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: 3012

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 ROSCO, an approach for collection selection which handles collection relevance as well as 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 ROSCO selects text collections based upon these statistics. Finally, we demonstrate the effectiveness of ROSCO by comparing it to major text collection selection algorithms (CORI and ReDDE) under a variety of scenarios.

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 meta-searchers 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* or *resource selection*. Effective solution for this problem 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.

*This research is supported in part by ASU Prop 301 grant ECR A601. The authors would like to thank Ullas Nambiar, Hemal Khatri, Bhaumik Chokshi and Yi Chen for the many helpful discussions about this research.

WWW2006, May 22–26, 2006, Edinburgh, UK.

Most existing approaches for collection selection 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 (c.f. [15]). This general strategy works fairly well when the collections do not overlap. Perhaps not surprisingly, all published evaluations of collection selection focus on disjoint collections [15]. 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.

Our objective in this paper is to design a system that accesses collections in the order of estimated relevant and *new* (previously unseen) results they are likely to provide. 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 will provide novel results given the ones already selected.

Our intent is to be able to determine, for a given user query, which collections are more relevant (i.e. which collections contain the most relevant results) and which set of collections is most likely to offer the largest variety of results (i.e. which collections are likely to have least overlap among their relevant results). Intuitively, we would want to call the most relevant collection first, and then iteratively choose the most relevant remaining collections that have least overlap with the already selected collection(s).

In this paper, we present an algorithm called ROSCO that is sensitive to both relevance and overlap among collections. Figure 1 shows the schematic architecture of ROSCO. ROSCO builds on ReDDE [16], a state of the art relevance-based collection selection algorithm. Like ReDDE [16], ROSCO uses query-based random sampling of collections, to estimate their relevance with respect to a given query. Specifically, it builds a representative of each collection via query sampling, and uses such a sampling to estimate the size of

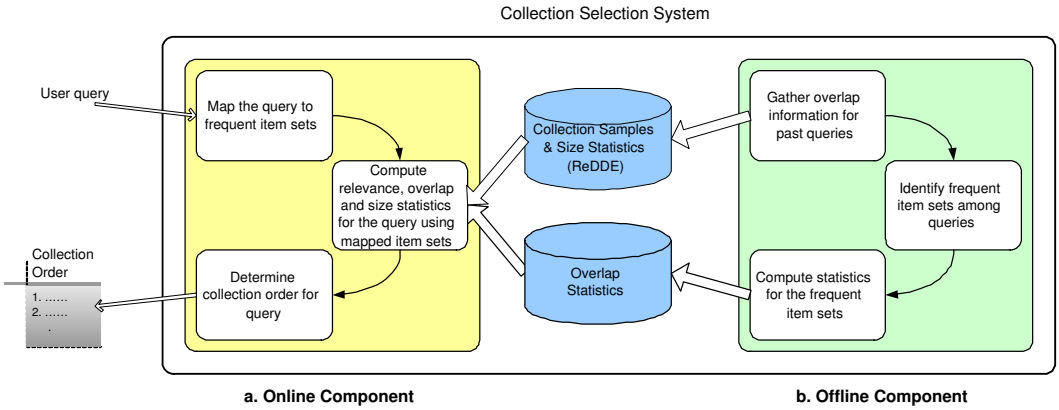

Figure 1: Architecture of ROSCO our collection selection system.

the collection and provide a basis upon which to estimate the relevance of a collection with respect to a query.

The main extension in ROSCO is that it also computes and uses overlap statistics. The challenges lie in effectively defining, efficiently computing, gathering, and then adequately using the overlap information. ROSCO stores 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, ROSCO stores them with respect to query classes instead. Query classes are defined in terms of frequent keyword sets, which are identified using itemset mining techniques [1] from 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 keyword sets in place of exact queries is that previously unseen queries can also be mapped to some item sets.

Once we have overlap statistics, the next challenge is combining them appropriately with relevance estimates. For this purpose, ROSCO uses the notion of residual relevance of a collection, given a set of collections that have already been selected. We will show how residual relevance is defined computed using the relevance estimates (*a la* ReDDE) and overlap statistics.

We present a systematic evaluation of the effectiveness of ROSCO in comparison to the existing techniques. Our evaluation is done on eight testbeds consisting of 100 text collections each. 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 ROSCO as well as two other major methods–CORI [3] and ReDDE [16]. Our experiments demonstrate that ROSCO outperforms existing methods in the testbeds that have overlapping collections, while being competitive in the others.

Organization: The rest of the paper is organized as follows. Existing work related to the particular problems presented above is discussed in Section 3. The ROSCO approach for computing and using collection overlap statistics is presented in Section 4 and Section 5. In Section 6, we discuss how the overlap statistics are integrated into the relevance-based approach used by ReDDE. The main point of departure here is that once the first collection is chosen based on relevance, overlap statistics are used to estimate *residual relevance* of the remaining collections. The experimental setup is described in Section 7, followed by the results in Section 8. Finally, we conclude in Section 9.

2. COLLECTION SELECTION: THE PROBLEM

The text collection selection problem that that we address in this paper can be stated formally 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 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. We denote this set $\mathcal{DK}$. If the collections in S_c each return the document sets D_i, then the percentage recall provided by S_c is defined as:

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

There are several points worth noting about this formulation:

First, we note that the effectiveness of collection selection approach is measured against the retrieval performance that could have been achieved if the query was processed against a single database that contained the union of all the collections.

Second, we note 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^*.

The third point to note is that the percentage recall, as defined above, is *normative*. To optimize with respect to $R*$, in practice we need to estimate statistics about the distribution of relevant documents across collections. Such estimates are

often made through statistical sampling and representations of the collections.

Various methods in the literature focus on different approximations of R^* (see Section 3). Most existing methods assume that the collections are non-overlapping (i.e. disjoint), and thus $|(\cup_i D_i)|$ is equal to $\sum_i |D_i|$, and thus focus exclusively on relevance. In this paper, we are interested in relaxing this assumption. We present ROSCO, which combines the overlap statistics with relevance statistics. In terms of our discussion above, ROSCO accounts for the overlap between collections (i.e., it recognizes that $|(\cup_i D_i)|$ may not be equal to $\sum_i |D_i|$).

3. RELATED WORK

As mentioned by Powell and French in their systematic comparison of collection selection algorithms [15], the main idea in existing systems is 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 [21], CORI [3], SavvySearch [10], and many other approaches [20, 13, 19, 11, 12, 5]. In their survey, Powell and French [15] 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 [16], is not as susceptible to variations in size. Furthermore, ReDDE seeks to select the collections that give top-k documents that would have returned by the same query had it been run on the union of all collections. This 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, ROSCO approach explicitly takes collection overlaps into account. As we mentioned, ROSCO adapts ReDDE techniques for relevance estimation, and extends them to consider collection overlap. ROSCO also builds on COSCO [9], an earlier system from our group. The main difference between ROSCOand COSCO is that the former focuses solely on overlap, while ROSCO combines relevance *and* overlap. Contemporaneous with our work, the work of Bender *et. al.* [17] also argues for using overlap statistics to collection selection. Their focus is however on peer to peer scenarios, and on efficiently disseminating the statistics between peers using bloom filters.

The work by Zhang *et. al.* [22] focuses on filtering redundant (overlapping) document results from a retrieved document stream. Our work is complementary to theirs in that we focus on the "collection selection" phase and thus can effectively reduce the number of redundant results in the retrieved document stream (thus reducing the need for filtering). Like us however, they too argue that considerations of document relevance and document redundancy (overlap) need to be independently modeled and combined for effective retrieval.

Using coverage and overlap statistics for source selection has been explored by Nie and Kambhampati [14]. 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" [15].

4. GATHERING AND STORING OVERLAP STATISTICS

Given a keyword query Q, and a set of collections $C_1, C_2 \cdots C_{i-1}$ that have already been selected for access, our aim is to estimate the amount of overlap a collection C_i is expected to have with the results already returned by $C_1 \cdots C_{i-1}$. Answering this in general requires estimates of overlap between C_i and the already accessed collections with respect to the query Q. In particular, we need pairwise overlaps of the form $ovlp(C_i, C_j)$, as well as the higher order overlaps of the form $ovlp(C_i, C_1 \cdots C_k)$. Although such overlap statistics have been used in the context of structured databases (in particular our own work in [14]), adapting such techniques to collection selection problem presents several challenges. In the following, we discuss these challenges, and our solutions.

4.1 Defining Collection Overlap

Need for Query Specific Overlap: We start by noting that the overlap between two collections needs to be characterized in the context of specific queries. It is possible for two collections to have very low overlap, when they are taken as a whole, but still have a significant overlap in terms of their results to a specific query. Consequently, in our work, we use query-specific overlap between collections. Specifically, ROSCO assumes the availability of a log of previous queries, and learns statistics with respect to them. The idea of learning statistics with respect to query logs has been shown to be effective in prior work in data integration (c.f. [14]).

Overlap in terms of duplicates vs. highly similar documents: The second issue is whether overlap between collections is defined in terms of "duplicates" or in terms of "highly similar" documents. Both types of overlap definitions are appropriate depending on the context. For example, in bibliography collections, where the same article may be present in multiple collections, overlap is best defined in terms of duplicates. In contrast, in the context of news collections, where "similar" but not identical news stories are reported by multiple news sources, overlap is best defined in terms of highly similar documents. Accordingly, our framework supports both types of overlap assessment (c.f. [8, 9]).

Assessing overlap in terms of duplicates is reasonably straightforward – we take the intersection of the set of documents returned by individual collections in response to a query (c.f. [14]).. Specifically, if $\mathcal{R}_i q$ and $\mathcal{R}_j q$ are the result sets from two collections C_i and C_j for a keyword query q, then, the overlap $ovlp_q(C_i, C_j)$ is defined as $|\mathcal{R}_i q \cap \mathcal{R}_j q|$.

It is trickier to define overlap so it takes into account highly similar rather than just duplicate documents. There are two issues here: (i) how do we determine highly similar documents and (ii) how do we avoid doing many pair-wise similarity comparisons between the results sets of two col-

lections. For the purposes of the first, the work by Zhang et. al. [22] shows that the standard document similarity metrics (such as cosine similarity) are effective. We still have to generalize this to collections. To avoid costly comparisons between all pairs of documents across collections, we use an overlap approximation which amounts to considering the set of result documents for a keyword query over a particular collection as a single document. Overlap between two collections for a particular keyword query would thus be calculated as the intersection between the bag union of the results of the two collections for that query. Specifically, if $\mathcal{R}_i q$ and $\mathcal{R}_j q$ are the result sets from two collections C_i and C_j for a keyword query q, we start by first making the union of the bags corresponding to the individual documents[1] in the result sets, $\mathcal{B}(\mathcal{R}_i q)$ and $\mathcal{B}(\mathcal{R}_j q)$. The overlap $ovlp_q(C_i, C_j)$ is defined as the bag intersection $|\mathcal{B}(\mathcal{R}_i q) \cap_B \mathcal{B}(\mathcal{R}_i q)|$[2].

ROSCO supports both the duplicate and similarity based overlap assessments, and the rest of the development in the paper is largely independendent of which overlap assessment is chosen. We should mention however that in our evaluations we used duplicate based overlap assessments, as these were most appropriate for our testbeds comprising bibliographic documents.

Pairwise vs. higher-order overlap statistics: The third issue in defining collection overlap is that of handling overlap between more than two collections with respect to a query. This will be needed when we are trying to compute how many novel results are likely to be given by a collection C_i given that we have already selected collections $C_1 \cdots C_{i-1}$. In theory, this assessment can be done if we have access to overlap statistics for every subset of collections w.r.t. the query. In practice however, such an approach will lead to storing exponential number of overlap statistics with respect to every query. Furthermore, even computing overlap over more than two collections presents technical difficulties when we are interested in similarity-based (rather than duplicate based) overlap. For these reasons, we compute store statistics for overlaps between *pairs of collections only*. The online component will approximate the overlap between several collections using only these *pairwise* overlaps.

4.2 Storing Overlap Statistics w.r.t. Query Classes

Once we decide the method of overlap assessment, for each individual past query in the query log, we store a vector of overlap statistics $\overrightarrow{ovlp_q}$ where the components of the vector are $ovlp_q(C_i, C_j)$ for all i, j from 1 to n, with $i < j$. In addition to these overlap statistics, we also keep track of the result sizes, $\mathcal{R}_i q$ for each query q and collection i.

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. A better approach (c.f. [14]) is to store statistics w.r.t. *query classes* (i.e., sets of related queries). For the case of keyword queries, query classes can also be defined in terms of keyword sets. A keyword query $k_i k_j$ corresponds to a query class which contains all the queries $k_i k_j k_1 \cdots k_l$ for all keywords $k_1 \cdots k_l$. Of particular interest are query classes that correspond to frequently occurring keyword sets among previously asked queries.

Essentially, our method consists in using the Apriori algorithm [1] to discover frequently occurring keyword sets among previously asked queries. (We assume that we have access to a query log that maintains the list of previous queries. Our evaluations are done on top of BibFinder, which maintains such a query log) 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 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 5.

4.3 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:

$$\overrightarrow{ovlp_{IS}} = \sum_{q_i \in \mathcal{Q}_{IS}} \frac{freq_{q_i}}{\sum_{q_j \in \mathcal{Q}_{IS}} freq_{q_j}} \times \overrightarrow{ovlp_{q_i}} \qquad (2)$$

As apparent in Formula 2, the statistics of the queries are weighted by the frequency of each query, which was collected in the previous phase in addition to $\overrightarrow{ovlp_q}$. Using $\frac{freq_q}{\sum_{q_j \in Q_{IS}} freq_{q_j}}$ 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]

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, $\overrightarrow{ovlp_{IS_{empty}}}$, are computed after having obtained all $\overrightarrow{ovlp_{IS}}$ vectors. $\overrightarrow{ovlp_{IS_{empty}}}$ 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:

$$\overrightarrow{ovlp_{IS_{empty}}} = \frac{\sum_{IS \in item\_sets} \overrightarrow{ovlp_{IS}}}{|item\_sets|} \qquad (3)$$

[1]The bag representation of a document consists of a set of $(term, occurrence)$ pairs).

[2]Recall that the intersection $D_1 \cap_B D_2$ 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. The union in defined analogously, with "maximum" replacing "minimum".

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

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.

5. USING OVERLAP STATISTICS AT RUNTIME

Using the overlap statistics at runtime involves two phases. First the incoming query is mapped to a set of query classes (frequent item sets) for which the system has statistics. Second, statistics for the query are computed using the statistics of all mapped item sets.

5.1 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.

5.2 Computing overlap 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 overlap 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 $\overrightarrow{ovlp_{q_{new}}}$, 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 $\overrightarrow{ovlp_{q_{new}}}$ as being equal to $\overrightarrow{ovlp_{IS_{empty}}}$. In summary, we can write the following definition for $\overrightarrow{ovlp_{q_{new}}}$:

$$\overrightarrow{ovlp_{q_{new}}} = \begin{cases} \frac{\sum_{IS \in IS_{q_{new}}} \overrightarrow{ovlp_{IS}}}{|IS_{q_{new}}|}, & \text{if } IS_{q_{new}} \neq IS_{empty} \\ \overrightarrow{ovlp_{IS_{empty}}}, & \text{if } IS_{q_{new}} = IS_{empty}. \end{cases} \quad (4)$$

6. COMBINING RELEVANCE AND OVERLAP

As mentioned earlier, we adapt the ReDDE approach for relevance estimation, and extend it to take overlap statistics. Given a new query, the ReDDE approach involves querying the collection representatives. Using the results from this sample index as well as the estimated collection sizes, an estimate of the number of relevant documents in each collection is made (see below). The collection with the largest number of relevant documents is selected first. Up to this point, ROSCO follows the ReDDE methods. It is in selecting the remaining sources that the ROSCO approach diverges. In particular, ROSCO focuses on estimating the *residual relevance* of the remaining sources, given the already selected sources. It is in this computation that the overlap statistics are used.

6.1 Gathering Size and Relevance Statistics

6.1.1 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.

6.1.2 Estimating Collection Size

Once a sample from each collection is available, collection size estimates are made. ReDDE uses the sample-resample method [16] to estimate collection size. This involves sending a query to both the collection and its (random) sample, and using the ratio of the cardinalities of the result sets, and the (known) size of the collection sample to estimate the size of the actual collection. Si and Callan showed that when the mean of several estimates is used, the absolute error ratio of the size estimate is small [16]. The sample-resample method however does not allow for overlap and thus requires an extension. So we modify their approach as follows: 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 overlapping sources.

6.2 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 and is discussed 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
```

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 new relevant documents that it is expected to give (where relevance is measured in terms of top k results returned by the union database; see Section 2. 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 remaining 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. The parameter k is set to a fraction of the total collection size. Following ReDDE, we set the fraction to be 0.003. 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.

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 overlap statistics are the mean of the statistics of the frequent item sets that are found (as described in Section 5). 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.

$$ResidualRelevance_q(C) = C.Score \times \left(1 - \frac{Overlap_q(C)}{C.EstimatedSize}\right) \quad (5)$$

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:

$$Overlap_q(C) = \sum ovlp_q(C, C_i) \quad (6)$$

Each C_i is a previously selected collection and the statistics for this have been computed as described in Section 5.

Once all of the collections which probably contain top-k documents have been selected, ROSCO can either stop (default), or continue to select additional collections by expanding its notion of relevance to include all results instead of just top-k documents. In the latter scenario, ROSCO will

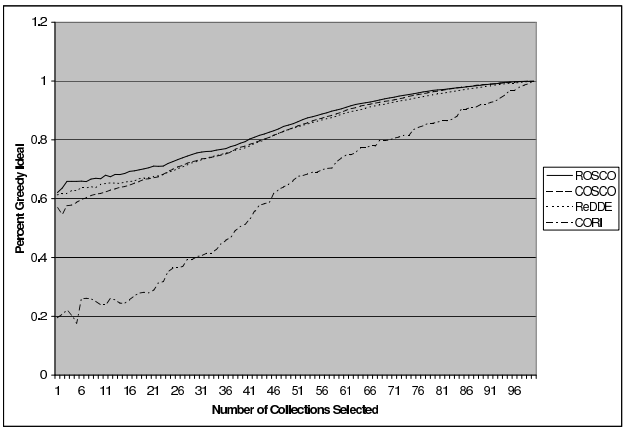

Figure 2: Results on testbed 2 (SRD)

switch into a 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:

$$ResidualCoverage_q(C) = \mathcal{R}_i q - \sum Overlap_q(C) \quad (7)$$

where $\mathcal{R}_i q$ are the result set size statistics as discussed in Section 4.2. Now that all of the collections have been selected then the order in which they were selected is returned.

7. 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.

A few words of explanation are in order with respect to the choice of the data comprising our test beds. Unlike some of the previous work on collection selection that derived their testbeds from TREC corpus, we used bibliography data. The primary reason for this was that we had ready access to user query logs from BibFinder, our own fielded bibliography mediator. Recall that our overlap statistics computation is done with respect to frequently occurring keywords in the user queries. Although bibliography data does not have pre-computed relevance judgements, this is not a problem for our evaluation. As we mentioned in Section 2, the focus in collection selection algorithms is not on the way the relevance between query and individual documents is computed, but rather on how to select collections such that the retrieval performance will be competitive with the same retrieval techniques being applied to a centralized "union" database of all collections. Accordingly, as discussed in Section 2, in our evaluation, we compare each collection selection method with respect to retrieval on the union database (using the same document retrieval strategy), in terms of the recall metric $R*$.

7.1 Testbed Creation

In order to do an accurate examination of the performance of the proposed solution in a variety of settings, we followed the example of ReDDE, and designed a set of testbeds that vary along different dimensions. The testbeds varied along three important factors: (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 two factors were considered by the testbeds used in ReDDE while the third is motivated by our interest in handling overlap across collections. 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 [15]. 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 doesn't . 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 as well as clustered distribution in this testbed.

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.

7.2 Tested Methods

The offline component of ROSCO was implemented as described previously. A large set of queries from the Bibfinder system's query log [14] was used as the training queries for overlap statistics. All the queries with a frequency greater than or equal to 4 were considered,[4] which resulted in 1,062 distinct queries and a total cumulative frequency of 19,425.

For computing the collection representative, each collection in each test bed 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. For the purposes of these experiments overlap meant duplicate documents.

In order to demonstrate the efficiency of ROSCO we compared it to two other well known methods CORI [3] and ReDDE [16]. 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. ReDDE [16] 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 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.

Finally, to evaluate the relative effects of handling relevance and overlap considerations in collection selection, we also experimented with COSCO [9] that differs from ROSCO in that it focuses only on overlap statistics. Notice that the comparison between ROSCO and ReDDE allows us to evaluate the effect of overlap analysis alone (since ReDDE is can be seen as ROSCO without overlap analysis!).

To establish a baseline and bounds for the performance of our systems, we have also experimented with three strawman approaches: (i) Greedy ideal (ii) Size-based and (iii) Random.

[4]Since the queries from the BibFinder query-list were relational in nature, each query selected was transformed into a keyword query by simply merging all the fields from the relational query.

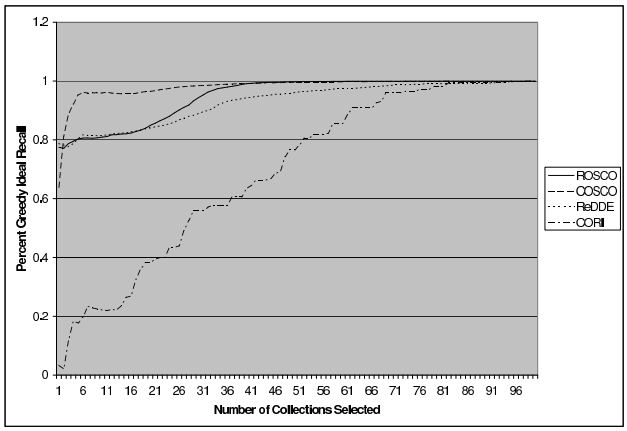

Figure 3: Results on testbed 4 (VRD)

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 pair-wise and higher 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.[5] 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). We included this because French and Powell [15] showed that most collection selection algorithms inadvertently follow a size based ranking.

8. EXPERIMENTAL RESULTS

For the experiments, 100 queries which were disjoint from the training queries were sent to each method. The percent recall $R*$ at each step (collection call) was determined for every method. Specifically, this involves comparing the results to the top-k results retrieved by running the same query on a database corresponding to the union of all the collections (see Section 2). We set the threshold parameter in Algorithm 2 such that we focus on top 100 results (i.e. $k = 100$). We keep the "document retrieval" method constant across all collections as well as the union database (we used the standard TF/IDF based cosine similarity [16]). The results

[5]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!

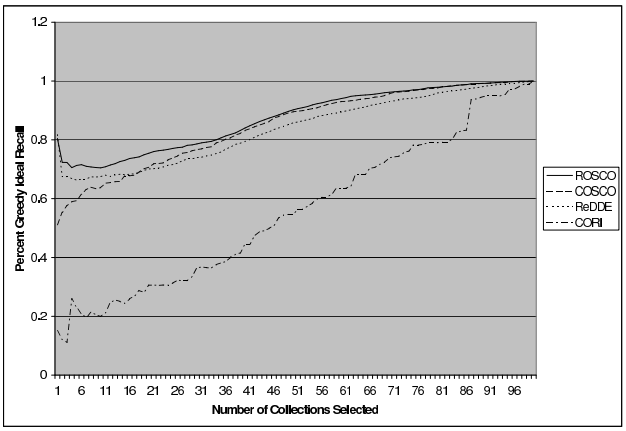

Figure 4: Results on testbed 6 (SCD)

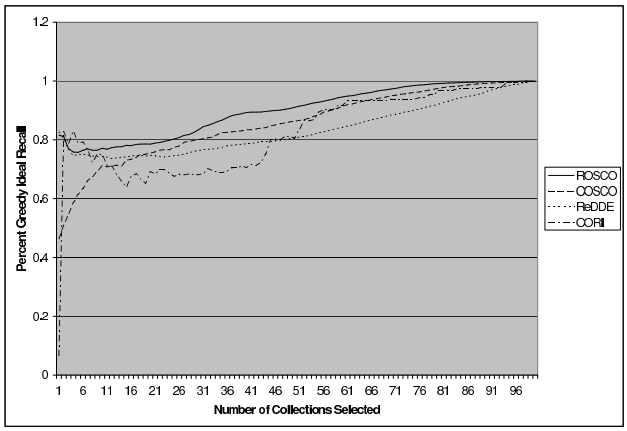

Figure 5: Results on testbed 7 (VCN)

are averaged over all the test queries 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 ROSCO will perform better than approaches like CORI when the collections have overlap. Between ROSCO and COSCO we would expect ROSCO to perform better as it considers both relevance and overlap.

Figures 3-8 show some of the results of our experiments. The first five compare ROSCO with COSCO, ReDDE and CORI in terms of their relative performance 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 2 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 3 shows the performance of the methods in testbed 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. ROSCO performs very well but not much better than size based ranking. Finally, CORI performs very poorly especially in the presence of overlap. Figure 4 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 5 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 6 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 7 shows the relative performance of all 7 approaches (including the 3 strawman approaches) in testbed 8. This figure focuses on the performance in the initial stages (upto the time a third of the 100 collections have been accessed), and provides another perspective on the way different methods compare to Greedy Ideal.

8.1 Summary and Discussion of Results

Summarizing the results over all testbeds (including the three that are not discussed here, but are included in [6]), we see that ROSCO performs 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.

ReDDE follows closely because ROSCO and ReDDE will always select the same first collection. In the testbeds that we set up, this meant that after the first selection both ROSCO and ReDDE will have selected a collection with roughly 70-85% of the greedy ideal. This percentage generally (in most testbeds) increases monotonically. The difference between ROSCO and ReDDE after this point is the effect that overlap has on ReDDE. Over time ROSCO gains on ReDDE in all test beds. However, in testbed 4 both ROSCO and ReDDE suffer (although the percentages skew the view of the actual impact). In this case selecting a slightly less than optimal choice up front has a huge impact.

A final word of explanation is in order regarding the performance of CORI in our experiments. The testbeds that we created vary three attributes (relevance, size, and overlap). CORI explicitly assumes no overlap and implicitly assumes roughly equivalent sizes (both [16] and [15] acknowledge this). So in all test beds except SRN (testbed 1) and SCN (testbed 5), CORI winds up having a big disadvantage. Our results are consistent with Si and Callan's own comparison between ReDDE and CORI (see Figure 1 in [16]). In the

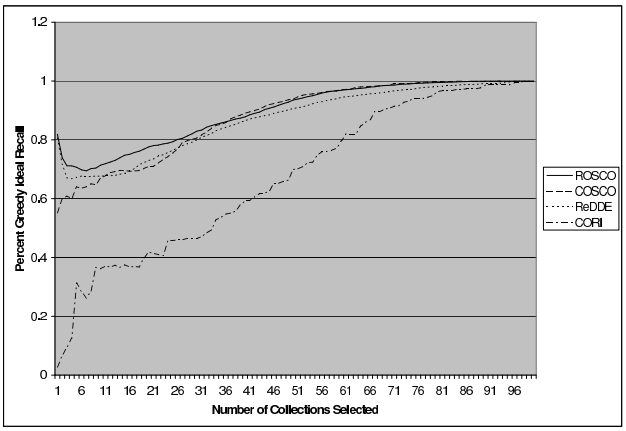

Figure 6: Results on testbed 8 (VCD)

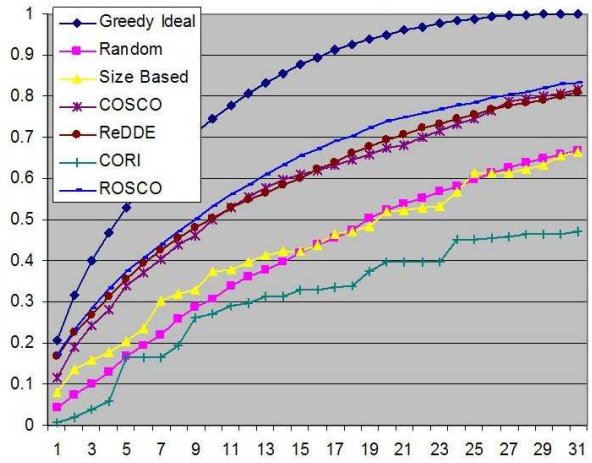

Figure 7: Percentage Recall of all methods for testbed 8 (VCD)

comprehensive results reported in [6], we show that CORI does perform much better on testbeds 1 and 5. Specifically, on testbed 1, CORI outperforms all other methods until about 10 collections have been selected. In testbed 5, it does perform significantly better than both sized based and random ranking.

9. CONCLUSION

This paper addressed the issue of collection selection for information retrieval in an environment composed of overlapping collections. We presented a method called ROSCO which adapts a state-of-the-art relevance based collection selection method, ReDDE, to consider collection overlap. We presented a systematic evaluation of the effectiveness of ROSCO over existing methods. Our experiments showed that ROSCO outperforms current best methods including CORI and ReDDE when there are overlapping collections, while being competitive in the others.

10. REFERENCES

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

[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] 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.

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

[14] 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.

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

[16] 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.

[17] M. Bender, S. Michel, P. Triantafillou, G. Weikum and C. Zimmer. Improving Collection Selection with Overlap Awareness in P2P Search Engines In *Proc. 28th Annual Intl. SIGIR Conf.*, 2005.

[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] 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.

[22] Y. Zhang, J. Callan and T. Minka. Novelty and Redundancy Detection in Adaptive Filtering In Proc. SIGIR 2002.