IMPROVING TEXT COLLECTION SELECTION WITH COVERAGE AND OVERLAP STATISTICS

by

Thomas L. Hernandez

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

December 2004

IMPROVING TEXT COLLECTION SELECTION WITH COVERAGE AND OVERLAP STATISTICS

by

Thomas L. Hernandez

has been approved

October 2004

APPROVED:

_____, Chair

_____

_____

Supervisory Committee

ACCEPTED:

_____

Department Chair

_____

Dean, Division of Graduate Studies

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 thesis 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. The collection selection approach presented here solves 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 thesis contains an experimental evaluation which shows that the presented approach performs consistently and significantly better than the Collection Retrieval Inference Network approach (CORI), previously considered to be the best collection selection system.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER 1 Introduction 1

1.1. Background 1

1.1.1. Single-source information retrieval 1

1.1.2. Multi-collection information retrieval 2

1.2. Motivating Example 4

1.3. Overview of Challenges Involved 7

1.4. Overview of the Proposed Solution 8

1.5. Outline of the Thesis 10

CHAPTER 2 Related Work 11

2.1. Collection Selection 11

2.2. Document Similarity 13

2.3. Coverage and Overlap Statistics 13

CHAPTER 3 COSCO: Collection Selection using Coverage and Overlap Statistics 15

3.1. General Approach 15

3.2. Gathering and Computing the Statistics: the Offline Component 16

3.2.1. Gathering query statistics 16

3.2.2. Identifying frequent item sets 22

3.2.3. Computing statistics for frequent item sets 23

Page

3.3. Collection Selection at Runtime: the Online Component 24
3.3.1. Mapping the query to item sets 24
3.3.2. Computing statistics for the query 27
3.3.3. Determining the collection order 27

CHAPTER 4 Experimental Evaluation 29
4.1. Experimental Setup 29
4.1.1. List of Real User Queries 30
4.1.2. Real Collections 30
4.1.3. Artificial Collections 34
4.2. Setting up CORI 36
4.3. Training the System 37
4.4. Testing the System 40
4.4.1. Testing the *Oracular* Approach 40
4.4.2. Testing the CORI Approach 43
4.4.3. Testing COSCO 44
4.4.4. Ablation Studies on Overlap Statistics 46
4.4.5. Comparison of the Various Approaches 46
4.4.6. Effect of the Granularity of Statistics 50
4.4.7. Effects of Query Distribution 53

CHAPTER 5 Discussion and Future Work 59
5.1. Capturing Overlap Between More than Two Collections 59
5.2. Computing Result-level Overlap 60

Page

CHAPTER 6 Conclusion 62

REFERENCES 64

LIST OF TABLES

Table		Page
1.	Statistics for the probing of the online bibliography collections.	33
2.	Complete test bed of collections with the number of documents they contain.	35
3.	Distribution of the probing and training query sets in terms of the number of keywords in each query.	38
4.	Frequent item sets mined with different support thresholds.	51

LIST OF FIGURES

Figure		Page
1.	Multi-collection information retrieval.	3
2.	Typical scenario in a multi-collection information retrieval system. Only a few collections end up being called for any given query.	6
3.	Architecture of COSCO, our collection selection system.	16
4.	Two collections with overlapping results. A line between two results indicates that there is a high similarity between them.	17
5.	A simple example of three collections with overlapping results. A line between two results indicates that there is a high similarity between them.	18
6.	Mapping query "data integration mining" to its related frequent item sets. **a**, **b**, and **c** show how the query would be mapped when the set of frequent item sets varies.	26
7.	Distribution of queries in the query-list in terms of their frequency.	31
8.	Cumulative percentage of queries in the query-list in terms of the query frequency.	31
9.	Offline component of the collection selection system.	38
10.	Online component of the collection selection system.	40
11.	Performance of *Oracular* – the oracle-like collection selection approach – on the 15-collection test bed.	41
12.	Performance of CORI on the 15-collection test bed.	44
13.	Performance of COSCO on the 15-collection test bed.	45
14.	Performance of a variation of our approach using only coverage statistics, on the 15-collection test bed.	47
15.	Performance of *Oracular*, CORI, COSCO, and a variation of our approach on the 15-collection test bed.	48

Figure Page

16. Percentage difference between the cumulative number of new results retrieved by CORI and by COSCO. 49

17. Performance of *Oracular*, CORI, our approach with frequency threshold 0.03%, and our approach with threshold 0.05%, on the 15-collection test bed. 52

18. Performance of COSCO on the 15-collection test bed using a query test set with a similar distribution as the training set. 55

19. Performance of *Oracular*, CORI, COSCO, and the coverage-only variation of our approach on the 15-collection test bed using a query test set with a similar distribution as the training set. 56

20. 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. 57

CHAPTER 1

Introduction

1.1. Background

1.1.1. Single-source information retrieval.

Traditional information retrieval techniques concentrate on solving the problem of finding which documents 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. News websites (e.g. CNN [3], The New York Times [17]), online encyclopedias (e.g. Encyclopedia Britannica [7], Columbia Encyclopedia [4]), and scientific bibliographies (e.g. ACM Guide [15], ScienceDirect [13], IEEE Xplore [10]) are all examples of keyword-based information retrieval systems which perform in the context of a single collection. In fact, perhaps less intuitively, even online search engines such as Google [8] and Yahoo [18] can be categorized as information retrieval systems searching through a single collection. The collection may be the set of pages available on the Internet in the case of web search engines, and proprietary content in the case of news, encyclopedia, and some bibliography websites.

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 [20]. Of course, in addition to simply comparing the contents of the document with the query in terms of keywords and frequencies, some systems take advantage of the environment and properties of the collection and compute other types of rankings which influence the final ordering of documents retrieved and returned to the user. For example, Google uses its PageRank algorithm to take into account not only the frequency of the query terms occurring in a page but also the popularity of each page [42].

1.1.2. Multi-collection information retrieval.

A slightly more complicated scenario than the single-source environment occurs when a user wishes to query several collections simultaneously. The challenge of retrieving relevant documents from a group of collections naturally involves information retrieval techniques as described in section 1.1.1. 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 indeed 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 time, quality of results, post-query processing (i.e. duplicate removal and results merging), network load, source load, and also when some sources ask for access fees. Naturally, as the number of collections increase, effective collection selection becomes essential for the performance of the overall retrieval

[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.

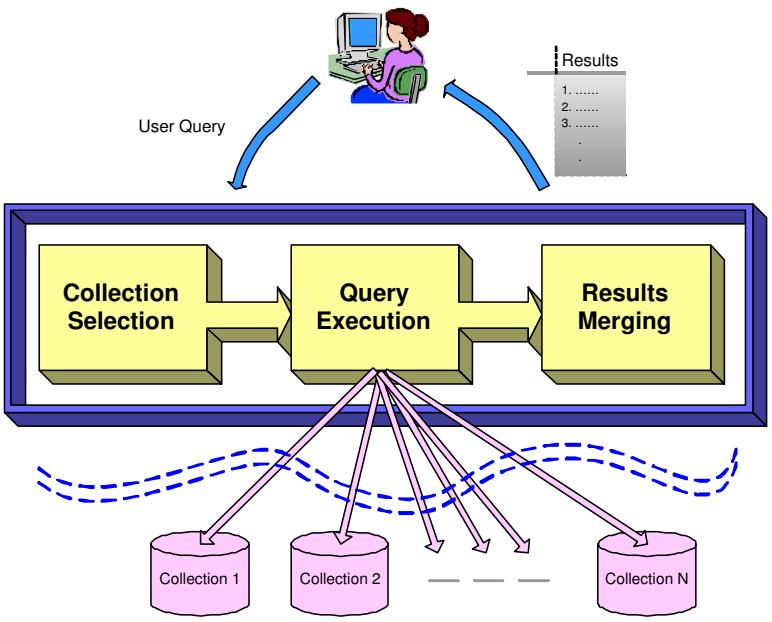

Figure 1. Multi-collection information retrieval.

system.

Figure 1 illustrates the general architecture of a multi-collection information retrieval system. As shown in the 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 *query execution component* is responsible for translating the user query into the schema of each of the underlying collections, sending that query out to the collections selected in the first phase, and finally effectively retrieving the results from the collections. The purpose of the *results merging component* is to process the list of results obtained in the second phase in order to return a clean (i.e. duplicate-free, consistent, and coherent) list of results to the user. The work presented in this thesis aims to address the specific issue of collection selection in an environment where the collections in the group contain some overlap. Section 1.2 illustrates this with an example.

1.2. Motivating Example

Several keyword-based systems have addressed the problem of collection selection and are mentioned in Section 2. Many independent experiments have shown that these systems perform quite well in an environment of multiple collections. However, a large majority of these systems assume that retrieval is performed over a non overlapping group of collections. Unfortunately there are many existing scenarios where this simplified non-overlap assumption is not justified, in particular in online meta-searching scenarios where the purpose is to search over several, usually overlapping, document databases or search engines. As mentioned by Chowdhury et al. [23], a real-world example of such a scenario is the search engine supported by the NCCAM.[2] It searches over several medical data sources, across which duplicates are quite common, given the nature of the data. Information retrieval systems for biological data are in fact needed. BioMediator [44, 39] is a data integration system for several online and overlapping biomedical databases, and as such it must also deal with the process of source overlap and source selection if it wants to achieve a reasonable degree of effectiveness and efficiency. The example below further motivates the problem.

> **Example:** Consider for instance a news meta-search engine similar to Google News [9], which attempts to retrieve articles from multiple news providers such as CNN, The New York Times (NYT), The Washington Post (WP), The Financial Times (FT), and The Wall Street Journal (WSJ). Chances are that for a particular event, most of these news providers would have an article on the subject and that all the articles relating to the same event would be very similar to each other if not simply the same article bought from an external source.

[2] National Center for Complimentary and Alternative Medicine, part of the National Institute of Health.

Considering the large number of news providers potentially available,[3] given a user query a news search engine aspiring to be as complete and effective as possible clearly cannot waste time and resources searching irrelevant or redundant news collections. In such a scenario, analyzing the coverage of each provider could help in determining which news sources are most relevant for a particular query. Furthermore, analyzing the overlap among the sources – in addition to the coverage – could help the news search engine in two important ways:

1. It could allow the engine to return many relevant and non-redundant articles instead of returning multiple copies of the same article with only minor variations.
2. It could help the engine optimize its query plan to call fewer news providers while still retrieving all or most of the relevant articles.

Figure 2 illustrates how collection selection would take effect in this particular example. Among the news providers mentioned above, one would assume for example that FT and WSJ would contain more documents for a query on "bank mergers" than the other providers. Similarly, one would expect these two providers to have a relatively high overlap in their articles. Let us assume that we want to determine which are the best two collections to access for that particular query in order to retrieve the most new results. Considering both the coverage and overlap properties of the available collections, it would probably be best to call only one of FT or WSJ – the one with highest coverage – and then call another collection with less overlap, such as CNN, to retrieve more diverse results. Figure 2 illustrates the process for this specific example. □

[3]Google News claims to include more than 4,500 news sources!

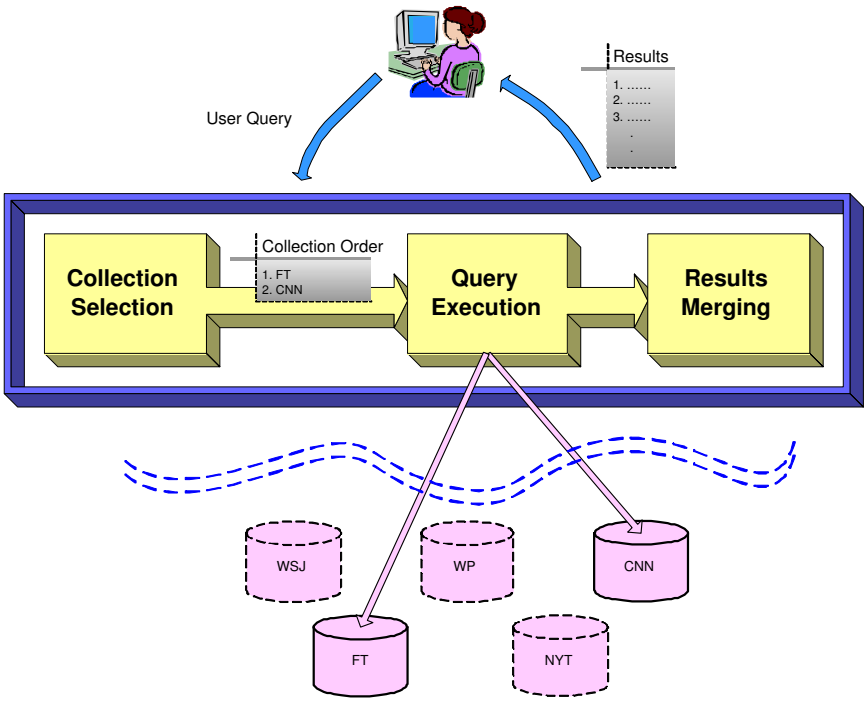

Figure 2. Typical scenario in a multi-collection information retrieval system. Only a few collections end up being called for any given query.

Of course, this example can be generalized to any keyword-based retrieval system that searches over a set of overlapping text collections. In fact, the experimental evaluation described in Chapter 4 was performed on a group of scientific bibliographies which, like the news collections mentioned in the example above, can have significant overlap in the publications they contain.

1.3. Overview of 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 could not use overlap to avoid calling a collection that contains very similar documents as one that has already been called.[4] The overlap between two text collections could thus be defined 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 the first collection could be*

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

similar to several results from the second collection. In fact, the difficulty here is how to efficiently compute, gather, and then adequately use the overlap information. This thesis addresses the issues described above and presents a system that gathers and uses overlap information to improve the collection selection step in a text database environment.

1.4. Overview of the Proposed Solution

The solution developed in this thesis 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). In our motivating example from Section 1.2 with the query "bank mergers", FT could be the source with highest coverage, and therefore would be chosen first by our approach. Note that WSJ would probably also have high coverage for that query, but it would most likely have high overlap with FT. Our system would then find which collection among WSJ, CNN, NYT, or WP has the least similarity with FT, and decide to rank that collection as second best for that particular query. Clearly WSJ is not a good candidate at this point. Assuming the second best collection was determined to be CNN, the system would then try to find the collection that has least overlap with both FT and CNN. This process would continue until the number of desired collections is reached.

Our approach, called COSCO[5], actually stores coverage and overlap statistics with respect to queries since different queries may lead to a different set of relevant collections. In essence, each query has its own statistics. In our motivating example, our system would gather the number of articles each news source returns for the query "bank mergers", and store this coverage information in association with the specific query. Similarly, our system would compute overlap statistics for that particular query as the similarity between the set of results returned by each news source for the query "bank mergers".

Storing the statistics with respect to queries 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 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. For example, our system would store coverage and overlap statistics for the item set {bank, mergers},[6] but it would also store statistics for the smaller item sets {bank} and {mergers}. Now if a new query "company mergers" is asked, then our system could use the statistics stored for the item set {mergers}, even though no statistics were stored for the set {company, mergers}.

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 1.3, are more challenging to estimate and we propose to compute the overlap between

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

[6]Assuming this keyword set is frequent enough.

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 thesis.

1.5. Outline of the Thesis

The thesis is organized as follows. Existing work related to the particular problems presented above is discussed in Chapter 2. COSCO, the contribution of this thesis – and solution to the challenges previously mentioned – is presented in Chapter 3. The experimental evaluation is detailed in Chapter 4. Some future work is then suggested in Chapter 5, before concluding in Chapter 6.

CHAPTER 2

Related Work

Several directions of related work in information retrieval are relevant to the research detailed in this thesis. In order to address the problem of collection selection in a distributed and overlapping environment, there are essentially three subproblems that need to be considered: collection selection, document similarity estimation, and gathering/using overlap statistics. Previous work related to each of these problems is described next.

2.1. Collection Selection

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 [43, 25], the main idea has been to try to create a representative for each collection based on term and document frequency statistics of the collection, and then use these statistics at query-time to determine which set from the pool of collections is most promising for the incoming query. This is the case for gGlOSS [27, 28], the Cue Validity Variance ranking method (CVV) [53], and the Collection Retrieval Inference Network approach (CORI) [21]. These three collection selection mechanisms attempt to determine which are the collections that contain documents very similar or relevant to the query. More specifically, gGlOSS

ranks the collections in terms of their *goodness*, which is in turn estimated as the sum of all document-query similarities in a collection. The CVV ranking method (from the D-WISE system) essentially looks at the distribution of each query term across the pool of collections, determines how discriminatory each query term is, and then for each collection calculates the sum of document frequencies for query terms weighted by the discriminatory factor of that term across all collections. Finally, CORI[1] takes the approach of representing each collection as a "virtual" document, for which the virtual term list is the set of terms present in the collection and the virtual term frequency is actually the document frequency of each term in the collection. The statistics these systems typically need can be the document frequency of each term, the collection frequency of each term, the number of documents in each collection, and/or the weight of terms over a collection.

More approaches have been proposed. For instance, Yu et al. [52, 50] rank the collections in terms of the similarity of the most similar document in the collection. Meng et al. [38, 36] try to estimate the number of useful documents in a collection to determine which collections are most relevant. The MRDD approach from Voorhees et al. [49, 48] uses training queries to learn how many documents should be retrieved from each collection. SavvySearch [30] utilizes past retrieval statistics to evaluate how well each collection responds to each possible query term. Ipeirotis and Gravano [32] probe the collections to extract content summaries and use a hierarchical topical classification of the collections to rank them when given a query. Liu et al. [37] present a probabilistic model which attempts to capture the error in the estimation of database relevancy for specific queries. King and Li [35] have suggested using singular value decomposition to analyze collection-to-collection relationships. Finally, Conrad and Claussen [24] look at domain-specific environments and

[1]CORI will be described more thoroughly when discussing the experimental evaluation in Chapter 4

rely on the users to categorize their query, and then use that information to select a set of collections.

2.2. Document Similarity

Unlike in relational databases, where overlap is simply defined as the number of identical tuples, text databases require a definition of overlap based on document similarity. Measuring exact similarity between documents (to simulate the relational model of overlap) requires duplicate detection mechanisms such as those described in [46, 23, 45]. These methods rely on fingerprinting documents or chunks of documents and comparing these fingerprints – or the number of fingerprints in common – to point out duplicates. Others use methods involving more traditional IR techniques, including Jaccard similarity (used by Haveliwala et al. in [29]) or the basic Cosine similarity (used by Yu et al. in [52]). Any of the similarity methods mentioned here could potentially be used for overlap computation in a group of text collections, depending on how strict we want the overlap to be (i.e. identical documents or highly similar documents).

2.3. Coverage and Overlap Statistics

Existing work on coverage and overlap statistics gathering has been done using probing queries to estimate statistics for each collection. Nie and Kambhampati [40] have used probing to gather both coverage and overlap statistics in a relational data model. In [41], 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. Their query classification then allows them to correctly identify for a new user query which is the set of collections that has the maximum cumulative coverage (thereby taking into

account overlap between collections). Note, though, that the main difference between the work described in this thesis and Nie and Kambhampati's approach in [41] – consisting of attribute-value hierarchies and query classes – is that, unlike in a relational environment, there are no attributes to classify queries on in a text collection environment.

Ipeirotis and Gravano [32] have also used coverage information to guide their probing strategy but have not looked at overlap at all. Yerneni et al. [51] have looked at coverage of information sources in trying to maximize the number of results a web mediator would retrieve. They have also considered overlap between pairs of sources, but their approach focused on qualitative overlap information (e.g. disjointness, equivalence, etc.) which they assumed to be readily available. Finally, Voorhees et al. [49, 48] have suggested using overlap between *queries* to cluster them and then use these collection-specific clusters to estimate how many documents should be retrieved from each available collection. Unlike in the work presented in this thesis, they consider strict identity of documents as their overlap estimation metric. Furthermore, the overlap between *collections* is not taken into account, as they compute query overlaps for each individual collection.

CHAPTER 3

COSCO: Collection Selection using Coverage and Overlap Statistics

3.1. General Approach

The work described here addresses the problem of collection selection in a distributed group of overlapping collections. This problem has not previously been addressed in the collection selection literature, as it has usually been assumed that the group of collections constitutes a perfect partition of all documents available. COSCO, the solution presented in this thesis, 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, as explained with an example in Section 1.4. The general approach for COSCO is illustrated in Figure 3 and described in more detail in the next sections. As can be seen from the figure, 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.

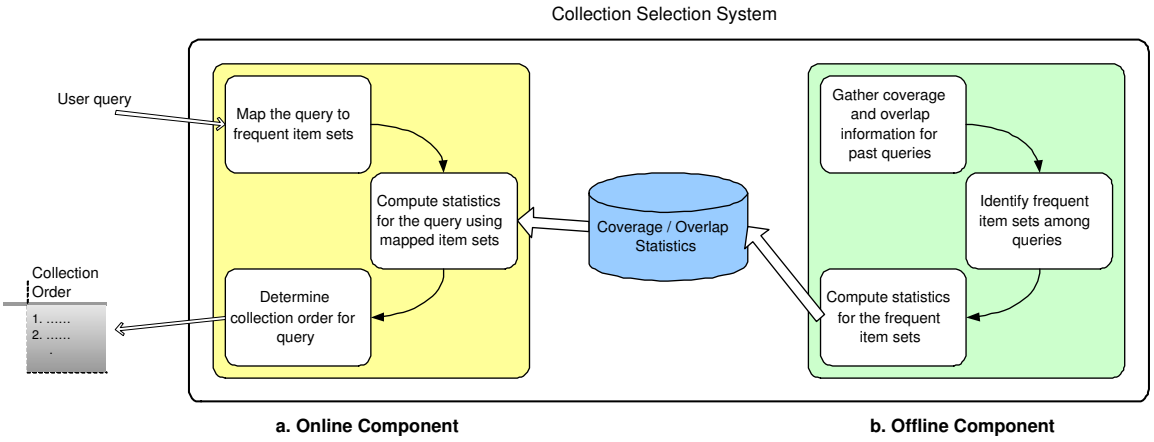

Figure 3. Architecture of COSCO, our collection selection system.

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. It must then identify frequent item sets among previously asked queries to better map new queries at runtime, as mentioned in Section 1.4. 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.*

Working with text collections, and therefore keyword-based retrieval, complicates both the retrieval process and the overlap computation. As was discussed in Section 1.3, the first issue that needs to be addressed is how overlap is defined in this new environment. Cost

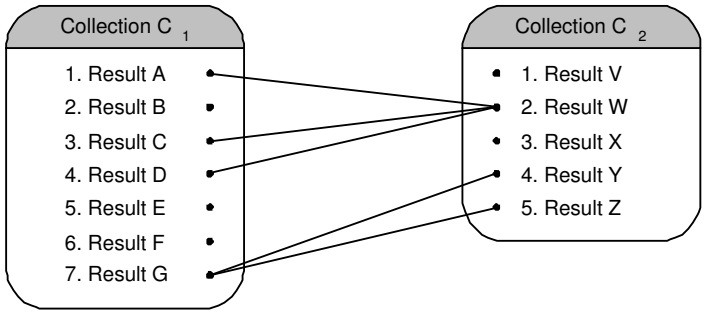

Figure 4. Two collections with overlapping results. A line between two results indicates that there is a high similarity between them.

of computation will definitely depend 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 more appropriate 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. The difficulty in this overlap computation follows from the fact that a single result in C_1 could very well be highly similar to several results in C_2, and vice versa. For example, Figure 4 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. These concepts are closely related to the notions of containment and subsumption relationships. Measuring overlap in the cases mentioned

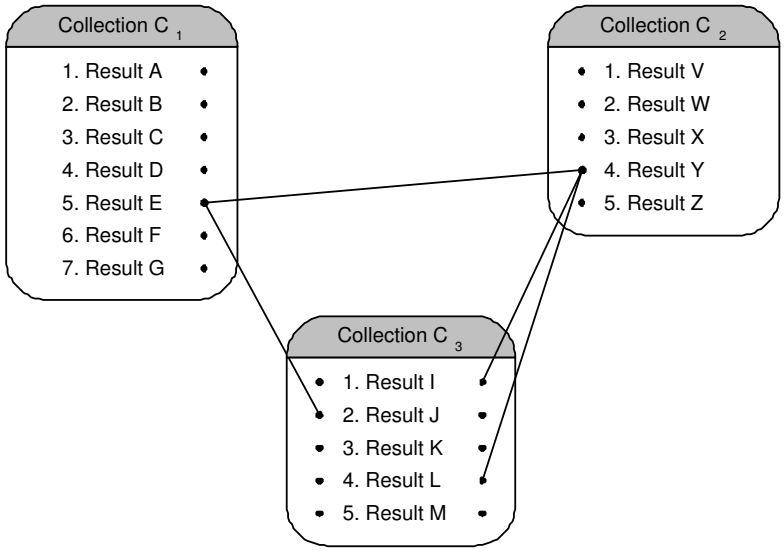

Figure 5. A simple example of three collections with overlapping results. A line between two results indicates that there is a high similarity between them.

above is essentially analogous to evaluating how one collection's result set is contained in or subsumed by another collection's result set.

An even more problematic situation arises when considering overlap between several sources. 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 5. 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. 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 would need to be handled.

3.2.1.2. *Definition of overlap.*

To address the challenges mentioned previously, this thesis proposes an overlap approximation which amounts to considering each query result set over a particular collection as a single document (i.e. a bag of words) instead of a set of documents (i.e. a set of bags). Overlap between two collections for a particular 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. 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.

Following this approach, the overlap between collections C_1 and C_2 for a query q is computed as the size of the intersection $R_{1q} \cap R_{2q}$, where R_{iq} is the document corresponding to the union of the results for query q from collection C_i.[1] In other words,

$$R_{iq} = \bigcup results_{C_i,q} \tag{3.1}$$

where $results_{C_i,q}$ refers to the set of all results returned by collection C_i for query q. Hence,

[1]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, $\{data, mining, integration, data, integration\} \cap \{data, integration, integration, system\} \equiv \{data, integration, integration\}$.

the definition for overlap is the following:

$$overlap_q(C_i, C_j) = |R_{iq} \cap R_{jq}| \tag{3.2}$$

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

3.2.1.3. *Other approaches to gather overlap statistics.*

Instead of considering entire sets of documents and "union-ing" these into a single result-set document as done in formula (3.1), another method to compute overlap between two collections could be to directly take advantage of the way the underlying collections return their results. More specifically, if snippet-like results are returned first by the collection instead of the actual results, then perhaps an overlap measure using simply the snippets as representatives of results would be more efficient and as effective. The drawback of this latter approach however is that different collections may display snippets in different formats, making them difficult to compare. More importantly, since each collection has its own method for extracting a snippet from a given document, it is not clear that two identical or highly similar documents would even have similar snippets (e.g. consider the case when a collection generates a query-dependent snippet while another collection stores a single static snippet for each document).

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

$$coverage_q(C_i) = \frac{|results_{C_i,q}|}{\sum_{k=1}^{n} |results_{C_k,q}|} \tag{3.3}$$

where n is the total number of collections being considered by the collection selection engine. Notice that the denominator $\sum_{k=1}^{n} |results_{C_k,q}|$ in Equation 3.3 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:

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

Note that in addition to coverage and overlap statistics, we also store $|R_{iq}|$ statistics. The size of the result set documents is indeed necessary and its usage will be clarified in Section 3.3.3 when describing the collection selection algorithm. The $\overrightarrow{stats_q}$ vector must therefore be computed for each previously asked query before moving on to the next step, described in Section 3.2.2.

3.2.2. Identifying frequent item sets.

With an overlap criterion now in hand and a statistics vector available for each known 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 be not only 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 identifying not only queries but also 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 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.

The Apriori algorithm [19] was used to discover the frequent item sets. Recall that the algorithm relies on the anti-monotonicity property, stating that a frequent set cannot have an infrequent subset.

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 Q_{IS} denote the set of previously asked queries that contain the item set IS. As an example,

$$Q_{\{data,integration\}} = \{\text{“}data\ integration\ system\text{”}, \text{“}data\ mining\ and\ data\ integration\text{”}, ...\}$$

The statistics for an item set IS are defined as the weighted average of the statistics of all the queries in Q_{IS}, according to the following formula:

$$\overrightarrow{stats_{IS}} = \sum_{q_i \in Q_{IS}} \frac{freq_{q_i}}{\sum_{q_j \in Q_{IS}} freq_{q_j}} \times \overrightarrow{stats_{q_i}} \tag{3.4}$$

As apparent in formula (3.4), 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.[2] 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 $\overrightarrow{stats_{IS}}$

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

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:

$$\overrightarrow{stats_{IS_{empty}}} = \frac{\sum_{IS \in item\_sets} \overrightarrow{stats_{IS}}}{|item\_sets|} \tag{3.5}$$

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. This step resembles a minimum set cover problem [26, 31] with a few variations. 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 the item sets {data}, {mining}, {integration}, {data, mining}, {data, integration} are all frequent, while {integration, mining} and {data, integration, mining} are not. In that case, the query will be mapped to both frequent two-term sets, even though smaller sets could cover the query (e.g. {data,

integration} and {mining}). Furthermore, note that if the item set {integration, mining} was frequent, then the query would be mapped to this set in addition to the other two-term sets. Finally, if the item set {data, integration, mining} was frequent,[3] then the query would only be mapped to this three-term set. An illustration of the query mapping process with some of the examples described above is shown in Figure 6.

The algorithm used to map the query to its frequent item sets is given in Algorithm 1. Practically speaking, the query q is mapped by first taking all frequent item sets that are

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
```

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.

Notice that the algorithm would still perform its intended task if lines 3 and 4 were removed and line 5 instead read "$IS_Q \leftarrow \text{PowerSet}(Q)$." The step described in these lines was inserted in the algorithm to minimize the power set computation by eliminating before-hand the query terms which are not frequent.

[3]Note that this would imply that item set {integration, mining} was also frequent.

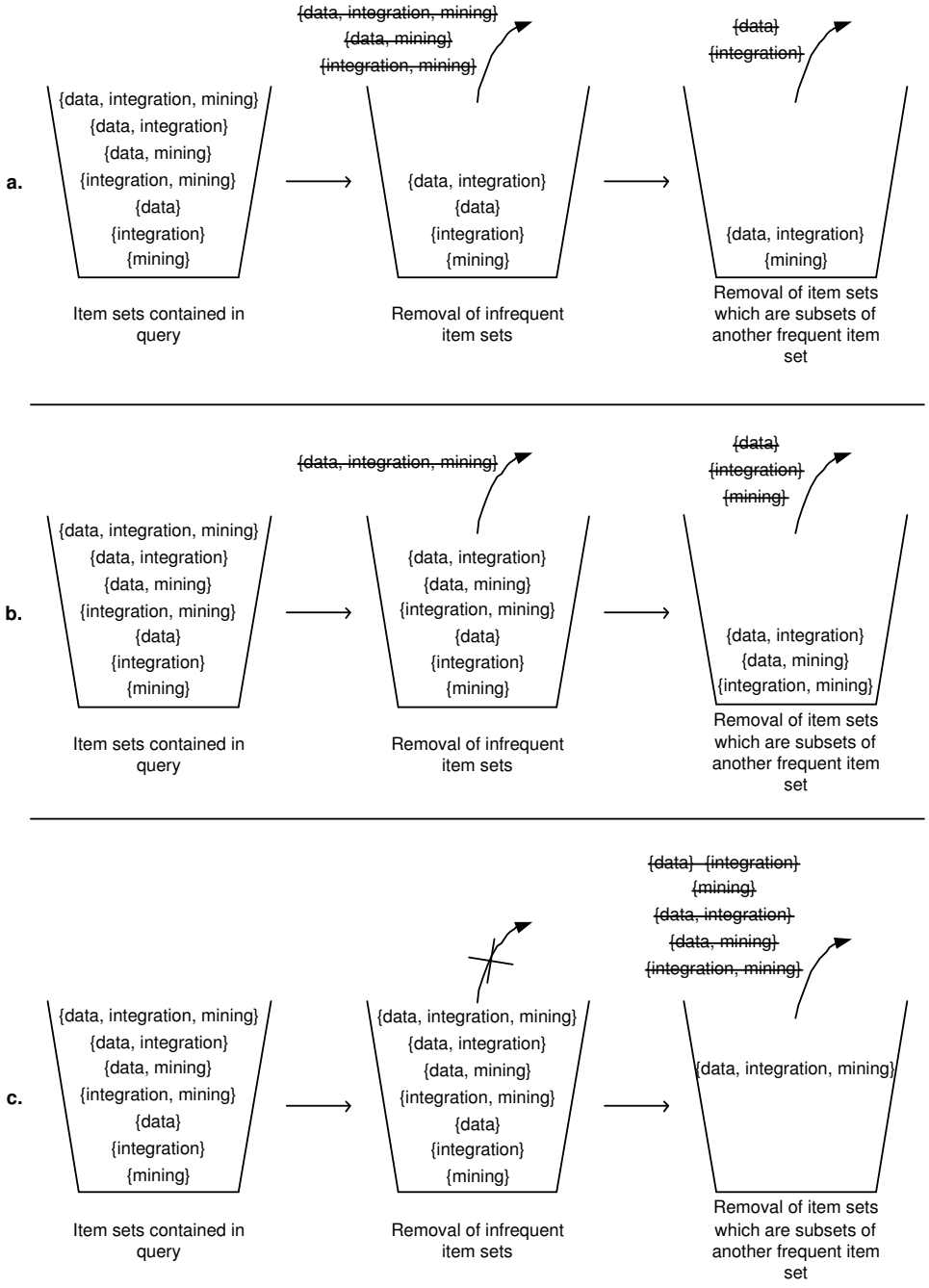

Figure 6. Mapping query "data integration mining" to its related frequent item sets. **a**, **b**, and **c** show how the query would be mapped when the set of frequent item sets varies.

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, in the case shown in Figure 6a, $IS_{q_{new}} = \{\{data, integration\}, \{mining\}\}$, which means that 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:

$$\overrightarrow{stats_{q_{new}}} = \frac{\sum_{IS \in IS_{q_{new}}} \overrightarrow{stats_{IS}}}{|IS_{q_{new}}|}$$

In the case where $IS_{q_{new}} = \{\} = IS_{empty}$, in other words the query q_{new} was not mapped to any item set, then we approximate $\overrightarrow{stats_{q_{new}}}$ as being equal to $\overrightarrow{stats_{IS_{empty}}}$. In summary, we can write the following definition for $\overrightarrow{stats_{q_{new}}}$:

$$\overrightarrow{stats_{q_{new}}} = \begin{cases} \frac{\sum_{IS \in IS_{q_{new}}} \overrightarrow{stats_{IS}}}{|IS_{q_{new}}|}, & \text{if } IS_{q_{new}} \neq IS_{empty} \\ \overrightarrow{stats_{IS_{empty}}}, & \text{if } IS_{q_{new}} = IS_{empty}. \end{cases} \tag{3.6}$$

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 (3.7). At each step k, we select collection C_l such that

$$l = \begin{cases} \underset{i}{argmax} \left[coverage_{q_{new}}(C_i) \right], & \text{for } k = 1 \\ \underset{i}{argmax} \left[|R_{iq_{new}}| - \sum_{C_j \in \mathcal{S}} overlap_{q_{new}}(C_i, C_j) \right], & \text{for } k > 1 \end{cases} \tag{3.7}$$

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 chapter 4, the approximation presented here is quite effective.

CHAPTER 4

Experimental Evaluation

The experiments described in this chapter were designed to test whether the collection selection system presented in this thesis can in fact perform better in an environment of overlapping text collections than the systems commonly accepted as being the most effective by the information retrieval community. The experiments were performed in the domain of scientific bibliography collections. This chapter covers the entire experimentation procedure performed to test our system, including the experimental setup, a description of the system used as a comparison, details about the training performed by the offline component of our system, and finally results obtained with the online component.

4.1. Experimental Setup

The test bed used for the experiments was composed of fifteen scientific collections: six were real online public bibliographies and the remaining nine were artificial collections created only for the purpose of the experiments described in this chapter. To complete the experimental framework, a list of previously asked queries was required to gather statistics and identify frequently occurring keyword sets. This section describes which list of past queries was used in the entire experimentation, and how the collections in the test bed were

either probed or created.

4.1.1. List of Real User Queries.

In order to test COSCO, the approach described in Chapter 3, a list of real user queries was required to collect coverage and overlap statistics, as well as to identify frequent item sets. As will be explained later, the queries were first used to probe the online bibliographies. The probing queries were selected as the most frequent queries appearing in the existing query-list gathered by BibFinder [1] in the work described in [41]. All the queries with a frequency greater than or equal to 4 were considered, which resulted in 1,062 distinct queries and a total cumulative frequency of 19,425. The distribution of the query-list in terms of the query frequencies is given in Figure 7. The cumulative distribution is shown in Figure 8, where it can be seen that approximately 87% of the queries had frequency less than 10, and 95% had frequency less than 50.

Since we were dealing with a keyword-based search scenario, and since the queries from the BibFinder query-list were relational, each query selected was transformed into a keyword query by simply merging all the fields from the relational query. For example, the query *author* = "*alon halevy*" *AND title* = "*data integration*" was converted to the keyword query "*alon halevy data integration*". Among the 1,062 distinct queries in the query list there were 1,482 distinct terms. 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 [33, 47, 34].

4.1.2. Real Collections.

The six publicly available collections used were the ACM Digital Library [14], the ACM Guide [15], ScienceDirect [13], Compendex [5], CiteSeer [2], and CSB [16]. In the remainder

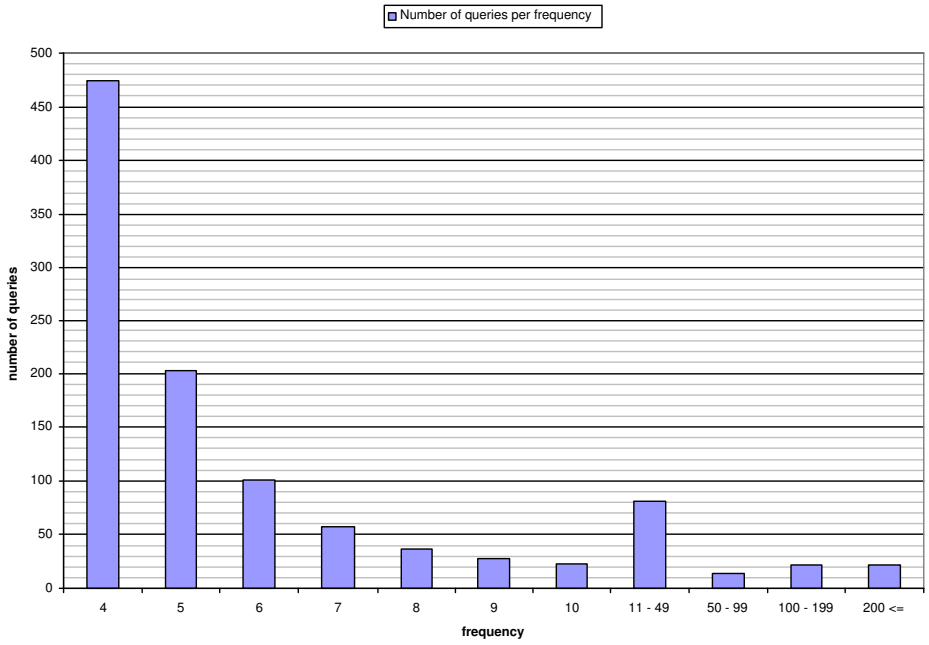

Figure 7. Distribution of queries in the query-list in terms of their frequency.

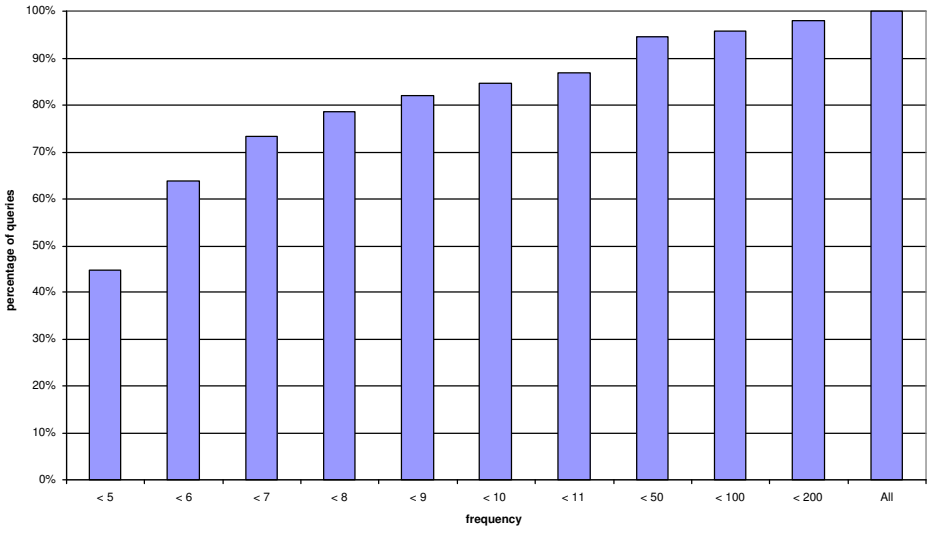

Figure 8. Cumulative percentage of queries in the query-list in terms of the query frequency.

of this chapter, these six collections will be referred to as *acmdl*, *acmguide*, *sciencedirect*, *compendex*, *citeseer*, and *csb*. These collections were picked among others because they offered a full-text keyword search, meaning that they provided the functionality of searching for keywords within the entire documents instead of only giving the option of binding specific attributes.[1] The text documents contained in these collections represent scientific publications and include the following textual attributes: author names, title, publication year, conference or journal name, and abstract. The concatenation of the attributes just mentioned constitutes a single document.[2]

The six online bibliographies were probed in order to collect actual results for the queries in the query-list. During probing, each of the queries were sent to each collection at regular yet polite time intervals. To remain in the keyword-based search scenario, the collection wrappers were built so that the queries would be asked as general keyword queries in the collection interfaces. In other words, no binding was performed. For each query, the top-20 results returned by a collection were retrieved. As mentioned earlier, the results were documents containing title, abstract, conference or journal name, year, and authors. These documents were then stored and indexed by query and collection. Note that the collections did not always contain all information for a particular result, in which case only the available fields were stored as a document. Note also that each collection had its own ranking strategy, which the end-user is generally not aware of. This means in particular that the same query may lead to a different set of top-20 results on two distinct bibliographies even though the complete result set may be identical for the two collections. Finally, when the collections gave the option of ranking either by relevance or by date, the results retrieved

[1] This is why some other scientific bibliography collections such as DBLP [6] or the Network Bibliography [12] could not be included in the collection test bed: they only provided specific attribute bindings and did not let the user perform a general full-text keyword search.

[2] Note though that not all attributes were always available for each publication.

were the top-20 ranked by relevance.

Other than the actual set of results returned by the collections, the total number of results matching the query in a collection was also stored, when available. A summary of the probing results on the six online collections can be seen in Table 1, which illustrates the differences in coverage of each source and the number of results that were actually retrieved. The "Total Results" row in the table shows the grand total of the number of results found by a collection. This total is simply informative and may include some results several times if they were returned for different queries. It does however provide an idea of the size of each collection. The "Retrieved" row refers to the grand total of top-20 results which were retrieved, stored, and indexed. The theoretical maximum for each element in this row is of course $20 \times 1,062 = 21,240$. However, since not all queries lead to 20 results, this maximum was not reached, and overall the probes retrieved a total of 89,177 results. Finally, the "Abstracts" row shows the number of retrieved results which contained an abstract.

	acmdl	*acmguide*	*sciencedirect*	*compendex*	*citeseer*	*csb*
Total Results	3,109,107	5,489,438	784,356	9,625,222	1,032,566	N/A[3]
Retrieved	15,207	17,364	13,047	14,458	16,343	12,758
Abstracts	10,824	12,381	12,379	12,048	15,389	1,425[4]

Table 1. Statistics for the probing of the online bibliography collections.

It is interesting to note that even though *compendex* displays close to three times the number of total results from *acmdl*, *acmdl* still manages to retrieve slightly more results than *compendex*. The explanation for that observation is that *acmdl* would more often return

[3]The query interface of CSB does not display the total number of results matching a specific query. For the purpose of our experiments we consider the number of retrieved results as the coverage for *csb*.

[4]The number of abstracts retrieved from CSB is relatively low because the CSB collection only returns BibTeX entries, which rarely contain the publication abstract.

some results, while *compendex* would sometimes return much more results than *acmdl* for a single query and yet only the first 20 would be retrieved.

4.1.3. Artificial Collections.

In addition to the six online collections described above, nine others were artificially created using the contents of the online collections which were obtained during the probing phase described in the previous section. The set of artificial collections was 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. To that end, two classes of artificial collections were created: collections which were subsets of one of the six real collections, and collections which contained subsets of multiple real collections:

- Six collections were created by taking a random proper subset of 7,000 documents from each of the six real collections. The *citeseer_sub* collection was generated by iterating through the query-list in a random order, and randomly adding some of the results from *citeseer* for each selected query, until the number of documents selected reached 7,000. The collections *acmdl_sub*, *acmguide_sub*, *sciencedirect_sub*, *compendex_sub*, and *csb_sub* were created following the same randomized strategy.

- Three additional collections were created by "union-ing" multiple subsets of different real collections. The collection *sc_cs* is simply a collection containing the union of *sciencedirect_sub* and *csb_sub*, and therefore has a total of 14,000 documents. Similarly, *cp_ci* is the union of *compendex_sub* and *citeseer_sub*, and also contains 14,000 documents. Finally, *mix_15* is a collection which contains 15% of each of the six real collections. The 15% of each collection were selected in the same randomized procedure described above, for a total of 13,374 documents. Notice that this implies that

the distribution of the contents of *mix_15* is proportional to the distribution of the number of retrieved results shown in Table 1. Furthermore this means, for example, that the 15% of *citeseer* contained in *mix_15* are most probably not a subset of *citeseer_sub* but instead have some degree of overlap with it.

Table 2 provides a summary of the test bed of fifteen collections used for the experiments. In order for these artificial collections to be used in an information retrieval

Real Collections	documents	Artificial Collections	documents
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	14,000
		cp_ci	14,000
		mix_15	13,374

Table 2. Complete test bed of collections with the number of documents they contain.

scenario, it was also necessary to have a search capability for each of them. Therefore, the nine collections were considered as being complete bibliographies and an actual text search engine was built on top of each, using open source libraries from [11]. These libraries provided all required functionalities such as keyword-based full-text search, ranking of results, as well as statistics on the documents and terms contained in the collection.[5] The nine additional collections were thus each considered as a separate, searchable bibliography.

Notice though, that the ranking method used by these newly created collections was most likely different from the method used by the real collections they originated from. An

[5]These statistics would especially come in handy to implement the CORI approach, described in Section 4.2.

obvious consequence was that it was quite possible that, for example, the top results from *acmdl_sub* for a query differed somewhat from the top results from *acmdl*. A possible reason could be that *acmdl* used more fields than just the text fields contained in the documents in *acmdl_sub* to rank its results. Another reason could be that while all artificial collections used a variation of the Cosine similarity method with *tf-idf* term weights, it is possible that some of the real collections used other ranking paradigms such as the number of citations, which may clearly lead to different sets of results. In short, 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.

4.2. Setting up CORI

Chapter 2 mentioned that the three best-known collection ranking algorithms were CORI [21], gGLOSS [27, 28], and CVV [53]. CORI has been shown several times [43, 25] to be the most stable and effective of the three, which is why it was used as a basis to compare the performance of our collection selection approach.

The CORI collection selection algorithm represents each underlying collection by its terms, their document frequencies, and a few other collection statistics such as word counts. The approach essentially corresponds to the *tf-idf* method for document retrieval techniques, except that it considers the collections themselves as the documents. In other words, CORI can be seen as a *df-icf* approach, where *df* is the document frequency of a term in a particular collection and *icf* is the inverse collection frequency of that term.

For a given query, the CORI algorithm will evaluate the belief $p(t_k|C_i)$ in collection C_i for each query term t_k. The belief is calculated as follows:

$$p(t_k|C_i) = 0.4 + 0.6 * T * I \tag{4.1}$$

$$T = \frac{df}{df + 50 + 150 * cw/\overline{cw}} \tag{4.2}$$

$$I = \frac{log(\frac{|C|+0.5}{cf})}{log(|C| + 1.0)} \tag{4.3}$$

where:

df is the number of documents in C_i containing the term t_k,

cf is the number of collections containing the term t_k,

$|C|$ is the number of collections being ranked,

cw is the number of words in C_i,

$\overline{cw}$ is the average cw of the collections being ranked.

Once the belief values have been computed for each query term with respect to each collection, the values are averaged over the terms for each collection. Finally, the collection selection process simply ranks these estimated belief values and considers the collections in the ranked order obtained. More details on CORI can be found in [21].

4.3. Training the System

As was described in the previous chapter, COSCO requires both an offline and an online component. The offline component is essentially used during a training phase, where the purpose is to compute and store some coverage and overlap statistics for the online component to use. Recall that the three main activities of the offline component, as shown in Figure 9, are to gather statistics for the queries, identify frequent item sets, and compute statistics for these item sets.

The first step in the training 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 4.1.1. Specifically, 956 queries were randomly selected among the 1,062

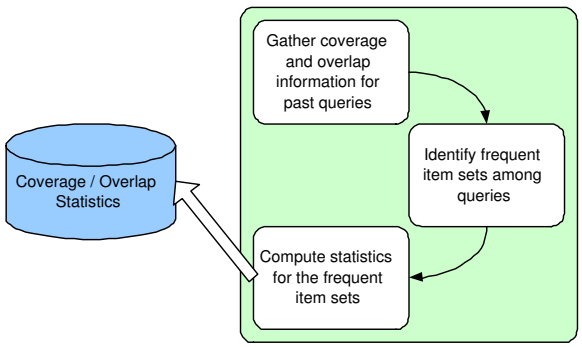

Figure 9. Offline component of the collection selection system.

distinct queries. The distribution of training queries in terms of their size is shown in Table 3. The remaining 106 queries were used when testing the system (see next section). The

Query size	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11+*	*Total*	*Average size*
Probing set	370	433	158	36	21	15	9	5	5	4	6	1,062	2.1685
Training set	334	388	143	32	19	15	7	5	4	3	6	956	2.1684

Table 3. Distribution of the probing and training query sets in terms of the number of keywords in each query.

training and testing queries thus formed two disjoint sets.

With the set of training queries identified, the next step was to probe the collection test bed by sending each query to each collection and retrieving the set of top-20 results. In addition to the list of results appearing in the top-20, the total number of results found in the collection was also stored in order to later compute the coverage statistics. Note that in practice, we did not actually need to probe the six real collections since we had already done so when creating the collection test bed. In fact, the results for each query – including the training queries – had previously been stored and indexed. Hence only the artificial collections were actually probed with the training queries.

Figure 9 suggests that the next step is to identify the frequent item sets in the training query-list. When setting the minimum support threshold to 0.05%, an item set was essentially considered frequent if it appeared in 9 or more queries, which resulted in a total of 681 frequent item sets. Of those, 412 were single-term item sets, 205 were two-term sets, 50 were three-term sets, 12 were four-term sets, and 2 were five-term sets.[6] The computation itself only took a few seconds to run.

Note that to avoid useless item sets, stop-words were removed from the queries before the item set computation, although the list was kept to a very small set.[7] In addition, the case arose where the same term could appear twice in the same query, which could lead to inconsistencies in the frequent item set computation. Consider the query "peer to peer network". For this query, the set {peer, network} was thus considered to appear only once although one could naturally argue that it appears twice. To keep the item set computation consistent, it was only counted once in this instance, since otherwise it would be possible for a frequent item set to appear more times than one of its subsets.

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 for the 0.05% frequency threshold, and in a 7.39MB file for the 0.03% frequency threshold.

[6]The two five-term sets were {2000, engine, giles, lawrence, search} and {ad, dynamic, hoc, networks, wireless}.

[7]The list of stop-words removed from the queries during the experiments contained the following words: "a", "an", "and", "in", "for", "of", "on", and "the".

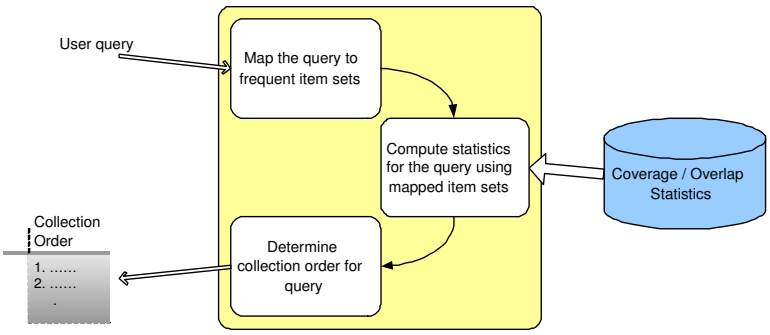

Figure 10. Online component of the collection selection system.

4.4. Testing the System

Testing our collection selection approach consisted in having the online component of our system process each query which was set aside when creating the training query-list. Recall from the previous section that there were 106 test queries, or 10% of all probing queries. The online component, as explained in Section 3.3 and depicted in Figure 10, is responsible for mapping the incoming user query to a set of frequent item sets for which the system maintains statistics, estimating coverage and overlap statistics for the query, and finally determining the best collection order given these statistics.

4.4.1. Testing the *Oracular* Approach.

To better evaluate the overall performance of our system, COSCO was compared not only to CORI, as was mentioned earlier, but also against the optimal collection selection strategy. The perfect strategy would involve iteratively selecting the collection which truly offers the most new results given the collections already selected. This method, which we will call *Oracular*, can be seen as an oracle-like method since it requires to actually know which and how many results each collection will return for a particular query. *Oracular* was thus

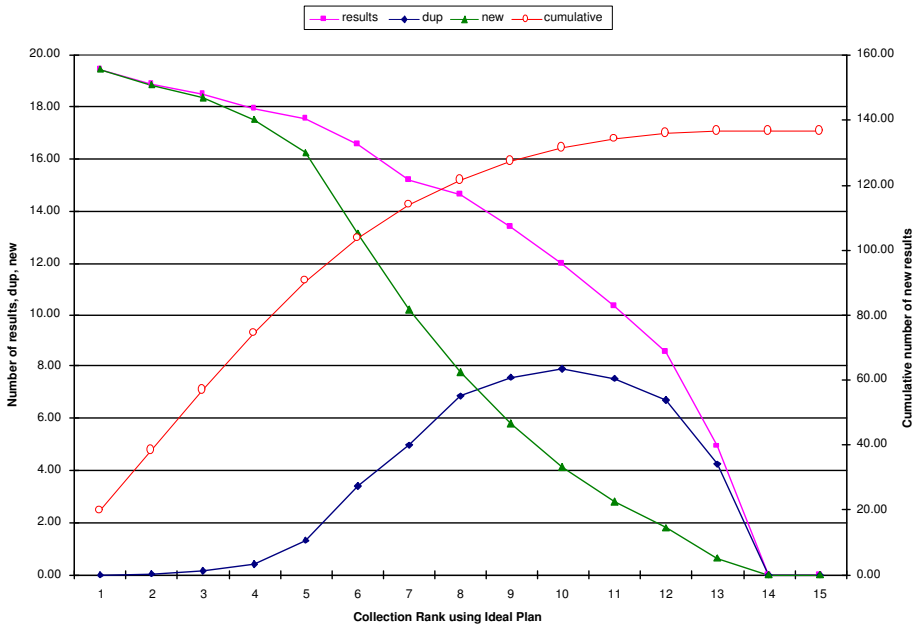

Figure 11. Performance of *Oracular* – the oracle-like collection selection approach – on the 15-collection test bed.

implemented for the purpose of these experiments.

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. Note that the similarity measure used is not relevant, as long as the same measure is used when comparing the different selection approaches. In the experiments described here, the similarity measure used was the Cosine Similarity, with term frequencies as the term weights in a document.

Having clarified what "new" and "duplicate" results are, the performance of the various approaches can now be analyzed. Figure 11 illustrates the overall performance of *Oracular*, averaged over the set of test queries. The collection selection approaches can be

analyzed individually with the four types of graphs present in the figure. The four graphs are *results*, *dup*, *new*, and *cumulative*. *results* simply plots the average number of results retrieved by the x^{th} collection when calling collections in the order determined by (in this case) *Oracular*. *dup* plots the average number of results among those retrieved by the x^{th} collection which had high similarity with at least one result retrieved from the $x-1$ previous collections called. Similarly, *new* plots the average number of new results among those retrieved. The main graph to consider 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 x collections.

From the *cumulative* plot in Figure 11, *Oracular* clearly performs as expected, retrieving the bulk of the results in the first few collection calls. Notice that the *dup* plot displays a counter-intuitive bell-shape curve. One would indeed expect the number of duplicates to increase as more collections are called. In fact, this is due to the specific implementation of *Oracular*: whenever the algorithm determined that none of the remaining collections could provide any new result, the result retrieval process halted and the useless collections were not called. Hence the number of duplicates shown by the plot actually drops. Had the useless collections been called despite their void contribution, the number of duplicates would have increased sharply towards the end of the plot, as one would expect. The same explanation is valid for the *results* plot. The average number of results would not drop as it does had the useless collections been called.

4.4.2. Testing the CORI Approach.

We have said previously that in order to correctly assess the benefits of our approach, we would compare its performance to that of the CORI collection selection system on the same test bed with the same set of queries. In order for the CORI approach to function properly, it is necessary to have some very specific statistics about each collection, including collection and document frequencies for each term, and word counts for each collection. Fortunately the search engine built on top of each artificial collection enabled us to have easy access to those numbers. However, these statistics were not available for the real collections. To solve this problem, the documents retrieved during the initial probing phase (with 1,062 queries) were considered to be a representative collection sample. Following the type of query-sampling approach suggested in [22], the necessary collection statistics were then estimated from that sample.

Figure 12 illustrates the overall performance of CORI, averaged over the set of test queries. The CORI plots are naturally in sharp contrast with the *Oracular* plots. When considering the *cumulative* plot, CORI actually achieves reasonably good performance, although far from its performance in non-overlapping environments. The plot shows a surprisingly close-to-constant rate of increase in the cumulative number of new results. This can also be seen by looking at the *new* plot, which is relatively stable in the vicinity of 8 new results every time a collection is called. Similarly, the *dup* plot seems to stay in the vicinity of 7 to 8. This essentially means that every time a collection is called, CORI will retrieve almost as many new results as there are duplicates, which could certainly be expensive and inefficient. Also of interest, CORI seems to be calling first collections which return more results, as the *results* plot has a general descending trend. This can be explained by the CORI algorithm itself. Intuitively, the inverse collection frequency weight

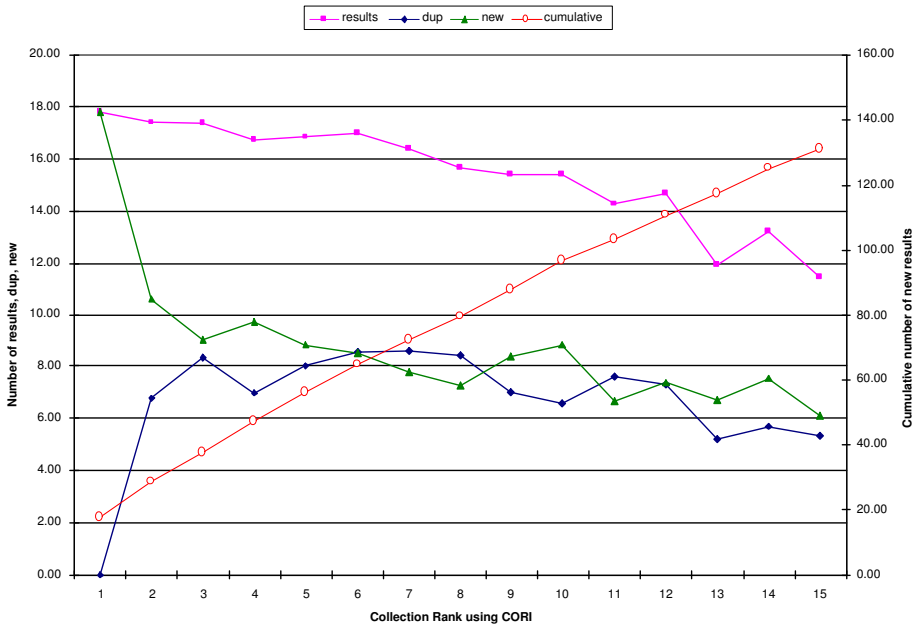

Figure 12. Performance of CORI on the 15-collection test bed.

used by CORI has much less importance in this scenario of overlapping collections, as most if not all of the collections will contain each term. Therefore, the main factor contributing to the score of a collection is the document frequency of the query terms in that collection, which tends to increase as the collections get larger.

4.4.3. Testing COSCO.

The collection selection approach presented in this thesis was tested following the same procedure as described previously. Figure 13 shows how COSCO performed against the test bed and using the statistics computed by the offline component. The graphs for COSCO clearly show an improvement over CORI. 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

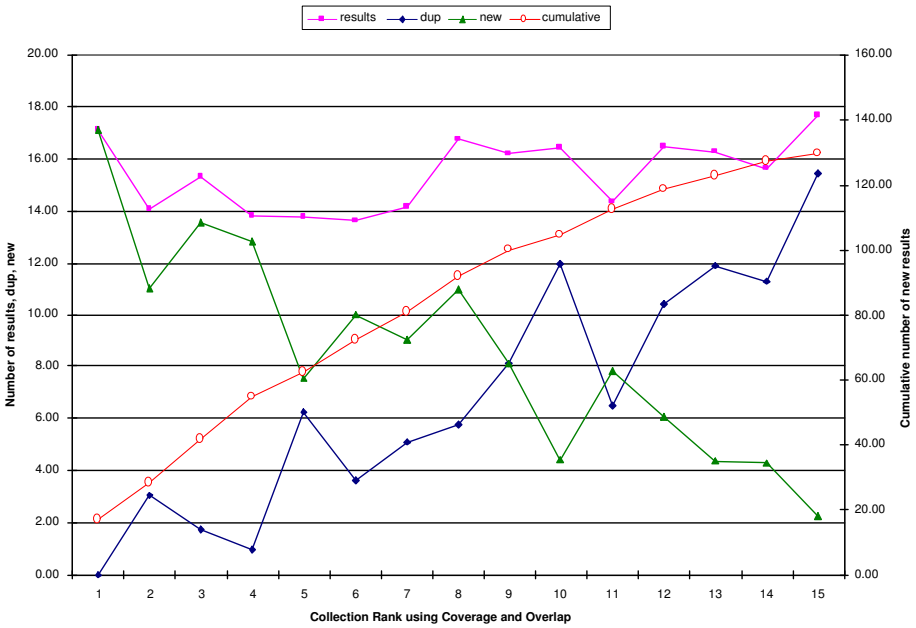

Figure 13. Performance of COSCO on the 15-collection test bed.

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, as seen in the *Oracular* system performance. The irregularities observed in the *dup* and *new* plots are due to the approximation made in our approach and to the fact that we are only storing statistics about coverage and overlap in place of exact and fully descriptive data. These irregularities point to the fact that our system cannot always determine which collection has the largest number of new results.

In addition, the *results* plot is in contrast with the *results* plot obtained with CORI. It shows that COSCO will prefer calling smaller collections with greater potential for new results than simply larger collections as CORI seems to be doing. 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. A more comparative analysis between COSCO and CORI is given in Section 4.4.5.

4.4.4. Ablation Studies on Overlap Statistics.

In evaluating our system, we were also curious to find out to what extent the overlap statistics benefited the collection selection strategy. Therefore in addition to our main solution, discussed above, we also tested 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. Figure 14 shows the performance of that approach. It actually performs quite well through the first collections. Unfortunately it fails to capture a large number of new results before the very last collection call. This behavior can be explained by the fact that some collections in the test bed, such as *csb* and *citeseer* for example, often had a relatively low coverage compared to the other collections and therefore would not be called early even though they actually returned a large number of new results. This is precisely what the use of overlap statistics compensates for, as we have seen previously.

4.4.5. Comparison of the Various Approaches.

Figure 15 summarizes the previous analysis by combining the cumulative plots of each of the four approaches we have discussed so far: *Oracular*, CORI, COSCO, and the coverage-only approach. *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

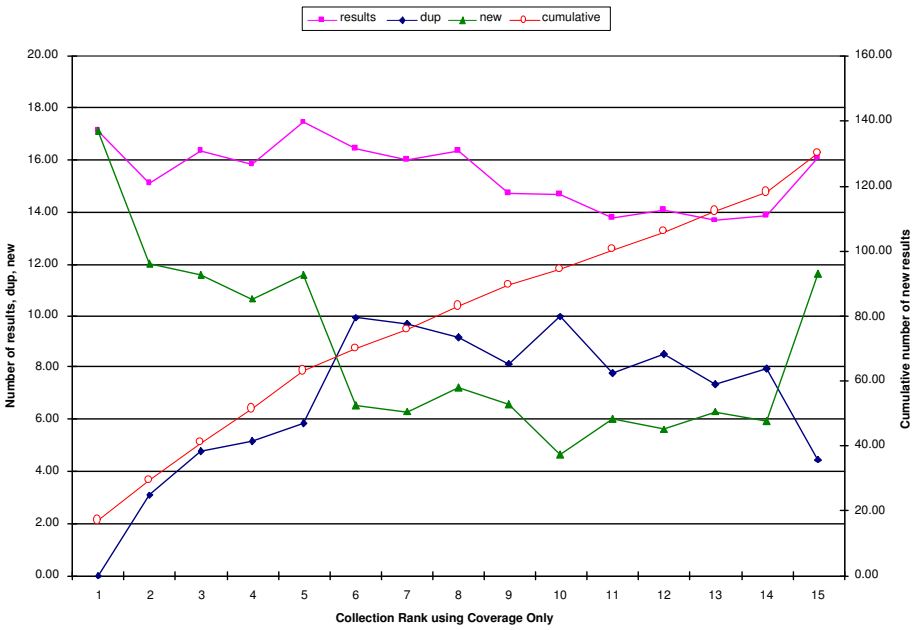

Figure 14. Performance of a variation of our approach using only coverage statistics, on the 15-collection test bed.

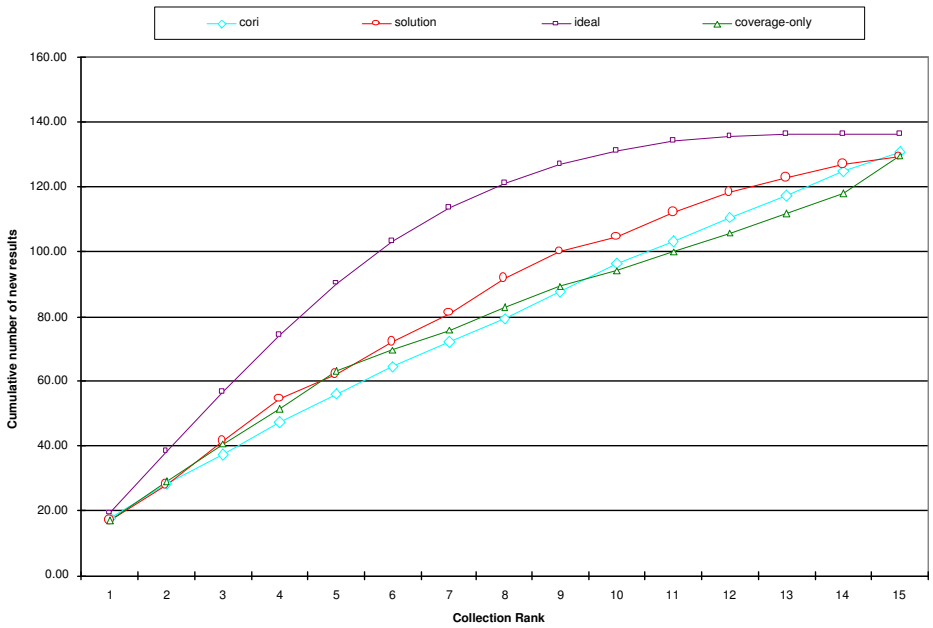

Figure 15. Performance of *Oracular*, CORI, COSCO, and a variation of our approach on the 15-collection test bed.

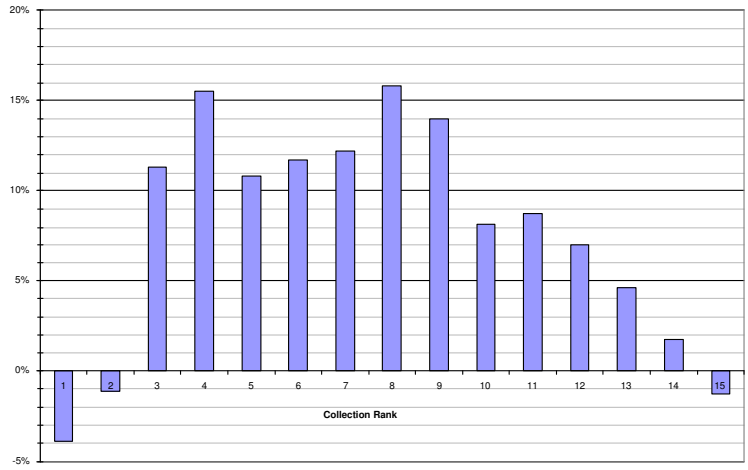

Figure 16. Percentage difference between the cumulative number of new results retrieved by CORI and by COSCO.

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

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. These numbers are naturally dependent on the query test set and the nature of the test bed, but nevertheless they probably give an idea of how these systems would perform against each other in a larger environment. 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.

Figure 16 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 new results than CORI, in the same number of collections.

Interestingly, 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.

Note that the negative percentage obtained for the last collection is due to another reason. This is caused by the fact that COSCO may sometimes determine from the statistics computed for the query that a particular collection has zero coverage, in which case we would not call this collection even though it may in fact contain some new results. This leads to a total number of new results that could be slightly inferior to CORI's total.

4.4.6. Effect of the Granularity of Statistics.

Finally, we also wanted to determine how the number and granularity of the statistics stored influenced the quality of the collection selection. To that end we ran the same experiments using our approach with the only difference that the frequency threshold for the minimum support for the item sets was set to 0.03%, instead of 0.05%. When setting the threshold to 0.03% (i.e. an item set is considered frequent if it appears in 6 or more queries), the

computation took approximately 15 minutes and resulted in 5,791 frequent item sets. A more detailed distribution of the frequent item sets found with both thresholds is given in Table 4. Logically, the lower threshold should lead to more frequent item sets being

Item Set Size	Frequency Threshold 0.05%	0.03%
1	412	647
2	205	639
3	50	634
4	12	831
5	2	996
6	0	942
7	0	662
8	0	330
9	0	110
Total	681	5791

Table 4. Frequent item sets mined with different support thresholds.

identified, and thus more statistics being stored. The intuition behind this approach was that with more frequent item sets the system might be able to approximate better the statistics for the new incoming queries.

Unfortunately the experiments did not confirm the intuition, as shown in Figure 17. The performance is approximately the same as when fewer item set statistics are stored. Of the 15 data points for cumulative results, 7 show a slight improvement over the performance using the 0.05% threshold while the remaining 8 data points show a slight decrease in performance. These results are somewhat counter-intuitive and could be explained by looking more closely at the way the test queries were mapped to item sets. With the 0.05% threshold, exactly 52 queries (of 106) were mapped to item sets for which statistics were stored. With more statistics, the number of mapped queries was 56, which is not significantly more, and which would thus hardly improve the performance of the system since the

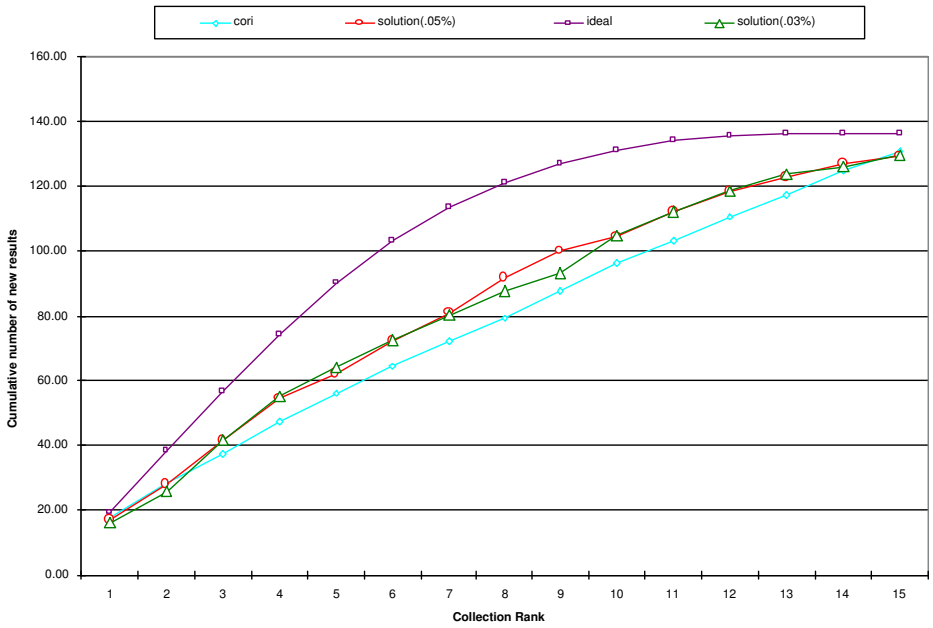

Figure 17. Performance of *Oracular*, CORI, our approach with frequency threshold 0.03%, and our approach with threshold 0.05%, on the 15-collection test bed.

additional statistics do not help to map more queries.

4.4.7. Effects of Query Distribution.

It is interesting to realize 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 106 queries used to test our system did not have the same distribution as the training queries. In fact, as mentioned previously, only approximately 50% of the test queries were actually mapped to one or more item sets for which statistics were stored. With the lower support threshold for frequent item sets, 55% of the test queries were mapped.

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. Next, we briefly describe the experimental setup for this scenario and analyze the results obtained.

4.4.7.1. *Setup for additional experiments.*

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. In other words, 19,425 queries were asked, among which some were identical. 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. The randomization in the selection ensured that the relative distribution of queries

was the same in both the test set and training set. Using this new set of training queries, the offline component accomplished its three main tasks and finally stored coverage and overlap statistics for the frequent item sets present in the training set. The number of frequent item sets identified was 621 and 1.2MB were required to store their corresponding statistics. Finally, the test set was processed by our online component, as explained in previous sections of this thesis.

4.4.7.2. *Performance of COSCO with improved distribution of test queries.*

With the new experimental setup, approximately 90% of the test queries were successfully mapped to an item set for which the offline component had stored statistics. Figure 18 shows how COSCO performed against the new test bed and using the newly computed coverage and overlap statistics. As before, the four graphs are *results*, *dup*, *new*, and *cumulative*. The results are somewhat similar to those obtained previously, but there are significant improvements in some aspects which confirm our intuition. COSCO again ensures that the number of new results retrieved is larger than the number of duplicates in the first few collections being accessed, and it does so through the first 8 collections in the ordering. The most noticeable difference with the results obtained with the first query test set (shown in Figure 13) appears in the *new* plot of Figure 18, which offers a greatly improved regularity in its descending trend. Recall that the *new* plot with the first test set had a descending trend but showed many irregularities. The improved regularity demonstrates that when our initial assumption – stating that future queries will have similar distribution to past queries – is true, our system behaves in a quite desirable way: at any point in the collection order, it almost consistently chooses the collection which offers the most new results, which was not the case in our previous experiments.

Figure 19 illustrates the performance of COSCO compared to the performance of

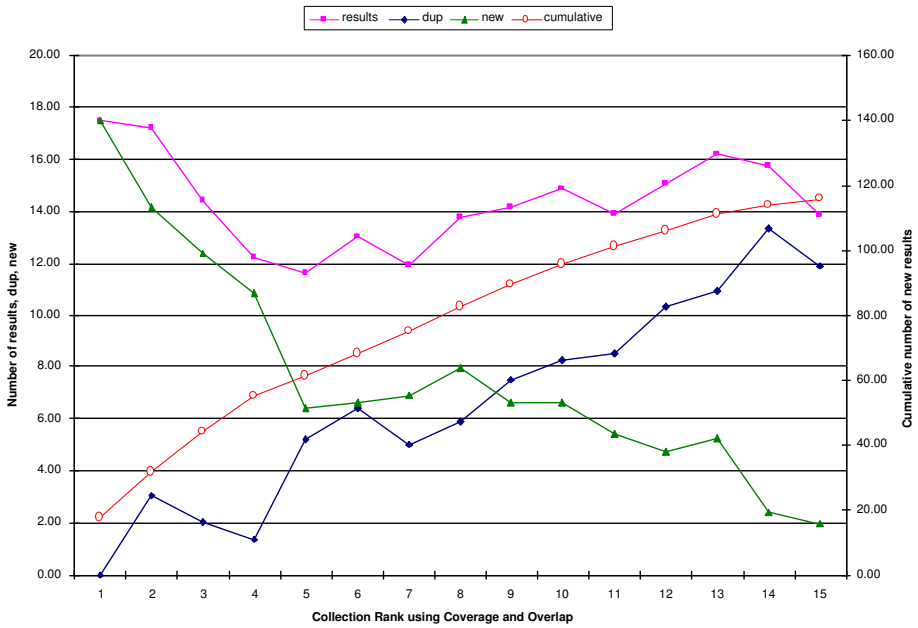

Figure 18. Performance of COSCO on the 15-collection test bed using a query test set with a similar distribution as the training set.

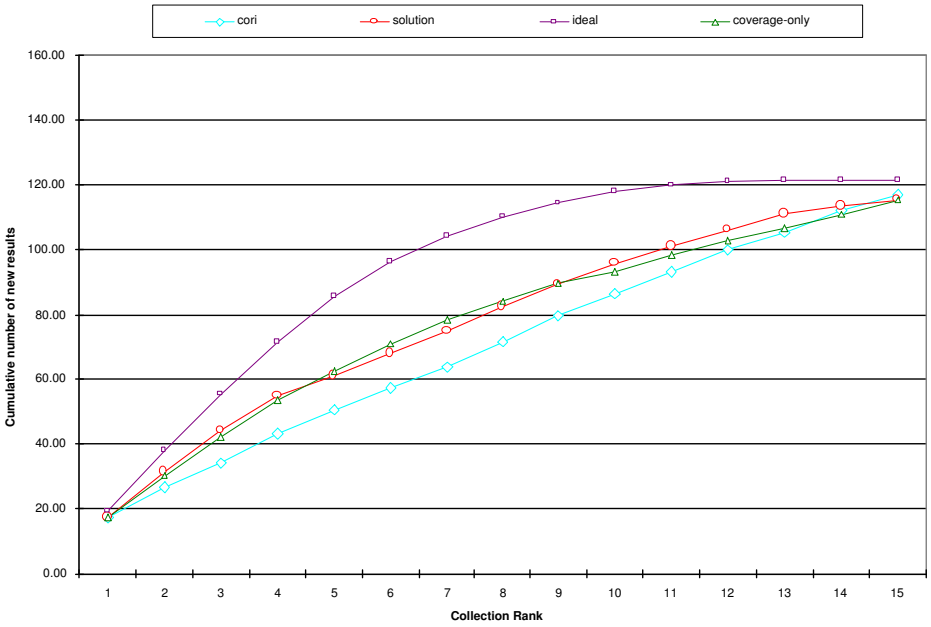

Figure 19. Performance of *Oracular*, CORI, COSCO, and the coverage-only variation of our approach on the 15-collection test bed using a query test set with a similar distribution as the training set.

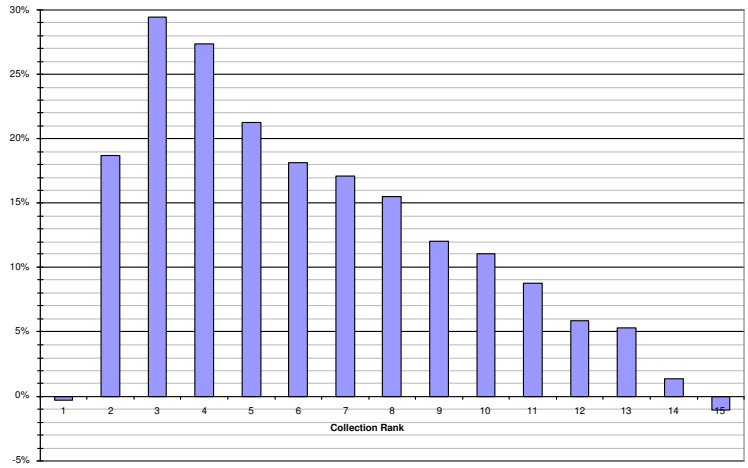

Figure 20. Percentage difference between the cumulative number of new results retrieved by CORI and by COSCO using a query test set with a similar distribution as the training set.

Oracular, CORI, and the coverage-only variation of our approach. With the new set of experiments, COSCO stills outperforms CORI throughout the entire 15-collection ordering. Naturally, *Oracular* is still better than all other approaches, as one would expect. Notice that with these additional experiments, the coverage-only approach shows a performance close to that of our main approach and also does better than CORI.

Figure 20 is perhaps more useful to realize how much better COSCO performs when being compared to CORI. 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 20 actually shows that our system performs even better in the new experiments. 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. Note that only when calling the last two collections does our system retrieve less

than 5% more new results. This decrease towards the end of the ordering is expected since when both approaches are about to call all collections, then naturally the total number of new results converges to the same value. The negative percentage seen for the first collection and last collection appear for the same reasons mentioned in Section 4.4.5. It is interesting to note, though, that unlike the results obtained previously, COSCO performs better than CORI immediately after the first collection, and that the small negative percentage obtained for the first collection corresponds to the difference between an average of 17.54 results for CORI and 17.48 results for our approach, which is clearly not a significant difference.

In conclusion, these additional experiments with a test query distribution reflecting the training queries demonstrated that our system does indeed perform significantly better than CORI. More importantly, our 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.

CHAPTER 5

Discussion and Future Work

The discussion and relevant extensions mentioned in this section relate to the approximation used in our collection selection approach. As explained in this thesis, the approximation relies on two concepts:

1. Using only pairwise overlaps,

2. Using result-set documents for overlap computation.

Each of these approximations could be addressed separately to try to improve the overall performance of the system.

5.1. Capturing Overlap Between More than Two Collections

Multi-collection overlap estimates would probably improve the collection order selection. In fact, the problem which arises from using only pairwise overlaps, as done in COSCO, is the following: when considering the next collection to add to the ordering, any overlap that the additional collection may have with the collections selected earlier may be accounted for several times, thereby wrongly estimating the total overlap between the new collection and the already selected collections. This in turn could have our system wrongly

underestimate the benefit of calling a specific collection. This is the price our approach must pay in order to keep the statistics computation and storage low.

A possible extension for our approach would thus be to compute and use overlap statistics for all possible collection sets, determine whether this improves the collection selection process, and if so analyze to what extent. Of course, any benefit of this strategy would have to be weighted against the exponential cost – in terms of the number of collections – of computing all these overlap statistics.

Not only would that cost be exponential, but it would also depend on how difficult it is to evaluate the level of overlap between n collections. As mentioned in Section 3.2, there is no trivial way of estimating the overlap between more than two collections. The fact that we are dealing with a text domain and hence similarity of documents – as opposed to equality of tuples – turns the multi-collection overlap computation into a difficult and costly problem. Recall the brief example from Section 3.2.1 where some results in C_1 overlapping with results in C_2 could overlap with results in C_3 even though there is no overlap between C_2 and C_3. In such a case, what would be an efficient yet useful way of quantifying the overlap? As the number of collections being considered grows, this type of problem will naturally occur more frequently.

5.2. Computing Result-level Overlap

Our overlap statistics are actually approximations of the true overlap between collections. The approximation comes from using the result-set document of each collection for a particular query, instead of the actual results. The assumption was that the overlap between the result-set documents of two collections could effectively represent the "true" overlap between both collections.

In fact, a more precise overlap estimation would consider the overlap between results taken independently of each other, and then estimate the overall overlap between the collections. Result-level overlap would require each result-set document (one per query for each collection) to be decomposed into individual result documents. The overlap computation between two collections for a particular query would then need to be performed on a result-to-result basis. The similarity of each pair of results would be computed, and similarities above a certain threshold would represent overlap. The overlap between two collections would thus be the number of returned documents that were *highly similar*. However this could lead to cases where a single result from a collection is similar to several results from the second collection, as was shown in Figure 4. This situation could be addressed by ensuring that a document is involved in only one overlapping pair, possibly the one with greatest similarity.

As with the previous proposed extension, the advantage of estimating and using result-level overlap would have to be weighted against the added cost of computation. Furthermore, it is worth noting that combining both this proposed extension and the previous one – thus aiming at multi-collection result-level overlaps – could definitely lead to more accurate statistics but would most likely be a major challenge in terms of computation.

CHAPTER 6

Conclusion

This thesis has addressed the issue of collection selection for information retrieval in an environment composed of overlapping collections. It was shown that most of the collection selection approaches which have been proposed so far are not designed to effectively handle significant overlap between collections. These techniques would automatically select collections by the relevance of their content even though some collections would not provide any new results given those that had already been retrieved. In fact, the previous approaches generally assumed that the collections formed a perfect partition of all available documents, which works well for the specific strategies used, but which is certainly not a realistic assumption in the domain of Internet text collections and search engines.

We thus presented COSCO, a selection approach which took into consideration the overlap between collections before determining which collection ought to be called next. The strategy consisted in having an online and offline component. The later of the two was responsible for computing and storing coverage and overlap statistics with respect to frequently asked keyword sets. When a new query was asked, the online component would then attempt to map the query to a set of frequent sets and their corresponding statistics would then be used to compute the coverage and overlap statistics of the new query. Finally, with these estimated query statistics in hand, the collections were selected by first picking

the one with highest coverage, and then iteratively choosing the one with least overlap with the collections previously selected.

Experiments were performed with the intention of showing how COSCO would perform with a set of overlapping collections. The results of the experiments demonstrate that on the same set of collections and test queries, our approach achieves better results than CORI, the system usually picked as being the most reliable of the current collection selection approaches. When directly compared to CORI, COSCO lead 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.

REFERENCES

[1] *BibFinder: A Computer Science Bibliography Mediator*, http://rakaposhi.eas.asu.edu/bibfinder, 2004.

[2] *CiteSeer – Computer and Information Science Papers*, http://www.citeseer.org, 2004.

[3] *CNN*, http://www.cnn.com, 2004.

[4] *Columbia Encyclopedia*, http://www.bartleby.com/65, 2004.

[5] *Compendex Database*, http://www.engineeringvillage2.org, 2004.

[6] *DBLP – Computer Science Bibliography*, http://www.informatik.uni-trier.de/~ley/db, 2004.

[7] *Encyclopedia Britannica Online*, http://www.eb.com, 2004.

[8] *Google*, http://www.google.com, 2004.

[9] *Google News (BETA)*, http://news.google.com, 2004.

[10] *IEEE Xplore*, http://ieeexplore.ieee.org, 2004.

[11] *Jakarta Lucene. The Apache Jakarta Project*, http://jakarta.apache.org/lucene, 2004.

[12] *Network Bibliography*, http://www.cs.columbia.edu/~hgs/netbib, 2004.

[13] *ScienceDirect*, http://www.sciencedirect.com, 2004.

[14] *The ACM Digital Library*, http://www.acm.org/dl, 2004.

[15] *The ACM Guide to Computing Literature*, `http://www.acm.org/guide`, 2004.

[16] *The Collection of Computer Science Bibliographies*, `http://liinwww.ira.uka.de/-bibliography`, 2004.

[17] *The New York Times*, `http://www.nytimes.com`, 2004.

[18] *Yahoo!*, `http://www.yahoo.com`, 2004.

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

[20] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern information retrieval*, ACM Press / Addison-Wesley, 1999.

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

[22] James Callan and Margaret Connell, *Query-based sampling of text databases*, Information Systems **19** (2001), no. 2, 97–130.

[23] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe, *Collection statistics for fast duplicate document detection*, ACM Transactions on Information Systems **20** (2002), no. 2, 171–191.

[24] Jack G. Conrad and Joanne R. S. Claussen, *Early user–system interaction for database selection in massive domain-specific online environments*, ACM Transactions on Information Systems **21** (2003), no. 1, 94–131.

[25] James C. French, Allison L. Powell, James P. Callan, Charles L. Viles, Travis Emmitt, Kevin J. Prey, and Yun Mou, *Comparing the performance of database selection algorithms*, Proceedings of ACM SIGIR Conference, 1999, pp. 238–245.

[26] Michael R. Garey and David S. Johnson, *Computers and intractability: A guide to the theory of np-completeness*, W. H. Freeman & Co., 1979.

[27] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic, *The effectiveness of GlOSS for the text database discovery problem*, Proceedings of ACM SIGMOD Conference, 1994, pp. 126–137.

[28] Luis Gravano, Héctor García-Molina, and Anthony Tomasic, *GlOSS: text-source discovery over the Internet*, ACM Transactions on Database Systems **24** (1999), no. 2, 229–264.

[29] Taher Haveliwala, Aristides Gionis, Dan Klein, and Piotr Indyk, *Evaluating strategies for similarity search on the web*, Proceedings of the World Wide Web Conference, 2002.

[30] Adele E. Howe and Daniel Dreilinger, *SAVVYSEARCH: A metasearch engine that learns which search engines to query*, AI Magazine **18** (1997), no. 2, 19–25.

[31] Juraj Hromkovic, *Algorithmics for hard problems*, Springer-Verlag New York, Inc., 2003.

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

[33] Bernard J. Jansen and Udo Pooch, *A review of web searching studies and a framework for future research*, Journal of the American Society for Information Science and Technology **52** (2001), no. 3, 235–246.

[34] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic, *Real life information retrieval: a study of user queries on the web*, SIGIR Forum **32** (1998), no. 1, 5–17.

[35] John King and Yuefeng Li, *Web based collection selection using singular value decomposition*, Proceedings of the International Conference on Web Intelligence, 2003.

[36] King-Lup Liu, Clement Yu, Weiyi Meng, Wensheng Wu, and Naphtali Rishe, *A statistical method for estimating the usefulness of text databases*, IEEE Transactions on Knowledge and Data Engineering **14** (2002), no. 6, 1422–1437.

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

[38] Weiyi Meng, Clement Yu, and King-Lup Liu, *Building efficient and effective metasearch engines*, ACM Computing Surveys **34** (2002), no. 1, 48–89.

[39] P. Mork, A. Halevy, and P. Tarczy-Hornoch, *A model for data integration systems of biomedical data applied to online genetic databases*, Proceedings of the Symposium of the American Medical Informatics Association, 2001.

[40] Zaiqing Nie and Subbarao Kambhampati, *Frequency-based coverage statistics mining for data integration*, IJCAI Workshop on Information Integration on the Web, 2003.

[41] ———, *A frequency-based approach for mining coverage statistics in data integration*, Proceedings of the International Conference on Data Engineering, 2004.

[42] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank citation ranking: Bringing order to the web*, Tech. report, Stanford Digital Library Technologies Project, Stanford University, 1998.

[43] Allison L. Powell and James C. French, *Comparing the performance of collection selection algorithms*, ACM Transactions on Information Systems **21** (2003), no. 4, 412–456.

[44] R. Shaker, P. Mork, J. S. Brockenbrough, L. Donelson, and P. Tarczy-Hornoch, *The BioMediator system as a tool for integrating biologic databases on the web*, Proceedings of the IJCAI Workshop on Information Integration on the Web, 2004.

[45] Narayanan Shivakumar and Héctor García-Molina, *SCAM: A copy detection mechanism for digital documents*, Proceedings of the Conference on the Theory and Practice of Digital Libraries, 1995.

[46] Narayanan Shivakumar and Héctor García-Molina, *Finding near-replicas of documents on the web*, Proceedings of WebDB, 1999.

[47] Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael Moricz, *Analysis of a very large AltaVista query log*, Tech. Report 1998-014, Digital SRC, 1998.

[48] Ellen M. Voorhees, Narendra Kumar Gupta, and Ben Johnson-Laird, *The collection fusion problem*, Text REtrieval Conference, TREC, 1994.

[49] Ellen M. Voorhees and Richard M. Tong, *Multiple search engines in database merging*, Proceedings of the Second ACM International Conference on Digital Libraries, 1997, pp. 93–102.

[50] Zonghuan Wu, Weiyi Meng, Clement Yu, and Zhuogang Li, *Towards a highly-scalable and effective metasearch engine*, Proceedings of the World Wide Web Conference, 2001, pp. 386–395.

[51] Ramana Yerneni, Felix Naumann, and Hector Garcia-Molina., *Maximizing coverage of mediated web queries*, Tech. report, Stanford University, 2000.

[52] Clement T. Yu, King-Lup Liu, Wensheng Wu, Weiyi Meng, and Naphtali Rishe, *Finding the most similar documents across multiple text databases*, Advances in Digital Libraries, 1999, pp. 150–162.

[53] Budi Yuwono and Dik Lun Lee, *Server ranking for distributed text retrieval systems on the internet*, Database Systems for Advanced Applications, 1997, pp. 41–50.