Improving Text Collection Selection with Coverage and Overlap Statistics

Thomas Hernandez
Arizona State University
Dept. of Computer Science and Engineering
Tempe, AZ 85287
th@asu.edu

Subbarao Kambhampati
Arizona State University
Dept. of Computer Science and Engineering
Tempe, AZ 85287
rao@asu.edu

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 which should thus be accessed to answer the query. This paper addresses the challenge of collection selection when there is full or partial overlap between the available text collections, a scenario which has not been examined previously despite its real-world applications. A crucial problem in this scenario lies in defining and estimating the overlap between text collections. We present COSCO, a collection selection approach which addresses these issues by approximating the overlap as the similarity between sets of results returned by the collections. Collection statistics about coverage and overlap are then gathered for past queries and used for new queries to determine in which order the overlapping collections should be accessed to retrieve the most new results in the least number of collections. This paper contains an experimental evaluation which shows that the presented approach displays the desired behavior of retrieving more new results early on in the collection order, and performs consistently and significantly better than CORI, previously considered to be one of the best collection selection systems.

Keywords

collection selection, collection overlap, statistics gathering

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. The emphasis there is on pinpointing the most relevant documents present in a single – usually very large – set of documents, or collection. Examples include news websites, online encyclopedias, scientific bibliographies, and even general search engines such as Google and Yahoo. The general approach used by these systems to identify relevant documents is to analyze a query in terms of its keywords and use term frequencies and document frequencies obtained from the collection to determine which document in the collection is most similar to the query content [2].

WWW2005, May 10–14, 2005, Chiba, Japan.

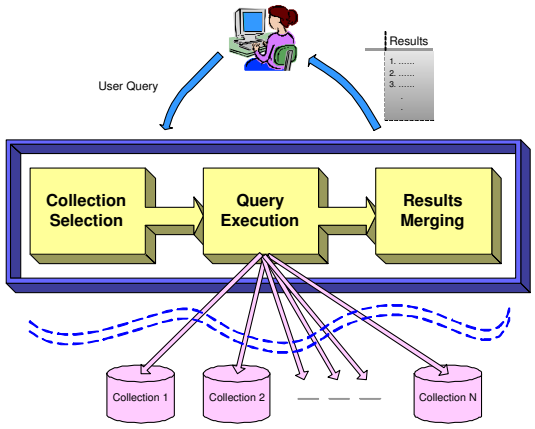

Figure 1: Multi-collection information retrieval.

A slightly more complicated scenario than the single-source environment occurs when a user wishes to query several collections simultaneously, as is the case in news meta-searchers or bibliography search engines. The challenge of retrieving relevant documents from a group of collections naturally involves the same types of approach as described above. However, having several potentially useful collections adds a distributed[1] aspect which must be solved prior to any actual retrieval of information. Unless the retrieval system intends to search every information source at hand – which of course would not be particularly efficient – it must choose 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 especially important because redundant or irrelevant calls are expensive in many respects: in terms of query execution cost, quality of results, 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.

Figure 1 illustrates the general architecture of a multi-collection information retrieval system. As shown in the

[1] *Distributed* simply refers to the fact that the information is spread over several collections; it does not necessarily imply that the collections are physically distributed.

figure, in addition to the *collection selection component*, the system requires two other components, which also constitute important research directions: query execution and results merging. The work presented in this paper aims to address the specific issue of collection selection and we will thus not go into more details about the two other components.

The general trend in the existing approaches for collection selection is to evaluate the "goodness" of each collection based on some type of information about term, document, and/or collection frequencies. In other words, these approaches require some term frequency statistics about each collection in order to select the sources they deem relevant to the query. This general strategy works fairly well when the collections do not overlap. However, because all of these approaches fail to take into account overlap between collections when determining their collection order, they may actually lead to two important problems:

1. They may decide to call a collection which has no new documents (considering the documents which have already been retrieved at that moment). Take for example the case of two mirror collections. If one is deemed highly relevant, the other one would also be highly relevant, and hence both collections would be called even though calling the second one does not provide any new results.

2. They may miss some documents from smaller sources. In particular, smaller sources would have a harder time appearing highly placed in the collection order, as their low frequencies could prevent them from competing "fairly."

Evidently a collection selection approach which could prevent the two cases mentioned above from occurring would be useful and perhaps even complementary to the existing approaches. Our motivation was thus 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 results. To do so, our system must be capable of making two types of predictions:

- How likely a collection is to have relevant documents, and

- Whether a collection is useful given the ones already selected.

This paper presents COSCO, our collection selection approach, which uses information on the coverage of individual collections to predict the first point, and information on the overlap between collections to predict the second point.

1.1 Challenges Involved

Using knowledge about coverage and overlap can definitely help in the overall distributed information retrieval process and in the collection selection phase in particular. In a structured environment like relational databases, the overlap for a given query between two databases can be simply defined as the number of result tuples that both databases have in common. However, overlap in the context of text collections is much less straightforward to define and assess than in the context of relational databases. Overlap in the relational model is easy to identify by using keys and easy to quantify by simply counting the number of duplicate tuples. In contrast, overlap between two text collections means that some documents are highly *similar*, as opposed to strictly identical. In fact, if we were to consider only identical text documents to determine the level of overlap between collections, we would not be able to avoid calling a collection that contains very similar documents as one that has already been called.[2] The overlap between two text collections could thus be quantified as the number of results that are similar above a certain threshold. The meaning of overlap having changed, its complexity of computation also has changed, not only because computing document similarity is usually more expensive than checking for duplicate tuple keys, but also because *a single result from one collection could be similar to several results from another collection.* In fact, the difficulty here lies in efficiently computing, gathering, and then adequately using the overlap information.

1.2 Overview of the Proposed Solution

The solution developed in this paper addresses the collection selection issue in an environment of overlapping collections by using statistics on both the coverage of each collection and the overlap between them. More specifically, the intent is to be able to determine, for any specific 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 results). Intuitively, we would want to call the most relevant collection first, and then iteratively choose high coverage collections that have least overlap with the already selected collection(s). It is interesting to note, though, that the framework we developed could also be used to determine which set of collections would actually retrieve more redundant documents – as opposed to more diverse documents, our stated goal – since the coverage and overlap statistics would serve us well in that respect.

Our approach, called[3] 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 instead. 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, as explained in Section 1.1, are more challenging to

[2]For example, we would hardly expect newspapers to publish perfectly identical stories.

[3]COSCO stands for **CO**llection **S**election with **C**overage and **O**verlap Statistics

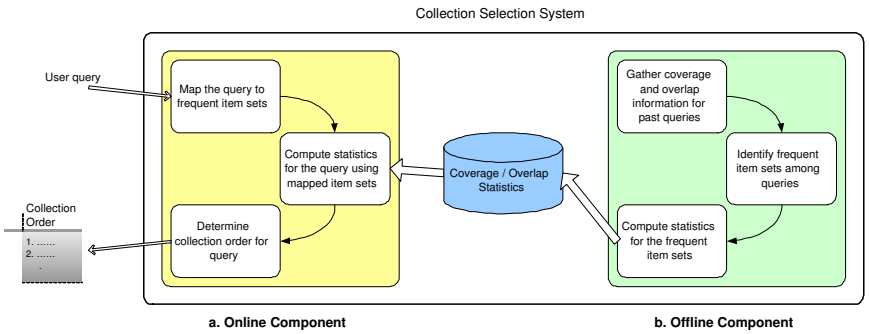

Figure 2: Architecture of COSCO, our collection selection system.

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.

1.3 Outline

The paper is organized as follows. Existing work related to the particular problems presented above is discussed in Section 2. COSCO, the contribution of this paper, is presented in Section 3. The experimental setup is described in Section 4, followed by the results in Section 5. Finally, we conclude in Section 6.

2. RELATED WORK

Several approaches have been taken to solve the collection selection problem. As mentioned by Powell and French in their systematic comparison of collection selection algorithms [14], 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 [6], the CVV ranking method [19], CORI [4], SavvySearch [8], and many other approaches [17, 12, 16, 9, 11, 5], but none actually address the challenges incurred by overlapping collections.

Using coverage and overlap statistics for source selection has been explored by Nie and Kambhampati [13], however our work differs sharply from theirs because their approach addresses the relational data model, in which overlap can be identified among tuples in a much more straightforward way. Others have suggested using coverage information [9, 18] or overlap information [18, 16] 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

3.1 General Approach

The problem of collection selection in a distributed group of overlapping collections has not previously been addressed in the literature, as it has usually been assumed that the group of collections constitutes a perfect partition of all documents available. COSCO, the system presented in this paper, concentrates on gathering coverage and overlap statistics of collections and using these statistics at query time to best estimate which set of collections should be searched. The general approach for COSCO is illustrated in Figure 2 and described in more detail in this section. As illustrated, the system 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.

3.2 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. First it must obtain the appropriate coverage and overlap information from the collections for a set of training queries. It must then identify frequent item sets among previously asked queries to better map new queries at runtime, as mentioned in Section 1.2. Finally it must compute new statistics corresponding to each of these item sets.

3.2.1 Gathering query statistics

3.2.1.1 Issues in defining overlap.

As was discussed in Section 1.1, the first issue that needs to be addressed is how overlap is defined in this text collection environment. Cost of computation obviously depends on whether we consider overlap to be equivalent to perfect identity or rather high similarity. In fact, 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 documents and – certainly in scenarios of text collections – redundant documents may not necessarily be perfectly identical.

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_1 that have a similarity higher than a predetermined threshold with a result in a collection C_2. Unfor-

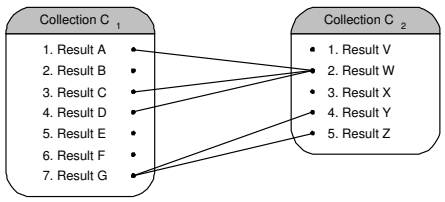

Figure 3: Two collections with overlapping results. A line between two results indicates that there is a high similarity between them.

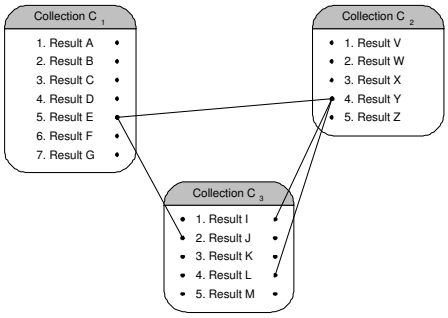

Figure 4: A simple example of three collections with overlapping results. A line between two results indicates that there is a high similarity between them.

tunately, using thresholds implies that collection overlap is non-symmetric, in that a single result in C_1 could very well be highly similar to several results in C_2. For example, Figure 3 shows a case where three results from collection C_1 – results A, C, and D – have a high similarity with a single result from collection C_2 – result W. Similarly, two distinct results from C_2 have a high similarity with result G from C_1.

An even more problematic situation arises when considering overlap between several collections. Extending the notion of result-to-result overlap described in the previous paragraph to more than two collections could become very expensive. The fact that we want to capture similarity of results – as opposed to simply equality – when computing the overlap statistics for multiple collections would indeed lead to some costly computation. Consider for example the simplified case illustrated in Figure 4. Result E in C_1 overlaps with result Y in C_2 as well as with result J in C_3, while result Y in C_2 only overlaps with results I and L in C_3. Conceptually, we could relate this possible situation to the fact that document overlap is not transitive. In such a scenario, it seems reasonable to consider that the three collections have some degree of overlap, however the difficulty lies in quantifying this overlap. Although this instance is not very likely to happen if the similarity threshold used to determine overlap between two results is high, it is still a situation that needs to be handled.

3.2.1.2 Definition of overlap.

To address the challenges mentioned previously, this paper proposes 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.[4] 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[5] $\mathcal{R}_{1q} \cap \mathcal{R}_{2q}$, 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,

$$\mathcal{R}_{iq} = \bigcup_{j=1}^{k} results_{C_i,q}(j) \quad (1)$$

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:

$$overlap_q(C_i, C_j) = |\mathcal{R}_{iq} \cap \mathcal{R}_{jq}| \quad (2)$$

Notice that collection overlap as defined above is now a symmetric relation since $overlap_q(C_i, C_j) = overlap_q(C_j, C_i)$.

3.2.1.3 Statistics stored.

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

[4]In other words a document contains a set of terms, each paired with its frequency of occurrence in the document.

[5]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 which appears in both D_1 and D_2 and for which the frequency is equal to the minimum number of times the word appears in either D_1 or D_2. For example, $\{(data, 2), (mining, 1), (integration, 2)\} \cap \{(data, 1), (integration, 2), (system, 1)\} \equiv \{(data, 1), (integration, 2)\}$.

definition will be used for the coverage of a collection C_i with respect to a query q:

$$coverage_q(C_i) = \frac{|results_{C_i,q}|}{\sum_{j=1}^{n} |results_{C_j,q}|} \quad (3)$$

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 3 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 3.3.3 when describing the collection selection algorithm.

3.2.2 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 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 3.3.

3.2.3 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:

$$\overrightarrow{stats_{IS}} = \sum_{q_i \in \mathcal{Q}_{IS}} \frac{freq_{q_i}}{\sum_{q_j \in \mathcal{Q}_{IS}} freq_{q_j}} \times \overrightarrow{stats_{q_i}} \quad (4)$$

As apparent in Formula 4, 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 $\frac{freq_q}{\sum_{q_j \in Q_{IS}} freq_{q_j}}$ 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.[6] 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:

$$\overrightarrow{stats_{IS_{empty}}} = \frac{\sum_{IS \in item\_sets} \overrightarrow{stats_{IS}}}{|item\_sets|} \quad (5)$$

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.

3.3 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.

3.3.1 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

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

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[7]. 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.

3.3.2 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 $\overrightarrow{stats_{q_{new}}}$:

$$\overrightarrow{stats_{q_{new}}} = \begin{cases} \frac{\sum_{IS \in IS_{q_{new}}} \overrightarrow{stats_{IS}}}{|IS_{q_{new}}|}, & \text{if } IS_{q_{new}} \neq IS_{empty} \\ \overrightarrow{stats_{IS_{empty}}}, & \text{if } IS_{q_{new}} = IS_{empty}. \end{cases} \quad (6)$$

3.3.3 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, then it is the collection that would provide 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 document.

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 7. At each step k, we select collection C_l such that

$$l = \begin{cases} \text{for } k = 1: \quad \underset{i}{argmax} \left[coverage_{q_{new}}(C_i) \right] \\ \text{for } k > 1: \\ \quad \underset{i}{argmax} \left[|\mathcal{R}_{iq_{new}}| - \sum_{C_j \in \mathcal{S}} overlap_{q_{new}}(C_i, C_j) \right] \end{cases} \quad (7)$$

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 3.2.1.2, 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 4, the approximation presented here is quite effective.

4. EXPERIMENTAL SETUP

The experiments described in this section were designed to determine how well COSCO performed in an environment of overlapping text collections and how it compared against state of the art approaches that do not take overlap statistics into account. For the latter, we used CORI,[8] which has been shown to be one of the best approaches [14]. The experiments were performed in the domain of scientific bibliography collections, where the documents are publications containing a title, an abstract, author names, etc. This section describes which list of queries was used in the experiments and which collection test bed the evaluation was performed on.

4.1 List of Real User Queries

A list of real user queries was required to collect coverage and overlap statistics, as well as to identify frequent item sets. The queries were selected from the query-list gathered by the BibFinder mediator [3, 13]. All the queries with a

[7] even though smaller sets could cover the query (e.g. {*data, integration*} and {*mining*})

[8] Our own implementation of CORI was based on the work described in [4].

frequency greater than or equal to 4 were considered,[9] which resulted in 1,062 distinct queries and a total cumulative frequency of 19,425. Approximately 87% of the queries had frequency less than 10, and 95% had frequency less than 50. Interestingly the average number of terms per keyword query was 2.2, which is surprisingly close to the average length of queries posed to online search engines according to some past surveys [10, 15].

4.2 Collection Test Bed

Six publicly available collections allowing full-text keyword search were used.[10] Each collection was sent the set of 1,062 queries and the top-20 results returned for each query were retrieved, stored, and indexed for each collection. Not all queries lead to 20 results and therefore a total of 89,177 results were actually retrieved.

In addition to the six online collections described above, nine synthetic collections were created with the intent of having both a relatively large test bed as well as a test bed which definitely shows some controlled degree of overlap between collections:

- Six collections were created by taking a random proper subset of 7,000 documents from each of the six real collections.
- Three additional collections were created by "unioning" subsets of other collections.

Table 1 provides a summary of the test bed of fifteen collections used for the experiments.

Real Collections	# of docs	Synthetic Collections	# of docs
acmdl	15,207	*acmdl_sub*	7,000
acmguide	17,364	*acmguide_sub*	7,000
sciencedirect	13,047	*sciencedirect_sub*	7,000
compendex	14,458	*compendex_sub*	7,000
citeseer	16,343	*citeseer_sub*	7,000
csb	12,758	*csb_sub*	7,000
		sc_cs[11]	14,000
		cp_ci[12]	14,000
		mix_15[13]	13,374

Table 1: Complete test bed of collections with the number of documents they contain.

The nine synthetic collections were considered as being complete bibliographies and a keyword-based search engine was built on top of each. It is important to realize that one cannot make the assumption that the results of a real collection C for a query q are a super set of the results from an artificial collection *C_sub* for the same query q, since the ranking method used by these newly created collections was certainly different from the method used by the real collections they originated from.

[9]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.

[10]The ACM Digital Library, the ACM Guide, ScienceDirect, Compendex, CiteSeer, and the Computer Science Bibliography.

[11]*sc_cs* is the union of *sciencedirect_sub* and *csb_sub*.

[12]*cp_ci* is the union of *compendex_sub* and *citeseer_sub*.

[13]*mix_15* contains 15% of each of the six real collections.

5. EXPERIMENTAL RESULTS

5.1 Training the System

The training phase of COSCO is essentially handled by the offline component of the system and its purpose is to compute and store some coverage and overlap statistics for the online component to use.

The first step in the process is to prepare a list of training queries. In the experiments described here, the training query list was composed of 90% of the query-list described in Section 4.1. The remaining 10% were used when testing the system (see section 5.2). The training and testing queries thus formed two disjoint sets. Note that this means in particular that our underlying assumption that future queries would follow the same distribution as past queries does not fully hold, which should make it more challenging for COSCO.

With the set of training queries in hand, the next step was to probe the collection test bed by sending each query to each collection, retrieving the set of top-20 results, and keeping coverage and overlap information for each query. Identifying the frequent item sets in the training query-list was performed by setting the minimum support threshold to 0.05%,[14] which resulted in a total of 681 frequent item sets. Note that to avoid useless item sets, stop-words were removed from the queries before the item set computation. The computation itself only took a few seconds to run.

Finally, the last step the offline component must perform is to compute the statistics for the frequent item sets and store them for later use by the online query-processing component. The statistics vectors $\overrightarrow{stats_{IS}}$ (one for each frequent item set) and $\overrightarrow{stats_{IS_{empty}}}$ were computed as explained in Section 3.2.3. The statistics fit in a 1.28MB file.

5.2 Testing the System

Testing our collection selection approach consisted in having the online component of our system process each test query. The main goal of these experiments was to analyze to what degree each approach, including ours, was able to output a collection order which ensured that when a collection was called, it was the one which provided the most new results. A useful way to measure this is by keeping track of the cumulative number of "new" results retrieved in terms of the number of collections called. At this point we should specify what is meant by a "new" result. A new result is one that is not a duplicate of a result which has been retrieved previously. We use the term duplicate in a loose way, meaning that we consider a result to be a duplicate of another if both are highly similar. In the experiments described here, the similarity measure used was the Cosine Similarity [2], with term frequencies as the term weights in a document.

5.2.1 Performance of COSCO

[14]An item set was essentially considered frequent if it appeared in 9 or more queries.

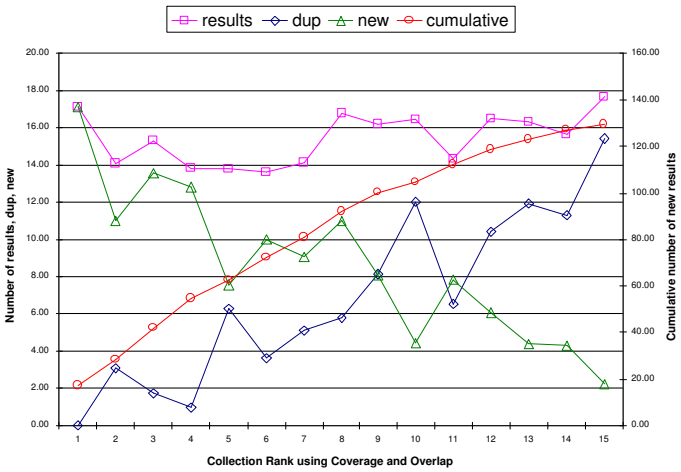

Figure 5: Performance of COSCO on the 15-collection test bed.

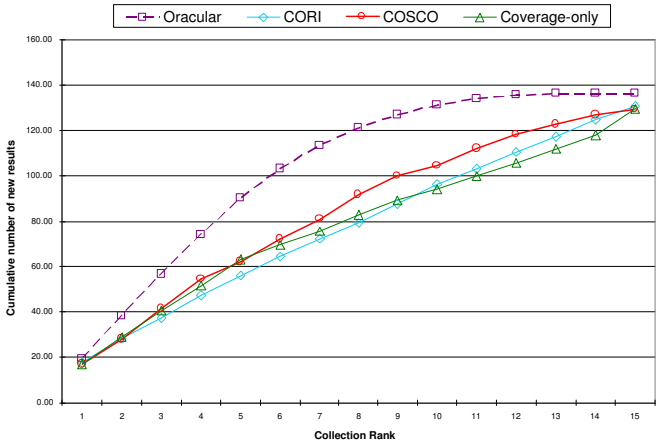

Figure 6: Performance of *Oracular*, CORI, COSCO, and a variation of our approach on the 15-collection test bed.

Figure 5 shows the average performance of COSCO on the test bed when using the statistics computed by the offline component. The graph contains four useful plots. *results* simply plots the average number of results retrieved by the i^{th} collection when calling collections in the order determined by COSCO. *dup* plots the average number of results among those retrieved by the i^{th} collection which had high similarity with at least one result retrieved from the $i-1$ previous collections called. Similarly, *new* plots the average number of new results among those retrieved. As explained earlier, the main graph to observe in this figure is *cumulative* (whose y-axis is on the right-hand side of the figure), which plots the cumulative number of new results retrieved. *cumulative* truly shows how well the system can suggest the order in which the collections should be accessed in order to retrieve the most results possible from only i collections. Note first that the number of duplicates is lower than the number of new results retrieved as the number of collections called increases, up to a point where the number of duplicates naturally surpasses the new results. Specifically, our approach was able to suggest a collection order which ensured that, on average, the first nine collections called would return more new results than duplicates.

The number of new results retrieved follows a globally descending trend, which is also a desirable behavior in a collection selection system. In addition, the *results* plot shows that COSCO will prefer calling smaller collections with greater potential for new results than simply larger collections. It can be seen in fact that in order to retrieve the most new results after the first collection, our approach chooses to call first collections which return fewer yet new results, and only calls those larger collections in the second half of the collection calls. Notice how the number of duplicates surpasses the number of new results approximately at the same time those larger collections start being called.

5.2.2 Comparison of COSCO with the other Approaches

To better evaluate the overall performance of our system, COSCO was compared not only to CORI but also against an oracle-like collection selection strategy, which we call *Oracular*. We implemented *Oracular* by iteratively selecting the collection which truly offers the most new results given the collections already selected. Clearly this post-hoc method requires to actually know which and how many results each collection will return for a particular query.

Figure 6 displays the *cumulative* plots of the different approaches. *Oracular* clearly performs as expected, retrieving the bulk of the results in the first few collection calls. When considering the performance of CORI, the *cumulative* plot shows a surprisingly close-to-constant rate of increase. The detailed performance of CORI is not shown here due to space limitations,[15] but interestingly, at each new collection call, it was apparent that CORI would retrieve approximately the same number of new results as duplicates, unlike COSCO which was clearly able to retrieve more new results than duplicates in the top of the collection order.

Figure 6 also shows the performance of a variation of our approach which only used coverage statistics to determine the collection order. In that approach, the collections are called simply in order of decreasing estimated coverage.

From this figure it can be seen that CORI almost consistently requires $k+1$ collections to retrieve the number of results our approach retrieved in k collections. The largest gap appears for $k=9$. Both CORI and the coverage-only approach need 11 collections to retrieve what our approach did in 9. Finally it is worth noting that the coverage-only strategy outperforms CORI in the first half of the collection calls, before falling behind for the later half.

Naturally, *Oracular* is still far superior to the three other approaches, including ours, but it is certainly not reasonable to think a system using approximations and statistics – either collection and term-based or coverage and overlap-based – would be able to perform nearly as well as the oracle-like solution. Considering this, the plots do show that not only does COSCO perform consistently better than CORI, it also achieves a performance which is characteristic of a good

[15] More extensive experimental results can be found in [7].

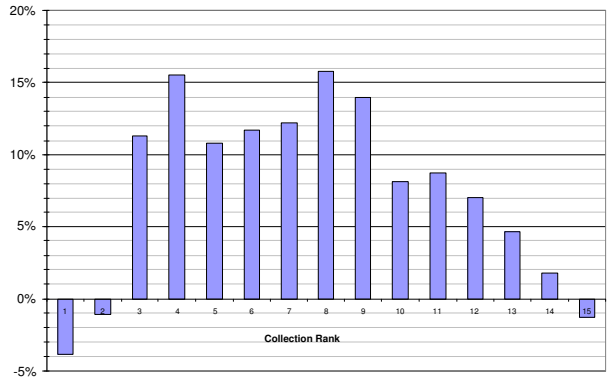

Figure 7: Percentage difference between the cumulative number of new results retrieved by CORI and by COSCO.

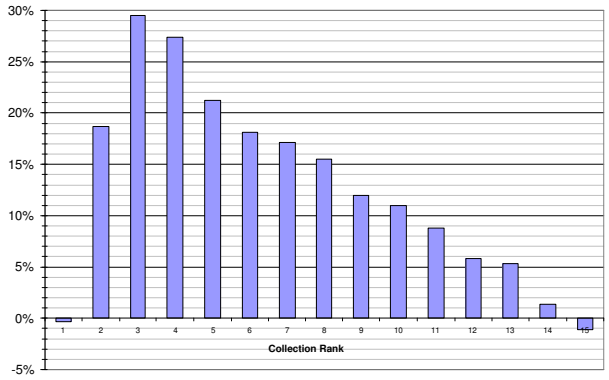

Figure 8: Percentage difference between the cumulative number of new results retrieved by CORI and by COSCO using a query test set with a similar distribution as the training set.

collection selection system: it retrieves more new results in the early collection calls.

Figure 7 illustrates the same experiments results under a different perspective. The figure displays the percentage difference between the cumulative number of new results retrieved by CORI after the x^{th} collection and those retrieved by COSCO. In fact, through most of the process our approach is able to retrieve between 5% and 15% more[16] new results than CORI, in the same number of collections.

5.2.3 *Effects of Query Distribution*

It is interesting to note that the performance of COSCO, as shown so far, was quite satisfactory in spite of a testing scenario which would most likely not allow our system to make full use of its underlying concepts. In particular, the set of queries used to test our system did not have the same distribution as the training queries. In fact, only approximately 50% of the test queries were actually mapped to one or more item sets for which statistics were stored.

Intuitively, since the initial assumption for our frequent item set approach was that queries frequently asked in the past would most likely be frequently asked in the future, we would expect our system to perform even better in a testing scenario which reflected this assumption. To test our intuition, additional experiments on our system were thus performed while ensuring that the query test set followed the same distribution as the training set.

To achieve similar distributions in both test and training sets, the same initial list of 1,062 queries was used. We mentioned earlier that the total frequency of the 1,062 distinct queries was 19,425. The strategy here was to consider these queries independently, and thus randomly select 90% of them (i.e 17,482) for the training set, and the remaining 10% (i.e. 1,943) for the test set. Once the training was performed by our offline component on this new set of queries, the test set was processed by our online component.

With this new experimental setup, approximately 90% of the test queries were successfully mapped to an item set for which the offline component had stored statistics.

We only show the performance of COSCO with respect to CORI in Figure 8, as this is where the greatest improvement can be observed.[17] Recall that in the previous experiments our approach was able to retrieve up to 15% more results than CORI when calling the same number of collections. Figure 8 actually shows that our system performs even better in the new experiments, which confirmed our intuition. It can in fact retrieve up to approximately 30% more results than CORI after only 3 collections, which indicates that COSCO is much more effective in selecting the top few collections. Up until the 8^{th} collection our system is able to retrieve upwards of 15% more new results than CORI.

In conclusion, this second set of experiments demonstrated that our system does indeed perform significantly better than CORI when future queries follow the same distribution as past queries. More importantly, our complete set of experiments showed that even in a scenario where future queries do not follow the distribution of past queries, our system consistently outperforms CORI and displays a behavior consistent with the Oracle-like behavior, retrieving more new results early on in the collection order.

6. CONCLUSION AND FUTURE WORK

This paper addressed the issue of collection selection for information retrieval in an environment composed of overlapping collections. We presented COSCO, a collection se-

[16]Notice that the plot shows a negative percentage for the first two collections, implying that COSCO retrieves fewer results than CORI. However, keep in mind that the percentage is based on the cumulative number of additional new results. Therefore when considering the actual number of results retrieved after the first collection – CORI retrieves an average of 17.79 results while COSCO retrieves 17.10 results, one can realize that the negative percentage is in fact insignificant. Similarly for the second collection call, where both CORI and our approach retrieve an average of approximately 28 results.

[17]More results for this second set of experiments can be found in [7].

lection system which takes into consideration the coverage of individual collections and the overlap between collections before determining which collection should be called next. The strategy consisted in having an offline component to gather and store coverage and overlap information about frequent keyword sets, and an online component which would use these stores statistics to determine the collection order for an incoming query.

Experiments showed that, when directly compared to CORI, COSCO leads to more new results being retrieved in the same number of collections. This is ultimately what the perfect collection selection method should guarantee: that the set of collections accessed results in the maximum possible number of new results obtainable from that many collections. Although we have shown and acknowledged that our system can obviously not pretend to achieve the same oracle-like behavior as the *Oracular* approach, it does provide significantly better collection orders than CORI and seems to be adopting the same type of behavior as *Oracular*. We have also shown that using coverage statistics alone cannot achieve the same quality of results. Finally, noting that COSCO outperformed CORI despite experiments which did not constitute the best possible scenario for our approach, we also showed that when the distribution of the test queries reflects the distribution of the training queries, then our system performs even better against CORI, which demonstrates the robustness of our approach.

Interestingly, while the current study and work related in this paper focus on improving diversity in the documents retrieved from a group of collections, one should note that the coverage and overlap statistics we concentrate on could also be used to accomplish just the opposite and instead determine which collections would return redundant documents.

Future directions based on this work include exploring the usefulness of removing some of the approximations made. For example, what would be the cost and benefit of estimating collection overlap using result-level similarities instead of using the similarity between result set documents. Furthermore, would it be possible and worth the computation to measure multi-collection overlap with more than just the pairwise overlaps, as was done in the work presented in this paper. Finally, probably the most interesting direction of future work attempts to design a collection selection system which would take into account both the content-based relevance of the documents and/or collections, as well as the overlap between the collections. This essentially considers our work as a complementary strategy to those that have been proposed in the literature, and preliminary work on the subject seems to point to a promising system.

7. 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] BibFinder: A Computer Science Bibliography Mediator. `http://rakaposhi.eas.asu.edu/bibfinder`, 2004.

[4] 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.

[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] 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.

[7] T. Hernandez. Improving text collection selection with coverage and overlap statistics. MS thesis, Arizona State University, October 2004.

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

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

[10] 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.

[11] 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.

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

[13] 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.

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

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

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

[17] 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.

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

[19] 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.