in which they were selected is returned. This ordering constitutes the approximated best subset of collections of size n where n is the first n collections in the list.

5 Experimental Setup

In this section, we begin by describing how performance of the various methods is measured. We then describe in detail how the testbeds were created to evaluate performance. A detailed analysis of these testbeds shows that they do indeed provide a diverse and substantial array of environments in which to test the various methods. Each of the tested methods will be described in detail as well as how the training was performed. In this section, we focus on how the testbeds are created and the performance of the algorithms is measured. The experimental results are described in the next section.

5.1 Testbed Creation

In order to do an accurate examination of the performance of the proposed solution in a variety of settings, three important factors were identified that describe various collections: (i) the variability of the size of collections (**S**ame or **V**ariable), (ii) the distribution of relevant documents (**R**andomly distributed or **C**lustered), and (iii) the presence or absence of overlap (**N**o duplicates or **D**uplicates). The first factor was chosen because it has been shown that some

methods seem to perform poorly when collection sizes differ whereas others perform well with similar sized collections [16]. The second factor is important because intuitively some collections are more relevant on some topics than others. Furthermore, relevance based methods assume that this is the case. Therefore, the effect of the distribution of relevant documents should be important to these methods. Finally, as we argued, most real world collections have overlap and it is thus necessary to understand the effect of overlap on the performance. Varying these three factors produces eight combinations. These combinations form the basis of the eight testbeds included in the experiments.

In order to form the eight testbeds, a large number of documents were required. Therefore, 38,323 abstracts were obtained from various online scientific abstract providers: ACMDL, ACM GUIDE, CSB, COMPENDEX, SCIENCE DIRECT, CITESEER, DBLP, IEEE XPLORE AND NETBIB. Each testbed has 100 collections within it. The difference between the testbeds is how they distribute the abstracts amongst the collections.

Although we performed experiments with all eight testbeds, due to space restrictions, we focus mainly on 4 of the testbeds that have overlap and one testbed which doesnt. The full set of results are available in [6].

Testbed 2 (SRD): Same Size, Random Distribution, Duplicates: This testbed was created by randomly assigning 1000 documents to each collection. The documents were picked with replacement which leads to overlap, but note that the collections are the same size and have a random distribution of relevant documents.

Testbed 4(VRD): Varying Size, Random Distribution, Duplicates: This testbed randomly picked a size for each testbed and then randomly picked the documents with replacement from the total pool of documents. The sizes were picked in an exponential fashion thereby yielding significant differences in size. note that there is overlap in this testbed. Again, the distribution of relevance is random in this testbed.

Testbed 6(SCD): Same Size, Clustered Distribution, Duplicates: This testbed used k-means clustering to create 250 clusters; however, it randomly assigned clusters with replacement to create 100 collections of nearly identical size. Therefore, there is overlap in this testbed and clustered distribution.

Testbed 7(VCN): Varying Size, Clustered Distribution, No Duplicates: To create this testbed, k-means was used to cluster the documents but only 100 clusters were created. These clusters became the 100 collections for the testbed. Thus, there is no overlap and the collection sizes vary. Also, the collections vary in relevance.

Testbed 8(VCD): Varying Size, Clustered Distribution, Duplicates: The last testbed was also created using k-means clustering with 250 clusters. This time though, each collection was randomly assigned 3 clusters with replacement. The sizes of the collections vary quite a bit and there is overlap between the collections. Also, it is a clustered distribution so relevance varies from collection to collection according to the query.

5.2 Tested Methods

In order to demonstrate the efficiency of COSCO and ROSCO, they are compared to two other well known methods CORI [3] and ReDDE [17]. In the following sections, the implementation of each method is briefly discussed.

ROSCO: The offline component of ROSCO was implemented as described previously. A large set of queries from the Bibfinder system [15] was used as the training queries. Each collection in each testbed was sampled by using 10 randomly selected training queries. The samples were used to build the representative. Next, 10 size estimates were made for each collection. The final size estimate is the mean of these estimates. Finally, queries that appeared more than 5 times were used in the frequent item set computation. A support value of .05% was required during the Apriori algorithm.

COSCO: ROSCO and COSCO require the same set of overlap statistics. So these two methods shared them for the experiments. ROSCO differs from COSCO in that it uses top-k relevance and overlap during its first phase while COSCO always considers coverage and overlap. The second phase of ROSCO is identical to COSCO. So COSCO does not use the size estimates and relevance representative.

ReDDE: ReDDE [17] and ROSCO use the same set of size and collection representation statistics so these were also shared. ROSCO contrasts with ReDDE because it considers overlap. ReDDE does not use the overlap statistics. ReDDE has no notion of residual relevance and all calculations are thus done with the assumption that every document (or document class) belongs to precisely one collection.

CORI: Until recently, CORI [3] has been widely accepted as the best, most stable method for collection selection; hence, it was included in this study. CORI models each collection as a virtual document. It can be viewed as a *df-icf* method where *df* is the document frequency of a term within a collection and *icf* is the inverse collection frequency of a term. Single source text retrieval is performed over the collection of these virtual documents to determine the order in which the collections should be called. CORI used the same sample as ROSCO and ReDDE. Once this sample was obtained then the document frequency or the number of documents containing each term in a collection was determined. Also, the collection frequency of a term was determined by finding the number of collections in the testbed which contain the term.

To establish a baseline and bounds for the performance of our systems, we have also experimented with three straw-

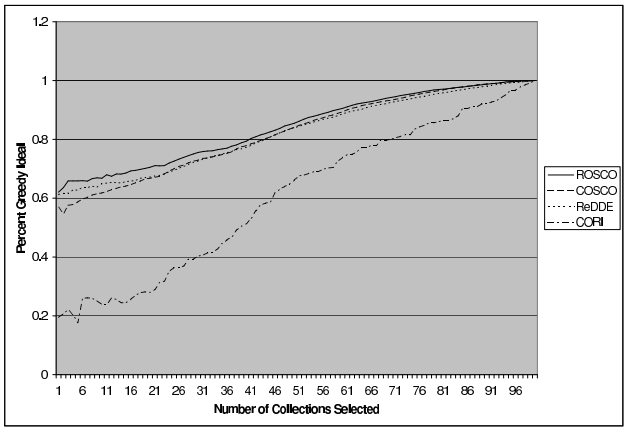

Figure 1. Percent of greedy ideal for testbed 2 (SRD)

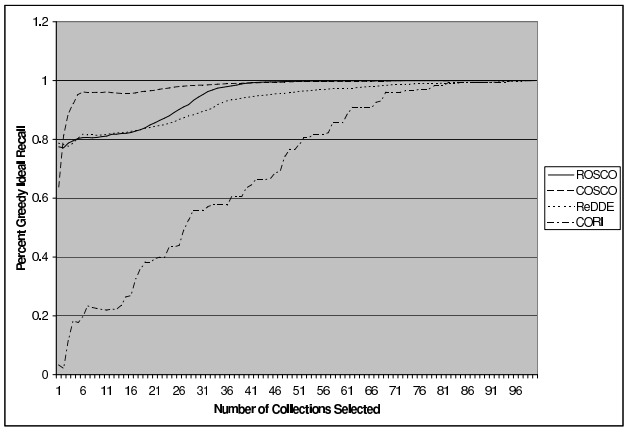

Figure 2. Percent of greedy ideal for testbed 4 (VRD)

man approaches: (i) Greedy ideal (ii) Size-based and (iii) Random.

Greedy Ideal: This method attempts to greedily maximize the percentage recall (Equation 1), assuming oracular information. Specifically, greedy ideal assumes complete knowledge of every collection and will always pick the collection with the most documents in the top-k first followed by the collection with the real highest residual relevance next and so on. It understands both higher order and lower order overlap. For the empirical study, Greedy Ideal is implemented as a "post-facto" method–which calls all the collections with the query, and analyzes their results to decide on the ideal order. The method is "greedy" in the sense that given a collection subset of size c that is greedy maximal then it will pick a subset of size $c + 1$ that is also greedy maximal and therefore it never backtracks.[4] Greedy ideal provides an upper bound on performance over the long run.

Random: This method picks collections randomly and thus provides a lower bound on long run performance. Any algorithm should outperform random in the long run. The difference between a given method and the random method shows the degree of improvement over the baseline performance.

Size Based: This method picks the collections in the order of their sizes starting with the largest (without regard to relevance or overlap). French and Powell [16] showed that most collection selection algorithms inadvertently follow a size based ranking. Size based ranking selects the largest collection first and then the next largest and so on. Therefore, including the size based ranking in the experiments allowed an examination of whether the various methods seem to follow that ranking.

[4] A non-greedy version will have to consider all possible $c + 1$-sized subsets of collections and will thus be exponential even for the post-facto analysis!

6 Experimental Results

For the experiments, 100 queries which were disjoint from the training queries were sent to each method. The percent recall at each step (collection call) was determined for every method. Then the results were averaged in order to provide a clear look at performance. Ideally, we would like to get high recall with the fewest collection calls (and thus the methods that perform better at the lower end of number of collections accessed are preferred). In addition to percentage recall, we also measured the performance of the individual methods relative to the performance of Greedy Ideal. Our hypothesis is that COSCO and ROSCO will perform better than approaches like CORI when the collections have overlap. Between COSCO and ROSCO, we would expect ROSCO to perform better as it considers both relevance and overlap.

Figures 1-6 show some of the results of our experiments. The first five compare COSCO, ROSCO, ReDDE and CORI in terms of their relative peformance with respect to Greedy Ideal in five of the eight testbeds. The last shows the percentage recall of these four methods as well as the three strawman methods (Greedy Ideal, Size-based and Random) in one of the (more realistic) testbeds. The results show that ROSCO clearly outperforms the other methods by 5% to 25% when selecting a small subset of collections .

Figure 1 shows that in the presence of overlap that CORI's performance suffers dramatically. Furthermore, in this case ROSCO outperforms all of the other methods. Figure 2 shows the performance of the methods in testbed

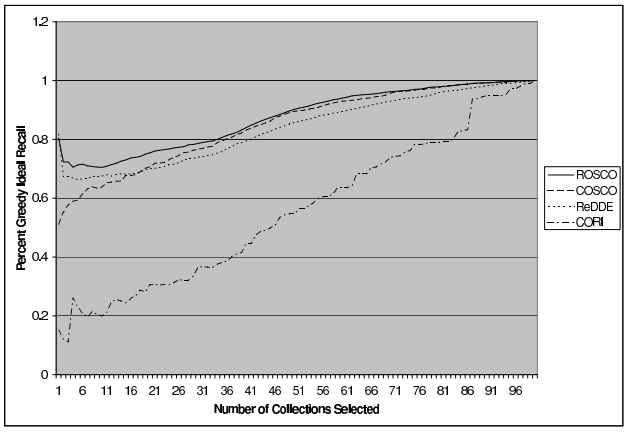

Figure 3. Percent of greedy ideal for testbed 6 (SCD)

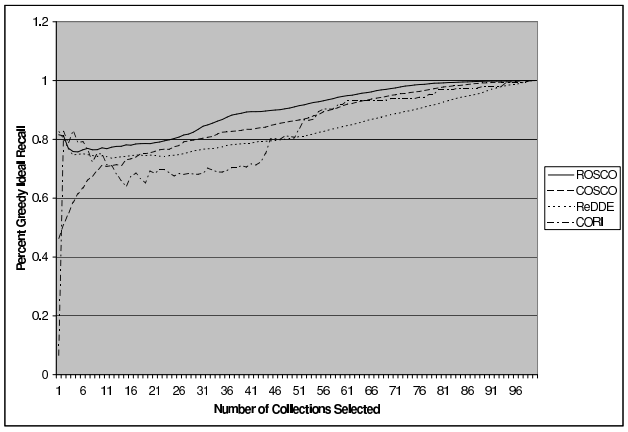

Figure 4. Percent of greedy ideal for testbed 7 (VCN)

4 where collection sizes vary and the distribution of relevant documents is *random*. This is probably not like real world scenarios since collections most likely do not have randomly assigned documents but are authoritative on certain topics. In this testbed, both ROSCO and ReDDE suffer in the beginning. COSCO performs very well but not much better than size based ranking does. Finally, CORI performs very poorly especially in the presence of overlap. Figure 3 again illustrates that ROSCO outperforms all of the other methods in testbed 6. CORI's performance degrades significantly in the presence of overlap.

Finally, testbeds 7 and 8 represent those that are most likely encountered in the real world. The collections vary in size and they have a clustered distribution. Figure 4 shows that in testbed 7 , ROSCO outperforms the other methods at every step except for a small range where CORI performs the best; but, CORI is very unstable in this testbed. Figure 5 shows that in testbed 8, ROSCO outperforms the other methods until about half of the collections have been selected when COSCO begins to outperform ROSCO. However, since collection selection aims to select a small subset of collections, it is more important to perform well early on. Notice that CORI's performance deteriorates quickly because of the presence of overlap. Finally, Figure 6 shows the relative performance of the all 7 approaches (including the 3 strawman approaches) in testbed 8. This figure provides another perspective on the way different methods compare to Greedy Ideal.

Summarizing the results over all testbeds (including the three that are not discussed here, but are included in [6]), we see that both COSCO and ROSCO perform better than CORI in all testbeds where there is overlap among collections. CORI also seems to be much less stable. As expected, ROSCO is an improvement on COSCO as well. Although ReDDE follows ROSCO closely in some testbeds, ROSCO consistently improves on it by 3%-7%. ROSCO performs the best over all the collections with the exception of testbeds 3 and 4. These testbeds however, reflect scenarios that are less likely in the real world.

7 Conclusion

This paper addressed the issue of collection selection for information retrieval in an environment composed of overlapping collections. We presented two approaches: (i) COSCO, which takes into consideration the coverage of individual collections and the overlap between collections before determining which collection should be called next and (ii) ROSCO, which builds on the COSCO system to combine overlap *and* relevance statistics to identify collections that can provide highest percentage of top-k documents (from the virtual union of collections).

In order to systematically compare these algorithms to other state-of-the-art collection selection algorithms, we created a set of eight test beds which vary across three fundamental dimensions. A detailed comparative evaluation is conducted over these testbeds among ROSCO, COSCO as well as two existing algorithms CORI and ReDDE. Our experiments showed that ROSCO and COSCO outperform existing methods in the testbeds that have overlapping collections, while being competitive in the others. ROSCO, which builds on COSCO, outperforms the latter as expected.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of VLDB Conference*, 1994.

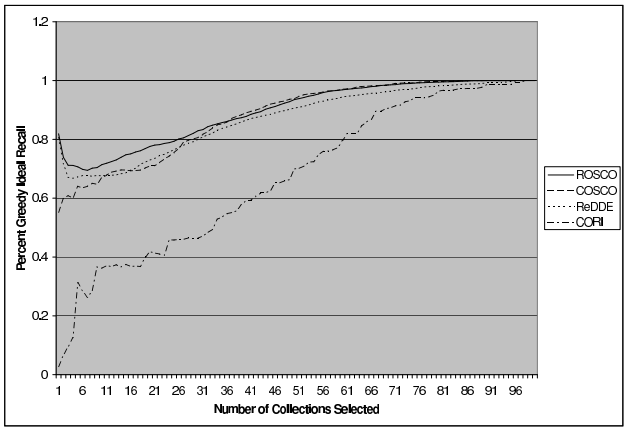

Figure 5. Percent of greedy ideal for testbed 8 (VCD)

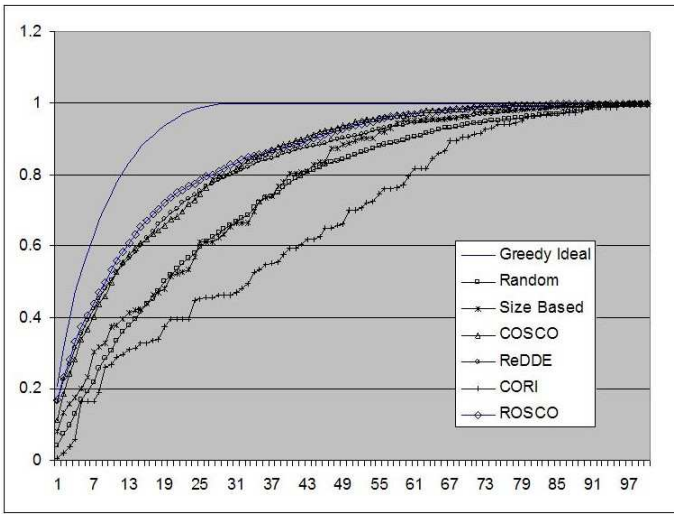

Figure 6. Percentage Recall of all methods for testbed 8 (VCD)

[2] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[3] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of ACM SIGIR Conference*, pages 21–28, 1995.

[4] Callan, J. and Connell, M. Query-based sampling of text databases. *Information Systems*, 19(2):97-130, 2001.

[5] J. G. Conrad and J. R. S. Claussen. Early user–system interaction for database selection in massive domain-specific online environments. *ACM Transactions on Information Systems*, 21(1):94–131, 2003.

[6] J. (W) Dyer. Relevance and Overlap in Text Resource Selection Honors Thesis. Dept. of CSE. Arizona State University. April 2005. rakaposhi.eas.asu.edu/wes-thesis.pdf

[7] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2):229–264, 1999.

[8] T. Hernandez. Improving text collection selection with coverage and overlap statistics. M.S. Thesis. Dept. of CSE. Arizona State University. October 2004. rakaposhi.eas.asu.edu/thomas-thesis.pdf

[9] T. Hernandez and S. Kambhampati. Improving text collection selection with coverage and overlap statistics. WWW (Special interest tracks and posters) 2005.

[10] A. E. Howe and D. Dreilinger. SAVVYSEARCH: A metasearch engine that learns which search engines to query. *AI Magazine*, 18(2):19–25, 1997.

[11] P. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of VLDB Conference*, 2002.

[12] B. J. Jansen and U. Pooch. A review of web searching studies and a framework for future research. *Journal of the American Society for Information Science and Technology*, 52(3):235–246, 2001.

[13] Z. Liu, C. Luo, J. Cho, and W. Chu. A probabilistic approach to metasearching with adaptive probing. In *Proceedings of the International Conference on Data Engineering*, 2004.

[14] W. Meng, C. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.

[15] Z. Nie and S. Kambhampati. A frequency-based approach for mining coverage statistics in data integration. In *Proceedings of the International Conference on Data Engineering*, 2004.

[16] A. L. Powell and J. C. French. Comparing the performance of collection selection algorithms. *ACM Transactions on Information Systems*, 21(4):412–456, 2003.

[17] Si, L. and Callan, J. 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*, 2003.

[18] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large AltaVista query log. Technical Report 1998-014, Digital SRC, 1998.

[19] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. The collection fusion problem. In *Text REtrieval Conference, TREC*, 1994.

[20] Z. Wu, W. Meng, C. Yu, and Z. Li. Towards a highly-scalable and effective metasearch engine. In *Proceedings of the World Wide Web Conference*, pages 386–395, 2001.

[21] R. Yerneni, F. Naumann, and H. Garcia-Molina. Maximizing coverage of mediated web queries. Technical report, Stanford University, 2000.

[22] B. Yuwono and D. L. Lee. Server ranking for distributed text retrieval systems on the internet. In *Database Systems for Advanced Applications*, pages 41–50, 1997.