

A Thesis by **John King**, B.I.T.

Deep Web Collection Selection

In Fulfillment
of the Requirements for the Degree
Master of Information Technology

School of Software Engineering and Data Communications
Faculty of Information Technology
Queensland University of Technology, Brisbane
December 2004

Copyright © John King, MMIV. All rights reserved.

j5.king@qut.edu.au

The author hereby grants permission to the *Queensland University of Technology, Brisbane* to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Keywords

information retrieval, deep web, collection selection, singular value decomposition, latent semantic analysis, sampling, query focused, probabilistic

Deep Web Collection Selection

by
John King

Abstract

The deep web contains a massive number of collections that are mostly invisible to search engines. These collections often contain high-quality, structured information that cannot be crawled using traditional methods.

An important problem is selecting which of these collections to search. Automatic collection selection methods try to solve this problem by suggesting the best subset of deep web collections to search based on a query.

A few methods for deep Web collection selection have proposed in Collection Retrieval Inference Network system and Glossary of Servers Server system.

The drawback in these methods is that they require communication between the search broker and the collections, and need metadata about each collection.

This thesis compares three different sampling methods that do not require communication with the broker or metadata about each collection. It also transforms some traditional information retrieval based techniques to this area. In addition, the thesis tests these techniques using INEX collection for total 18 collections (including 12232 XML documents) and total 36 queries.

The experiment shows that the performance of sample-based technique is satisfactory in average.

Contents

List of Figures	iii
List of Tables	iv
Introduction	1
1 Motivations and Definitions	6
1.1 Motivations	6
1.2 Deep Web	7
1.3 Information Retrieval Terms	8
1.4 Collection Selection Metrics	11
1.5 Latent Semantic Analysis	12
2 Distributed Information Retrieval	14
2.1 CORI	14
2.2 GIOSS	15
2.3 bGIOSS and vGIOSS	15
2.4 Lightweight Probes	16
2.5 Incremental Probe Queries	17
2.6 Collection selection methods comparison	17
3 Sampling Technique for the Deep Web	18
4 Singular Value Decomposition Based Sampling Method	20
4.1 Definitions	20
4.1.1 Transpose	20
4.1.2 Identity Matrix	20
4.1.3 Orthogonal	21
4.1.4 Determinants	21

4.1.5	Eigenvectors and Eigenvalues	22
4.1.6	Gram-Schmidt Orthonormalisation	23
4.2	Singular Value Decomposition	23
4.2.1	Properties of Singular Value Decomposition	23
4.2.2	Singular Value Decomposition Example	25
4.3	The Meaning Of Singular Value Decomposition	29
4.4	Calculation of Singular Value Decompositions in Matlab	30
5	Algorithms and Experiments	31
5.1	Introduction	31
5.2	Testbed	32
5.2.1	Document Set	32
5.2.2	GP-XOR	33
5.2.3	Sampling	35
5.3	The algorithms considered	35
5.3.1	Baseline Distributed Retrieval	36
5.3.2	Query Focused Probabilistic Self Recovery	38
5.3.3	Singular Value Decomposition	42
6	Collection Selection Scoring	44
7	Results Analysis	48
7.1	Cumulative Distance Measure	48
7.2	Average Precision Measurement	52
7.3	Conclusion	56
8	Discussion and Conclusion	57
	Bibliography	62

List of Figures

1	Distributed Information Retrieval Process	3
1.1	Disjoint and Overlapping Collections	11
4.1	Matrix Decomposition	23
5.1	Collection Selection Using A Search Broker	32
5.2	Size of INEX Collections in Megabytes	34
5.3	Number of files in each INEX Collection	35
5.4	Baseline Collection Selection	37
5.5	Probabilistic Collection Selection	40
5.6	Sampled Singular Value Decomposition	42
6.1	Collection Selection Scoring	47
7.1	Cumulative Distance Measure For Exhaustiveness	51
7.2	Cumulative Distance Measure For Specificity	51
7.3	Average Precision	56

List of Tables

4.1	The Term-Frequency Table	25
5.1	The INEX Collections Names and Their Abbreviations	33
5.2	INEX collections, the number of files in each collection, and their sizes in megabytes	34
7.1	Sample comparison of the Topic 91 results	49
7.2	Distance measurements from exhaustiveness compared to probabilistic, baseline and SVD methods	50
7.3	Cumulative Distance Measure for Exhaustiveness	52
7.4	Precision measurements for Top 3 Topic 91 Results compared to Exhaustiveness	54
7.5	Exhaustiveness Precision Average Across 18 collections	55

Statement of Original Authorship

The work contained in this thesis has not been previously submitted for a degree or diploma at any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

John King

December 2004

Acknowledgments

Many thanks to my supervisor Dr. Yuefeng Li for his help, ideas, and constant support, without which I could never have completed this work. Thanks to my associate supervisor Dr. Shlomo Geva for his comments and suggestions, to Dr Frederic Maire for his comments, and to Michael Gardner for proofreading and comments. Thanks to CITI for funding my WI2003 conference trip, stationary, and coffee.

Introduction

The web can be broken into two major areas, the *surface web* and the *deep web*. The surface web consists of a massive interlinked collection of electronic information which can be indexed using computer programs that recursively follow these links. These programs, known as *spiders*, create centralised indexes of the information which can be searched with search engines. Surprisingly, much of the Web is not accessible to spiders, simply because the content is not directly linked. This part of the web is referred to as the *Deep Web*. Many deep web collections can only be accessed through a search interface, and therefore cannot be indexed. In a recent study, it was estimated that the surface web contained less than 4 billion documents, while the deep web contained over 550 billion documents [HC02, Ber01]. We need to find new methods to search the deep web and extract this information.

It is estimated that there are more than 200,000 collections in the deep web [HC02, Ber01]. With so many collections available, we need to use *Collection selection* to help select best information sources to use for a query. Collection selection is commonly performed by a *search broker*, which acts as an interface between the user and the collections. The search broker takes a query and sends it to the collections, returning the results to the user. Earlier collection selection work such as CORI(Collection Retrieval Inference Network) [CLC95] and GIOSS(Glossary of Servers Server) [GGMT99] often relied on cooperation between the search broker and the collections. Meta data is gathered, term statistics are retrieved, messages passed, and software is distributed. However, in reality few deep web collections are willing to co-operate by giving detailed statistics about themselves, allowing full access to their data, or allowing software to be run on their servers. The methods compared in this thesis do not require co-operation between the search broker and the collections.

The tasks performed by search brokers is commonly referred to as *Distributed Information Retrieval*, which is the searching of multiple servers and the merging of the results. There are two components in a traditional Distributed Information Retrieval system, *Collection Selection* and *Collection Fusion*.

Collection Selection is the selection of an optimal subset of collections from a large set

of collections for the purpose of reducing costs associated with Distributed Information Retrieval [CLC95, FPC⁺99, HT99, LCC96, CBH00, DTZ00, GGMT99, MLY⁺99, GGM95, CC00]. The central goal of collection selection is to make searching multiple collections appear as seamless as searching a single collection [VGJL95]. Another requirement of a collection selection system is to learn which collections contain relevant information. This reduces the number of overall search requests needed. If only a small, high quality subset of the available collections is searched then savings can be made in time, bandwidth, and computation.

Collection Fusion is the merging of a number of sets of documents into a single list ordered by relevance to a query [Li01]. Well executed collection fusion can give better results than a single representation scheme or retrieval system [TL01]. We will not use collection fusion in this thesis.

In a recent paper, Tsirikika et al [TL01] separated the Distributed Information Retrieval process into three stages, including *Document Selection* as a new step between collection selection and collection fusion. *Document selection* involves selecting a set of documents from each collection. This can be as simple as selecting an arbitrary number of document from each collection, or alternatively selecting documents based on a relevancy score. Intuitively, the more relevant the collection is, the larger the number of documents that should be taken from it.

Figure 1 shows the Distributed Information Retrieval process. A query is taken from the user, the collections relevant to the query are selected using *collection selection*, the number of documents to take from each collection is calculated using *document selection*, the documents taken from each selected collection are sorted using *collection fusion*, and the results are returned to the user as a set of documents ordered by relevance to the query.

This thesis will only discuss Collection Selection, leaving Document Selection and Collection Fusion for future work.

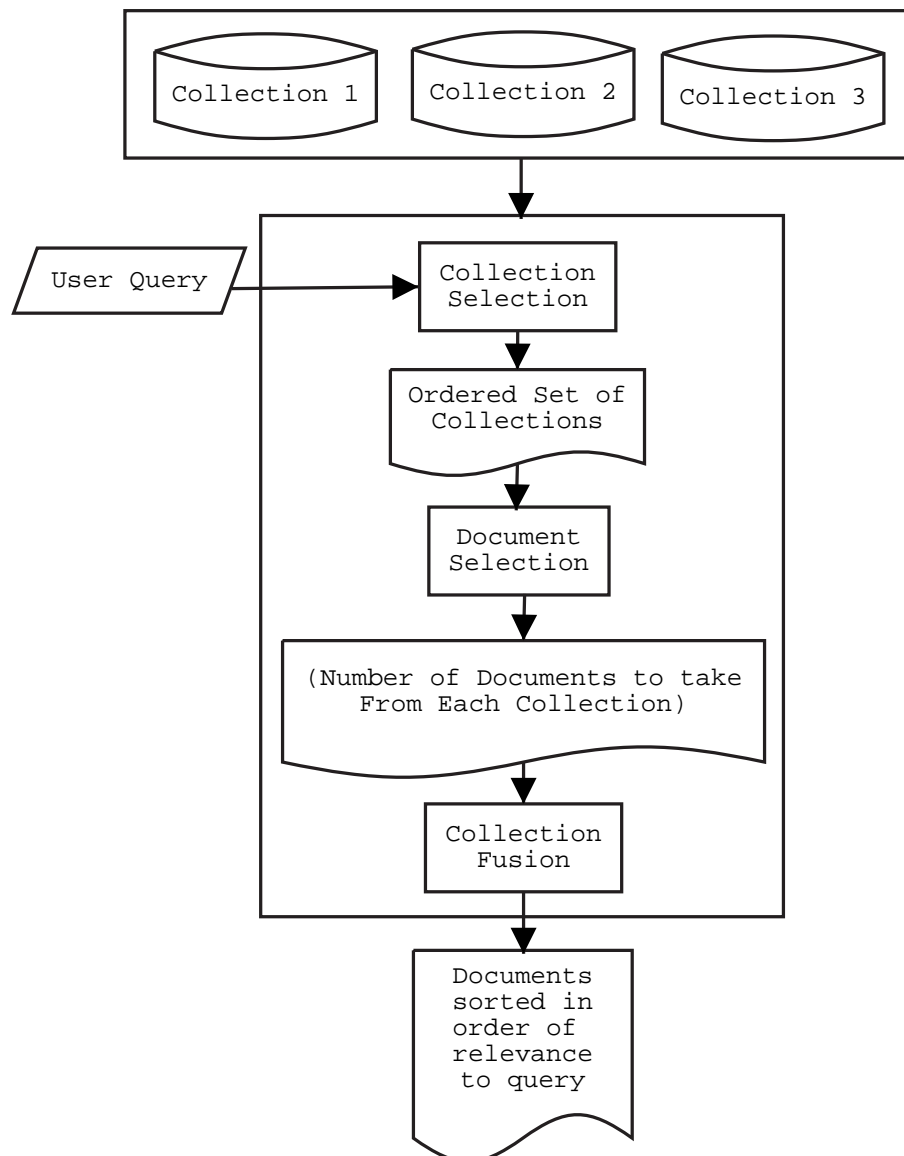


Figure 1: Distributed Information Retrieval Process

There are a number of differences between traditional and deep web collection selection. In deep web collection selection it is difficult to estimate the size and term statistics of collections because most deep web collections do not provide a way of finding the frequency of terms and the number of available documents. This also means that standard information retrieval metrics cannot be used with deep web collections. Most collections on the deep web are not indexable by traditional indexing methods as they only present a search interface and no way of indexing the contents. Some deep web collections also change frequently, meaning that traditional indexing methods have to be run frequently in order to keep index information valid. Deep web collection selection is typically performed using a *meta search engine*, which queries a number of deep web collections and combines the results into a single result set ordered by

relevancy. Meta search engines commonly use collection selection to improve results and cut down bandwidth usage.

Deep web collection selection is significant because deep web collections often contain high quality, structured information not accessible to modern search engines. This research will help the user select the best deep web collections for their needs. The need for deep web collection selection will keep growing as more and more collections are added to the deep web.

Common problems encountered during deep web collection selection include reducing expenses, increasing search speed, learning to adapt to change in the search environment, using ontology to increase precision/recall, and learning to adapt to the users preferences. This thesis introduces solutions to the evaluation and ranking of collections, and the using of feedback to select the best related collections for each search. The methods used do not require meta-data to be kept about each server, and do not require any of the servers to keep meta-data about themselves. Once a set of useful collections has been discovered using a small sample, the collection fusion component can then retrieve a larger sample from the smaller set of collections and still keep costs down.

There are a number of problems facing efficient deep web collection selection. These problems will be briefly addressed here and covered in more detail later in the paper. The first problem is that there are too many electronic collections available to track by hand. The massive growth of the deep web is parallel to the growth in electronic collections becoming available on the deep web. In July 2002, it was estimated that there were more than 200,000 searchable web sites on the deep web [HC02], with 550 times the size of the surface web. This is dynamic data, with constant growth and change. The second problem is that many deep web collections are not indexable by traditional methods. Only surface web documents have a hyperlinking system that supports easy indexing. The third problem is that the collection selection system needs to be able to respond quickly to user queries. This is important as collection selection is only the first part of the distributed information retrieval process. Users expect near instantaneous responses to their queries. Finally very short queries are common in deep web systems. This makes it difficult to make exact matches to documents and can produce noisypoor results.

Considering these problems, the aim for this work is to build a system that can take samples from a large number of collections and return a subset of relevant collections. An early web based collection selection system using singular value decomposition was presented in [KL03] by the author of this thesis.

A few of these collection selection problems have been addressed in previous work such as CORI [CLC95] and GLOSS [GGMT99]. CORI assumes that the best collections are the ones that contain the most documents related to the query. GLOSS uses a server which contains all

the relevant information of other collections. Users query GLOSS which then returns a ordered list of the best servers to contact to send the query to. The problem is that both these methods require communication between the search broker and the collections, making them of limited value in the deep web. In this thesis we will present a sampling technique which does not require communication between the search broker and deep web collections.

Chapter 1 describes motivations and gives definitions for some Information Retrieval terms. Chapter 2 introduces Distributed Information Retrieval. Chapter 3 introduces Deep-Web Information Retrieval. Chapter 4 covers Singular Value Decomposition. Chapter 5 describes our experiments. Chapter 6.1 shows how we evaluate our results. Chapter 7 presents our results. Chapter 8 concludes the thesis.

Chapter 1

Motivations and Definitions

This work is part of a project building a Deep Web Search Engine to search distributed deep web collections. The aim of the research is to combine the results from querying multiple deep web collections into a result such that it appears a single surface web collection has been used.

For this paper there will be some collections distributed across various servers (Distributed Infrastructure Component), and each collection will support an Information Retrieval architecture (Information Retrieval Component) that returns an ordered set of documents for each query to a Search Broker.

Section 1.1 gives some motivations for this research. Section 1.2 discusses the *deep web*. Section 1.3 introduces some terminology used in Distributed Information Retrieval.

1.1 Motivations

There are several motivations for this work. The first motivation is that the deep web is inaccessible to most search engine spiders, which means that many users are unaware of its rich content. The second is that it is difficult to index the deep web using traditional methods. The third is that because of the large number of collections on the deep web, it is too expensive to query them every time a search is performed. The fourth motivation is that because of the unconnected structure of the deep web, new search techniques must be developed to retrieve information.

The problem with current collection selection work is that it requires communication between the search broker and collections, full knowledge of each collection, or metadata about each collection. This prevents many collection selection techniques from being used in deep web collection selection

We use sampling techniques as a solution to these problems because full document statistics

are often unavailable on the deep web, and sampling does not require communication between search brokers and collections.

1.2 Deep Web

The web is composed of two main divisions, the *Surface Web* and the *Deep Web*. Currently the largest index of the surface web contains less than 5 billion documents, whereas the deep web is reported to contain more than 550 billion documents [HC02].

The *surface web* consists of hyperlinked web pages which can be easily indexed by a computer program. Each typically has outgoing links to other web pages, and incoming links which allow them to be reached from other pages, creating a spider-web like system of interconnected data. If the starting points are chosen well, it is possible to traverse most of the surface web in a relatively short time by recursively following these hyperlinks. The surface web is also highly distributed. Some search engines attempt to index the entire surface web into a centralised index (this index is also often distributed, but that is over far fewer machines for the purpose of faster search and load sharing). What makes the information on the surface web relatively easy to index is the web-like structure. Most documents point to at least one other document. There exist hubs and authorities [Kle99], which identify central, respected documents. The information from these maps of hyperlinks may also be mined, and in some systems each hyperlink is treated as a measure of popularity, and is used for off-the-page ranking of a document [BP98].

In contrast, the *deep web* consists of collections of unlinked data. In 2002 it was found that the top sixty deep web sites cumulatively contained 750 terabytes of data, while the surface web cumulatively contained only 19 terabytes [Ber01]. This data cannot be reached by spiders or found through incoming links. They most commonly occur as databases such as Oracle, Access, SQL Server, or as structured documents such as XML or PDF. These collections often contain good quality, structured, well maintained data, but cannot be easily accessed by hyperlinks. Commonly, deep web collections consist of a database which is accessible only through a search interface. These collections often have differing query interfaces, schema, and constraints. Examples of deep web content are phone directories, subject directories, patent collections, news articles, and holiday booking interfaces. Some of these collections may stay static, while other collections may change frequently.

Many Deep Web search engines/databases only permit viewing of a small number of search results. They may report many hundreds of thousands of results, but actually only allow access to a small number of these results. This means that detailed term usage statistics cannot be used and another method of finding results needs to be found. French [FPC⁺99] claimed that in that

case of no direct access to a database, it is in principle possible to build an index by issuing a single-term query for each word. This method is useful only when a very small number of results are returned. Also, this method would be very difficult to use when working with queries of two or more terms.

Another difficulty is that there are many different deep web search interfaces, some using stemming, stopwords, boolean operators, structure mining, and relevance feedback. Often the interfaces produce differently formatted output structures. Some include full information, some only include metadata or abstracts. Often extracting the results from a deep web collection requires manually configuring a screen-scraping system that will extract the important information from the results while ignoring other information such as advertisements and unrelated text. Downloading a deep web collection to a centralised index is often impossible without direct SQL access to the database behind the collection. Also the sheer size of the deep web, which is currently 550 times the size of the surface web, makes the scale of creating a centralised index impossible at present. The current solution is to attempt to cover small, specialised portions of the deep web and to do some form of *collection fusion* to merge the returned documents together.

Later in this paper we will present a sample-based collection selection solution to the deep web problem.

1.3 Information Retrieval Terms

In this section we introduce some basic definitions for deep web collection selection.

The atomic unit of Information Retrieval is the *term*. A term is a word and it can be of any language. A *query* is a term or set of terms entered by the user which provides clues to the user information need. A *document* is a generally a larger set of ordered terms, usually in sentence format. A *set of documents* are often called a *collection*. The count of how many times a term appears in a document is known as *term frequency*.

A document is often represented as a one-dimensional array. This array is also known as a *vector*. A *document vector* contains the term frequency of every word in the document. To compare short and long document vectors with each other, we often have to *weight* the term frequencies of each document vector. Weighting is also applied to increase the value of more descriptive terms. One of the most popular term weighting methods is $TF \times IDF$. *TF* stands for Term Frequency and it is the number of times that a term occurs in a document. *IDF* stands for Inverse Document Frequency, which is a measure of how common a word is in a collection. Common words will have a low *IDF* and rare words will have a high *IDF*. *TF* is multiplied by

IDF for each term in the document vector. The method for calculating *IDF* and then *TFIDF* is given as:

$$IDF = \log \frac{\text{collection size}}{\text{number of documents containing the term}} \quad (1.1)$$

$$TFIDF = TF \times IDF \quad (1.2)$$

The goal of Information Retrieval is to take a query and return a set of documents *relevant* to the query. *Relevance* is a measure of how well a document fits a user need. Relevance is difficult to quantify because different users have different information needs, and what might be relevant to one user may not be relevant to another. Some relevance judgements use a binary value, so that a score of 0 is given to an irrelevant document, and a score of 1 is given to a relevant document. Other relevance judgements use a scale, for example, assigning each document a relevance score between zero and ten.

There are a number of ways for measuring relevance. Four standard Information Retrieval measures of performance are *precision*, *recall*, *overload* and *mismatch*.

Precision is a standard IR measure of performance. It is defined as the number of relevant documents retrieved divided by the total number of documents retrieved:

$$Precision = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of documents retrieved}} \quad (1.3)$$

The goal of a IR system is to achieve 100% precision, however as this can be achieved by returning only one document the system should also try to maximise *recall*(see below). Giving more weight to common documents is one method of improving precision [NK98].

Recall is a standard IR measure of performance. It is defined as the number of relevant documents retrieved divided by the total number of relevant documents in the collection:

$$Recall = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents in the collection}} \quad (1.4)$$

The goal of a IR system is to achieve 100% recall, however as this can be achieved by returning all the documents the system should also try to maximise *precision* as well. Recall can be improved by selecting collections that retrieve different relevant documents [NK98].

Overload is where returned documents are not relevant. Overload is defined as the number of irrelevant documents retrieved divided by the total number of irrelevant documents in the collection:

$$Overload = \frac{\text{Number of irrelevant documents retrieved}}{\text{Total number of irrelevant documents in the collection}} \quad (1.5)$$

Overload is made worse if the irrelevant documents returned are highly rated.

Mismatch is where relevant documents are not retrieved from the collection. Mismatch is defined as the number of relevant documents not retrieved divided by the total number of relevant documents:

$$Mismatch = \frac{\text{Number of relevant documents not retrieved}}{\text{Total number of relevant documents in the collection}} \quad (1.6)$$

Mismatch is difficult to avoid when working with short or unspecific queries.

In order to quickly find relevant documents and collections, *indexing* is used. *Indexing* is the breaking down of a document into its term components. A collection of documents are transformed into an *inverted list*. For every term in the document, its term count and relative positions are extracted and added to the index. Words which occur frequently, *stopwords*, are discarded. A stopword is any word which has no semantic content. Common stopwords are prepositions and articles, as well as high frequency words that do not help retrieval. These words can be removed from the internal model of the query, document, or collection without causing loss of precision and recall.

There are two forms of Information Retrieval, Localised Information Retrieval and Distributed Information Retrieval. In Localised Information Retrieval, all documents reside on a single central server. In Distributed Information Retrieval, the documents are distributed across multiple servers, and the servers co-operate in order to return as many relevant documents as possible. There are several levels of cooperation, ranging from none to full. The levels of cooperation will determine how much information gets passed between the search broker and the server, and will determine the quality of results returned.

A *search broker* is the intermediary between the user and the collections in Distributed Information Retrieval. The search broker takes the query from the user, parses it, passes it to the collections in parallel, takes the results from the collections, parses them into a suitable format, and returns the results to the user. Other terms for search brokers are *meta search engine*, *broker*, or *search agent*. In some research, it is common for large amounts of information and statistics to be passed between the search broker and the collections.

Sometimes distributed collections are *overlapping*, which occurs when different collections contain some of the same information. If the collections are not overlapping, they are referred to as *disjoint*.

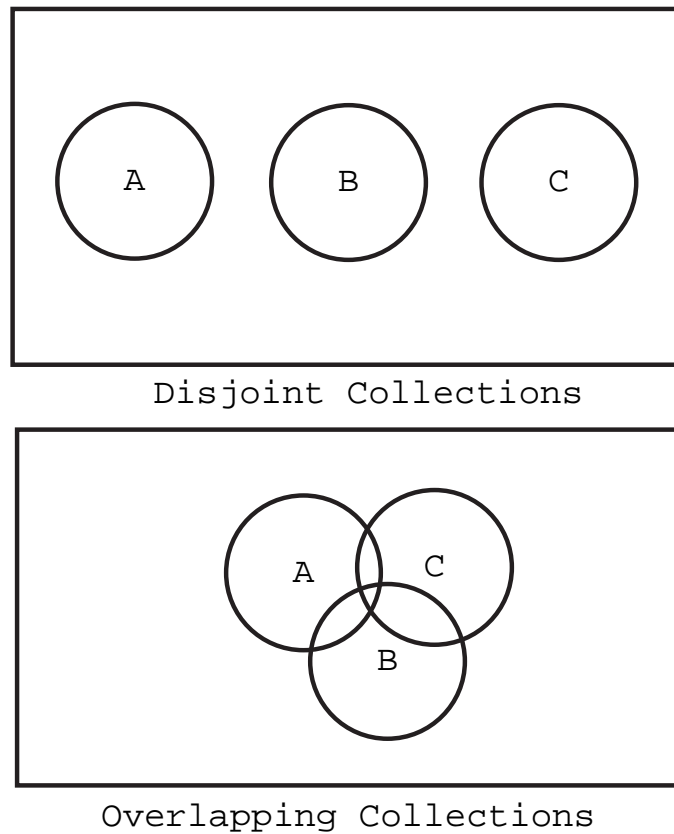


Figure 1.1: Disjoint and Overlapping Collections

Figure 1.1 illustrates the *disjoint* and *overlapping* collections. A disjoint system occurs when there are few overlapping terms among the collections. A overlapping system occurs when some terms exist among different collections.

Hyperlinks give the web its highly inter-connected web structure. Hyperlinks are pointers to other documents and collections on the web. On the web, programs called *spiders* recursively travel across hyperlinks indexing each document one by one and adding them to the inverted index. This is also known as *crawling*.

1.4 Collection Selection Metrics

Collection Selection Metrics are different to the precision and recall document selection metrics presented above.

Historically collection selection has used precision and recall measurements to evaluate the effectiveness of the collection selection technique. This collection selection metric set is an extension of the document precision and recall metrics, first mentioned in Gravano *et al* [GGM95] and most recently in Callans [?] paper. They measure the percentage of relevant collections

in relation to the number of collections retrieved, and compare the amount of relevance in all the collections with the relevance in the top n collections for a query.

The metrics are presented below.

$$Precision_n = \frac{\sum_{i=1}^n \begin{cases} 1 & \text{if } NumRel(db_{ei}) > 0 \\ 0 & \text{otherwise} \end{cases}}{n}$$

$$Recall_n = \frac{\sum_{i=1}^n NumRel(db_{ei})}{\sum_{i=1}^n NumRel(db_{bi})}$$

where

- e is the estimated collection ranking
- b is the corresponding relevance based collection ranking (*baseline*)
- n is the rank
- db_{ji} is the i^{th} collection of collection ranking j
- $NumRel(db_{ji})$ is the number of relevant documents in db_{ji}
- db is the collection

Unfortunately, these metrics are of use only when dealing with small collections, with each collection partitioned into distinct categories. For large collections, we will use two measures: precision and cumulative distance. The former is the definition discussed above, the later is a new definition used in this research. The details can be found in Chapter 7.

1.5 Latent Semantic Analysis

Latent Semantic Analysis(LSA) is a vector space model for analysing relationships between documents. Developed by Deerwester [DDL⁺90] and Berry [BDJ99], it is a statistical method of reducing the dimensionality of a matrix and associating related concepts together. Used with a term-document matrix, Latent Semantic Analysis takes the background structure of word usage and removes the noise. This noise reduction allows the higher order relationship between terms and documents to be clearly seen [DDL⁺90]. It considers documents which have many terms in common as being closer to each other than documents which have few words in common. This helps to solve two major information retrieval problems, *polysemy* and *synonymy* [DDL⁺90]. *Polysemy* is where a word has multiple meanings. An example of polysemy is that a student may be looking for information about a “jaguar“ car, and so enters the term

“jaguar”. However this term can also mean jaguar “cat“, and the Mac OS X “Jaguar” operating system. In computational linguistics polysemy is called word sense disambiguation. Term matching, while a partial solution to this problem, returns irrelevant documents and this reduces precision. *Synonymy* is multiple words having the same meaning. For example the term “cat” can be also referred to as “feline“, “lion”, “kitten“, and so on. Synonymy can be partially solved by human generated thesaurus, however this causes the problem of inconsistent human judgments occurring in human generated indexes. Based on context, the human reader has little problem inferring meaning about these words, however it is difficult for machines to infer meaning about these words. Latent Semantic Analysis helps with these problems, in fact the term itself does not have to occur within the collection for Latent Semantic Analysis to find that the collection is relevant. This is a great feature with off-the-page rankings being so popular with search engines at present. With Latent Semantic Analysis it is possible to fingerprint each collection and measure a sample of each collection against a query.

A popular use of LSA is to analyse relationships between documents, however this research uses it to analyse relationships between collections. LSA is very suitable for collection selection because of its ability to match related terms and concepts.

Dimensionality reduction is another important part of Latent Semantic Analysis, allowing dimensions to be reduced while still keeping relative distances. This reduction pulls together related documents and terms, while removing the background noise.

Chapter 2

Distributed Information Retrieval

This work builds on previous research in distributed systems, server selection, and information retrieval. From distributed systems comes the necessity of dealing with different text formats and languages. From server selection comes the issue of having to be efficient in the selection of the collection servers in order to reduce costs. From information retrieval comes the issue of consistently delivering relevant information to the user.

Previous Collection selection methods discussed here are CORI, GIOSS, bGIOSS, vGIOSS, Lightweight Probes, and Probe Queries. With the exception of Probe Queries these methods require cooperation between the search broker and the collection.

2.1 CORI

CORI(*Collection Retrieval Inference Network*) is a Bayesian probabilistic inference network which is commonly used for document selection. Callan [CLC95] and later Lu [LCC96] apply CORI to the collection selection problem. Callan's solution is elegant, and is a popular collection selection method.

CORI assumes that the best collections are the ones that are estimated to contain the most documents related to the query. These collections are therefore ranked in order of number of documents relevant to the information need. Once ranked the top n collections are presented to the user in a list. Methods are available for calculating the number of documents about a particular term in a collection, and for calculating normalised document frequency, inverse collection frequency, and the importance of a collection for a particular term.

CORI uses document frequency and inverse collection frequency to rank each document. Because of this CORI uses minimal storage size for a large collection of documents. Also because an inference network is used for both collection selection and document selection the

one system can rank both collections and documents within the same framework. However there are distinctions to be made between collection selection and document selection using an inference network. Collection selection is actually looking for as many documents as possible about a topic. Care must be taken to not discard small sets of relevant documents.

2.2 GIOSS

Gravano's [GGMT99] solution to the collection selection problem is to use a server which contains all the relevant information of other collections. Users query this *Glossary of Servers Server* (or GIOSS for short) which then returns a ordered list of the best servers to contact to send the query to.

GIOSS is used to evaluate and rank collections. Collections are evaluated by the usefulness for each query. This usefulness is measured by the estimated number of documents in the collection that are similar to the query. The collections most likely to be selected contain many documents relevant to the current query. Collections are ranked based on information about each collection. However full data on each collection cannot be stored due to size restrictions. Instead each term and each collection is given a number based on the weight of the term and the number of documents in the collection that contain the term. These numbers are periodically updated by a collector program which gets this information from each collection. GIOSS works best with a large collection of heterogeneous data sources.

2.3 bGIOSS and vGIOSS

Two variations of GIOSS are available, *bGIOSS* [GGMT94] is a boolean version of GIOSS, and *vGIOSS* [GGMT99](also know as *gGIOSS*) which is a vector-space version of GIOSS. A decentralised version of GIOSS is also available, called hGIOSS.

Each server contributing to bGIOSS uses a Boolean evaluation system, which communicates document and server statistics back to the broker. Each server is ranked by an estimation on how many documents in the collection satisfy the query. However this method assumes that query distribution is independent across the collections.

vGIOSS uses a vector to represent each document, with each term in the space given a weighting based on frequency and normalisation. A query is also represented as a sparse vector, and is compared to the other document vectors.

Server ranking methods *Sum(l)* uses vector sum of the server's normalised document vectors, while *Max(l)* uses the document frequency statistics about each server to compare the

vectors. They estimate the summed scores of documents which score above l , and use the inner product to rank collections. The goodness value of each collection c_i with respect to query q is:

$$G_{i,q} = \sum_{j=1}^M W_{i,j} \quad (2.1)$$

In Equation 2.1 $W_{i,j}$ is the sum of document weights contributed by term q_j in collection c_i and M is the total number of terms for collection c_i . A threshold is used to ignore terms that are less than a certain weight.

The problem with this is that unless each collection uses the same ranking method, comparing the results becomes difficult. A solution can be found through message passing between collections, but in the real world this is difficult to implement unless each collection has the same interface and methods.

2.4 Lightweight Probes

Hawking and Thistlewaite [HT99] propose *Lightweight Probes*(LWP) for server selection. These probes use a minimal amount of communication between the search broker and the servers, and operate without global server information and descriptions. Lightweight Probes aim to reduce probe processing and reduce costs while still gathering enough information to give good results.

Probes for server S_i with terms t_1 and t_2 are as follows:

$$S_i = c_1 f'_1 + c_2 f'_2 + c_3 f'_{cooccur} + c_4 f'_{prox} \quad (2.2)$$

The primes indicate that a normalised frequency was used. The best results obtained under training for $c_1..c_4$ were $c_4 = 100, c_3 = 10, 0 \leq c_2 \leq 1$, and $0 \leq c_1 \leq 1$.

The servers are sorted in order of S_i , where

f_i the number of documents containing each individual term t_i ($i=1$ or 2),

$f_{cooccur}$ the number of documents in which a specified number of the terms occur near each other,

f_{prox} the number of documents containing a specified number of the terms within a specified proximity of each other, which often indicates a relationship between the terms.

Lightweight Probes are a two term subset of the user query that is sent to all collections for all queries. Statistics are retrieved using a special communication protocol between broker and server.

2.5 Incremental Probe Queries

Craswell [Cra01] and Callan et al [CCD99] use incremental *probe queries* for server selection. The search broker periodically broadcasts a query to all the servers. From the returned documents, the search broker can apply server ranking methods and select the best servers. Multi-term probe queries are taken from a query log, rather than on the fly, and are not run at query time.

Probe Queries are periodically sent to each server in batches. While some collection selection techniques require the implementation of cooperative interfaces between the search broker and the server, probe queries do not require special protocols between broker and server.

2.6 Collection selection methods comparison

Craswell [CBH00] compares the collection selection methods described above, CORI and GLOSS. The paper evaluates six different collection selection methods across different data and configurations. The selection methods used in the comparison are CORI, CORI plus E'i, CORI plus Ei, vGLOSS Max(0), CVV, centralised 100% index, centralised 50% index, and centralised 25% index.

In a comparison of CORI and GLOSS [CBH00] it was found that CORI was the best collection selection method, and that a selection of a small number of collections could outperform selecting all the servers and a central index. Probe queries are a good method for evaluating collections without having full knowledge of the collection contents, with 50,000 documents evaluated instead of 250,000 documents.

These conclusions are important to our research because they show that a high quality subset of collections will be as effective as a full set of collections, and that probes of a collection are an effective method of ranking an entire collection.

This is related to Chen's [CM00] Yarrow system in which a number of servers are used to search the world wide Web. By selecting only the most relevant collections for each query the system will be faster and use less bandwidth.

CORI and GLOSS are not deep web collection selection techniques because they require full knowledge of each collection, communication between search broker and collection, and keep metadata on each collection. So while we review them as related work, we will not compare our SVD system to them, instead we compare our solution against two other benchmarks in chapter 6.

Chapter 3

Sampling Technique for the Deep Web

Deep Web Information Retrieval(D.W.I.R.) is information retrieval in the areas of the web that cannot be accessed by following permanent hyperlinks. Deep Web Information Retrieval is significantly different to traditional Distributed Information Retrieval(DIR).

With traditional Distributed Information Retrieval, the search broker has full access to collection statistics and is able to communicate with the collection. The information is largely static, and indices are often built to make retrieval faster. Queries are often large and highly specific. After searches are performed, full access is given to the results. There are standard, widely used metrics to measure how well a search performed.

With Deep Web Information Retrieval, there are no term statistics available to the search broker. The information is highly dynamic. Probe queries are used to get information. There is no communication between collections and the search broker. Queries sizes are often limited. After searches are performed, limited access is given to the results. Precision and recall metrics are useless [MB00] because there are no statistics available about the size and distribution of the data in the collections.

Deep web collection selection requires the use of unusual information retrieval approaches to be used. We must create new information retrieval methods to work-around the lack of collection statistics and communication.

The objective of this thesis is to find a collection selection method that works best with sampling. The methods tested are probabilistic, baseline and a SVD-based sampling method.

Our new information retrieval approach is the use of *query samples*. Most deep web collections only allow access to a small portion of the collection through the query interface. To work around this problem, samples of collections will be taken, and we will try to show that a representative sample of each collection can produce results similar to the indexing of an entire collection. For each query q , a sample of the top n documents is taken, and these documents are

treated as an entire collection.

Once a set of useful collections has been discovered using a small sample from each collection, the search broker component can then retrieve a larger sample from the smaller set of collections at a minimised cost.

Past DIR work often required full cooperation between the search broker and the collections. The communications can include term statistics, message passing, and software that must run on both search broker and collection. In the deep web, this kind of cooperation rarely exists. The goal of this research is to produce results using noncooperative collections that approach the results of working with cooperative collections.

Interfacing with noncooperative deep web collections requires a clumsy and unsophisticated process. In order to search a collection, an interface for communicating with the collection must be manually configured, followed by configuring a *probe query* of the collection, then a screen scraper must be trained to extract the results and then parse them into a format that can be used by the search broker. In this paper we will concentrate on the *probe queries* problem and ignore the screen scraping of results.

It is important to make the distinction between a data retrieval system and a information retrieval system. A data retrieval system differs from an information retrieval system in the acceptable level of incorrect documents allowed in the set of retrieved items. A data retrieval system is defined on mathematical principles and thus no incorrect documents are allowed in the set of retrieved items [BYRN99]. An information retrieval system deals with ambiguous natural language information with possibly many meanings and is thus allowed a medium level of incorrect documents in the set of retrieved items.

To make retrieval faster, deep web collections are commonly searched in parallel, as some collections may have a higher response time than other collections due to network load, database access times, and other factors out of control of the searcher.

Chapter 4

Singular Value Decomposition Based Sampling Method

In this chapter we discuss *SVD (singular value decomposition)*, a powerful matrix factorization method. In this research, we use SVD for determining relevant collections. The advantage of SVD-based sampling method is that it can quickly show the relationship between a set of terms (a query) and collections. For the sake of completeness, we show the entire process of calculating SVD in this chapter, even though parts of the resulting decomposition are not needed for our system. Also, Matlab uses a slightly different method to the one presented here

4.1 Definitions

We introduce some definitions used in calculating the singular value decomposition of a matrix. We describe transpose operations, identity matrices, orthogonal matrices, determinants, eigenvectors and eigenvalues, and Gram-Schmidt orthonormalisation.

4.1.1 Transpose

The *transpose* of a matrix is obtained by exchanging a matrix's rows with its columns. This is achieved by replacing all elements a_{jk} with a_{kj} . We use the superscript letter T (T) to denote the transpose of a matrix.

4.1.2 Identity Matrix

An *identity matrix* is a simple diagonal square matrix of the form:

$$I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

We use the letter I to denote a identity matrix.

4.1.3 Orthogonal

A matrix A is *orthogonal* if:

$$AA^T = I \tag{4.1}$$

where I is an identity matrix.

Two vectors are **orthogonal** if their dot product is 0, i.e. $x^T y = 0$.

Vectors are called **pairwise orthogonal** if any two of them are orthogonal.

4.1.4 Determinants

Determinants can be used for solving systems of linear equations, and can be used to test for matrix singularity, where the determinant will be zero if the matrix is singular.

To calculate the determinant of a 2 by 2 matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

we use:

$$\det(A) = a_{11}a_{22} - a_{12}a_{21}$$

For a general n -by- n matrix, we calculate the determinant using Leibniz formula:

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^n A_{i,\sigma(i)} \tag{4.2}$$

Where S_n is the set of all permutations, $\text{sgn}(\sigma)$ denotes the signature of the permutation σ : +1 if σ is an even permutation and -1 if it is odd. The sum is computed over all permutations σ of the numbers 1,...,n.

We use \det to denote the determinant of a matrix.

4.1.5 Eigenvectors and Eigenvalues

Eigenvalues and *eigenvectors* are used in physics, mechanics, and many other fields. Eigenvalues and eigenvectors are also important for calculating Singular Value Decomposition.

Eigenvectors are the vectors that are preserved in direction under matrix multiplication. This is an important property that helps us to see the base components of the matrix.

Given a matrix A , its eigenvalue λ can be determined using the following equation.

$$\det(A - \lambda I) = 0 \quad (4.3)$$

where I is the identity matrix.

The number λ in Equation 4.3 is called the *eigenvalue* of A [Wat91]. To calculate the eigenvalues we need to solve the equation for λ . Eigenvalues are sometimes also known as characteristic roots, proper values, or latent roots. Each eigenvalue is paired with a corresponding eigenvector. The set of all eigenvectors corresponding to an eigenvalue of A , including 0, forms a vector space called the *eigenspace* of A .

Given a λ , its eigenvector X can be determined using the following equation.

$$(A - \lambda)X = 0 \quad (4.4)$$

To calculate the eigenvectors we need to solve the equation for λ . The decomposition of a matrix into eigenvalues and eigenvectors is known as eigen decomposition.

Now we use an example to show how the eigenvalues and eigenvectors of a matrix are calculated.

Let:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\det(A - \lambda I) = \det \begin{bmatrix} 1 - \lambda & 2 & 3 \\ 4 & 5 - \lambda & 6 \\ 7 & 8 & 9 - \lambda \end{bmatrix}$$

We can verify there are three eigenvalues. They are:

$\lambda_1 = 16.1168$, $\lambda_2 = -1.1168$, and $\lambda_3 = 0$.

The corresponding eigenvectors of A are:

$$v_1 = \begin{bmatrix} -0.232 \\ -0.5253 \\ -0.8187 \end{bmatrix} \quad v_2 = \begin{bmatrix} -0.7858 \\ -0.0868 \\ 0.6123 \end{bmatrix} \quad v_3 = \begin{bmatrix} 0.4082 \\ -0.8165 \\ 0.4082 \end{bmatrix}$$

$$\begin{array}{ccccccc}
 \boxed{A} & = & \boxed{U} & \cdot & \boxed{\Sigma} & \cdot & \boxed{V^T} \\
 m \times n & & m \times n & & n \times n & & n \times n
 \end{array}$$

Figure 4.1: Matrix Decomposition

4.1.6 Gram-Schmidt Orthonormalisation

A set of vectors $v_1, v_2, \dots, v_k \in R^n$ is called *orthonormal* if they are pairwise orthogonal, and each vector has a Euclidean norm 1:

$$v_i^T v_j = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

It is often convenient and simpler to use an orthonormal basis when doing some matrix calculations.

The Gram-Schmidt algorithm is used to find the orthogonal basis of a matrix. After the orthogonal basis have been found, the orthonormal basis are calculated by normalising each vector in the orthogonal basis.

To calculate u_i , we project v_i orthogonally onto the subspace generated by u_1, \dots, u_{i-1} .

We start with linearly independent vectors v_1, \dots, v_k and we find mutually orthogonal vectors u_1, \dots, u_k which occupy the same subspace as the vectors v_1, \dots, v_k .

In the following Gram-Schmidt algorithm, the dot product of v and u is represented as $(v.u)$:

$$\begin{aligned}
 u_1 &= v_1 \\
 u_2 &= v_2 - [(v_2.u_1)/(u_1.u_1)]u_1 \\
 u_3 &= v_3 - [(v_3.u_1)/(u_1.u_1)]u_1 - [(v_3.u_2)/(u_2.u_2)]u_2 \\
 &\dots \\
 u_k &= v_k - [(v_k.u_1)/(u_1.u_1)]u_1 - [(v_k.u_2)/(u_2.u_2)]u_2 - \dots - [(v_k.u_{k-1})/(u_{k-1}.u_{k-1})]u_{k-1}
 \end{aligned}$$

4.2 Singular Value Decomposition

Singular Value Decomposition is a matrix factorisation and dimension reduction method that has many uses: eg., information retrieval, time series analysis, and pattern matching.

4.2.1 Properties of Singular Value Decomposition

Figure 4.1 shows the decomposition of the $m \times n$ matrix A , where $m \geq n$.

A is a matrix that represents collections such that the rows are terms and columns are collections (i.e. vectors). The *Singular Value Decomposition* of A is said to be the factorisation:

$$A = U \Sigma V^T \quad (4.5)$$

where the diagonal of Σ is said to be the singular values of the original matrix, A :

$$\Sigma = \begin{bmatrix} w_0 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & w_{n-1} & 0 \\ 0 & 0 & 0 & w_n \end{bmatrix}$$

and

$$U^T U = V^T V = I \quad (4.6)$$

$$w_1, w_2, \dots, w_{n-1}, w_n \geq 0 \quad (4.7)$$

In Equation 4.5, matrices U and V are orthogonal. The columns of U are called left singular values (terms) and the rows of V^T are called right singular values (collection vectors). In this research, we never use the left singular values, which are also commonly represented as U .

The orthogonal matrices V and U are formed by the eigenvectors of the matrices $A^T A$ and $A A^T$ respectively. The dimensions of A are reduced to a smaller set of eigenvectors that are closest to the original matrix. This dimension reduction produces a clustering effect in the reduced matrix, removing noise from matrix and bringing together related concepts.

	C1	C2	C3	C4	Query
appear	1	4	4	4	0
brief	0	4	3	2	0
extensive	3	4	2	2	0
language	4	4	0	1	1
object	0	2	0	0	1
self	0	0	4	0	0
year	1	1	3	0	0

Table 4.1: The Term-Frequency Table

4.2.2 Singular Value Decomposition Example

In Table 4.1 we illustrate an example term-collection matrix. There are four collections, each collection has a column(vector) to show the term frequencies. The query column contains a 1 in the "language" row and a 1 in the "object" row, which indicates that the user wants to search for the terms "language object". Terms within the query can also be given weights as to how important they are to the query, allowing relevance feedback to be implemented.

The following is the procedure of computing SVD (see [ITL]). We give an example of how we generate the collection correlation matrix, first by calculating SVD on the Matrix A shown in Table 4.1, then by normalising and transforming the resulting matrix.

(I) Calculating the eigenvalues of the matrix $A^T A$ and arrange them in descending order, assigning each one to a lambda variable λ_1 to λ_5 gives $\lambda_1 = 134.4504$, $\lambda_2 = 29.394$, $\lambda_3 = 8.9824$, $\lambda_4 = 3.6183$, and $\lambda_5 = 0.5548$

(II) Calculating the number of nonzero eigenvalues of the matrix $A^T A$ results in $r = 5$.

(III) Finding the orthogonal eigenvectors of the matrix $A^T A$ corresponding to the obtained eigenvalues, and arranging them in the same order to form the column-vectors of the matrix $X \in R^{n \times n}$ give the following vectors v_1 to v_5

$$v_1 = \begin{bmatrix} 0.0356 \\ -0.2939 \\ 0.0477 \\ 0.3251 \\ 0.8969 \end{bmatrix} \quad v_2 = \begin{bmatrix} 0.4014 \\ -0.4873 \\ -0.0645 \\ 0.6553 \\ -0.4097 \end{bmatrix} \quad v_3 = \begin{bmatrix} 0.6785 \\ -0.2899 \\ 0.3768 \\ -0.5568 \\ 0.0598 \end{bmatrix}$$

$$v_4 = \begin{bmatrix} -0.5203 \\ -0.3417 \\ 0.7669 \\ 0.0469 \\ -0.1491 \end{bmatrix} \quad v_5 = \begin{bmatrix} -0.3264 \\ -0.6894 \\ -0.5133 \\ -0.3908 \\ -0.044 \end{bmatrix}$$

Hence the matrix V is the result $[v_1, v_2, v_3, v_4, v_5]$, resulting in:

$$V = \begin{bmatrix} 0.0356 & 0.4014 & 0.6785 & -0.5203 & -0.3264 \\ -0.2939 & -0.4873 & -0.2899 & -0.3417 & -0.6894 \\ 0.0477 & -0.0645 & 0.3768 & 0.7669 & -0.5133 \\ 0.3251 & 0.6553 & -0.5568 & 0.0469 & -0.3908 \\ 0.8969 & -0.4097 & 0.0598 & -0.1491 & -0.044 \end{bmatrix}$$

(IV) We then form the diagonal matrix Σ by taking square roots of the eigenvalues and sorting in descending order in a diagonal direction of $\delta_i = \sqrt{\lambda_i}$.

$$\Sigma = \begin{bmatrix} 11.5953 & 0 & 0 & 0 & 0 \\ 0 & 5.4216 & 0 & 0 & 0 \\ 0 & 0 & 2.9971 & 0 & 0 \\ 0 & 0 & 0 & 1.9022 & 0 \\ 0 & 0 & 0 & 0 & 0.7448 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(V) We then find the first 5 column-vectors of the matrix U using the following equation:

$$u_i = \delta_i^{-1} A v_i (i = 1 : r) \quad (4.8)$$

$$u_1 = \begin{bmatrix} -0.5779 \\ -0.438 \\ -0.4782 \\ -0.3879 \\ -0.1227 \\ -0.1771 \\ -0.2204 \end{bmatrix} \quad u_2 = \begin{bmatrix} -0.2523 \\ -0.1896 \\ 0.2398 \\ 0.6548 \\ 0.1535 \\ -0.5658 \\ -0.2654 \end{bmatrix} \quad u_3 = \begin{bmatrix} -0.4008 \\ -0.3813 \\ 0.1721 \\ 0.3528 \\ -0.1735 \\ 0.5029 \\ 0.5068 \end{bmatrix}$$

$$u_4 = \begin{bmatrix} 0.4286 \\ -0.4375 \\ 0.2294 \\ -0.0516 \\ -0.7278 \\ -0.1356 \\ -0.1469 \end{bmatrix} \quad u_5 = \begin{bmatrix} -0.4711 \\ 0.5136 \\ 0.4342 \\ -0.2532 \\ -0.4148 \\ -0.2561 \\ 0.1548 \end{bmatrix}$$

(VI) We then add to the matrix U the rest of $m-r$ vectors u_6 and u_7 using the Gram-Schmidt orthogonalisation algorithm described in Subsection 4.1.6 and using u_1, \dots, u_5 .

$$u_6 = \begin{bmatrix} -0.1321 \\ -0.1945 \\ 0.6018 \\ -0.2862 \\ 0.2862 \\ 0.3734 \\ -0.5284 \end{bmatrix} \quad u_7 = \begin{bmatrix} 0.1368 \\ -0.3657 \\ 0.2843 \\ -0.3842 \\ 0.3842 \\ -0.4149 \\ 0.5471 \end{bmatrix}$$

Hence $U = [u_1, u_2, u_3, u_4, u_5, u_6, u_7]$

$$U = \begin{bmatrix} -0.5779 & -0.2523 & -0.4008 & 0.4286 & -0.4711 & -0.1321 & 0.1368 \\ -0.438 & -0.1896 & -0.3813 & -0.4375 & 0.5136 & -0.1945 & -0.3657 \\ -0.4782 & 0.2398 & 0.1721 & 0.2294 & 0.4342 & 0.6018 & 0.2843 \\ -0.3879 & 0.6548 & 0.3528 & -0.0516 & -0.2532 & -0.2862 & -0.3842 \\ -0.1227 & 0.1535 & -0.1735 & -0.7278 & -0.4148 & 0.2862 & 0.3842 \\ -0.1771 & -0.5658 & 0.5029 & -0.1356 & -0.2561 & 0.3734 & -0.4149 \\ -0.2204 & -0.2654 & 0.5068 & -0.1469 & 0.1548 & -0.5284 & 0.5471 \end{bmatrix}$$

and the singular value decomposition of A is:

$$U = \begin{bmatrix} -0.5779 & -0.2523 & -0.4008 & 0.4286 & -0.4711 & -0.1321 & 0.1368 \\ -0.438 & -0.1896 & -0.3813 & -0.4375 & 0.5136 & -0.1945 & -0.3657 \\ -0.4782 & 0.2398 & 0.1721 & 0.2294 & 0.4342 & 0.6018 & 0.2843 \\ -0.3879 & 0.6548 & 0.3528 & -0.0516 & -0.2532 & -0.2862 & -0.3842 \\ -0.1227 & 0.1535 & -0.1735 & -0.7278 & -0.4148 & 0.2862 & 0.3842 \\ -0.1771 & -0.5658 & 0.5029 & -0.1356 & -0.2561 & 0.3734 & -0.4149 \\ -0.2204 & -0.2654 & 0.5068 & -0.1469 & 0.1548 & -0.5284 & 0.5471 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 11.5953 & 0 & 0 & 0 & 0 \\ 0 & 5.4216 & 0 & 0 & 0 \\ 0 & 0 & 2.9971 & 0 & 0 \\ 0 & 0 & 0 & 1.9022 & 0 \\ 0 & 0 & 0 & 0 & 0.7448 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.0356 & 0.4014 & 0.6785 & -0.5203 & -0.3264 \\ -0.2939 & -0.4873 & -0.2899 & -0.3417 & -0.6894 \\ 0.0477 & -0.0645 & 0.3768 & 0.7669 & -0.5133 \\ 0.3251 & 0.6553 & -0.5568 & 0.0469 & -0.3908 \\ 0.8969 & -0.4097 & 0.0598 & -0.1491 & -0.044 \end{bmatrix}$$

U is orthogonal because:

$$U^T U = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

V is orthogonal because:

$$V^T V = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Next we multiply Σ by the transpose of V :

$$\Sigma V^T = \begin{bmatrix} -6.5613 & -4.2029 & -3.6 & -4.1065 & -0.9723 \\ 0.9076 & -3.0511 & -1.9023 & 3.3503 & -0.0421 \\ 0.5269 & -1.4221 & 1.391 & -0.6459 & 0.1692 \\ 0.9011 & -0.0607 & -0.6355 & -0.6683 & -0.6431 \\ -0.1906 & 0.0171 & 0.2871 & 0.2204 & -0.7815 \end{bmatrix}$$

Now we normalise the columns so they all have length 1

$$E = \begin{bmatrix} -0.9781 & -0.7805 & -0.8259 & -0.7626 & -0.6875 \\ 0.1353 & -0.5666 & -0.4364 & 0.6221 & -0.0298 \\ 0.0785 & -0.2641 & 0.3191 & -0.1199 & 0.1196 \\ 0.1343 & -0.0113 & -0.1458 & -0.1241 & -0.4547 \\ -0.0284 & 0.0032 & 0.0659 & 0.0409 & -0.5526 \end{bmatrix}$$

Finally, calculating $F = E^T E$ gives us the collection correlation matrix:

	C1	C2	C3	C4	Query
C1	1	0.6644	0.7524	0.8028	0.6325
C2	0.6644	1	0.8094	0.2759	0.5252
C3	0.7524	0.8094	1	0.3408	0.6489
C4	0.8028	0.2759	0.3408	1	0.5252
Query	0.6325	0.5252	0.6489	0.5252	1

Now we are left with the final matrix. This is the collection correlation matrix and can be used to find the best collection. In the below table we illustrate the sorted lists of collections most similar to the query. Each collection is given a score between 0 and 1, and is ordered in descending order. From this table it can be seen that Collection 3 is the best collection to select for the query "language object".

Document	$S(Query, d)$
C3	0.6489
C1	0.6325
C2	0.5252
C4	0.5252

4.3 The Meaning Of Singular Value Decomposition

When singular value decomposition is applied to a matrix, the lesser patterns and the background noise are removed, allowing us to see the main patterns of the matrix. This is caused by selecting the highest singular values, and reducing the lengths of the vectors in space. The query column also becomes transformed as a vector in this reduced space, and lies close to similar collections. This can even have the effect of bringing related eigenvectors closer to other eigenvectors which do not use the same terms. The closeness of the patterns of occurrence of words with similar meanings is what helps to defeat polysemy and synonymy.

4.4 Calculation of Singular Value Decompositions in Matlab

The following is the code we use to calculate the singular value decomposition of a matrix A in Matlab. Matlab is a software development environment for doing matrix and vector computations. The matrix A should contain terms in the rows and collections in the columns. SVD is performed on the matrix (the U matrix is ignored in the following calculations). The matrix σ and matrix V are then transformed and the columns are normalised so they all have length 1, leaving a matrix containing a diagonal of the number 1 and a triangular of mirrored values which describe how the corresponding eigenvectors relate to each other.

```
// Apply SVD to matrix A
[U,Sigma,V] = svd (A);
// Multiply singular values by the transpose of matrix V'
B = Sigma * V';
//Normalise columns using Frobenius norm
normalised_columns=normc(B);
//Multiply normalised collection matrix transposed
//by collection matrix again to give matrix with
//all values between -1 and 1
final_matrix = normalised_columns' * normalised_columns;
```

Chapter 5

Algorithms and Experiments

In this chapter we present the details of our collection selection experiments. In section 5.2 we describe the testbed we use. In section 5.3 we present the algorithms we compared.

5.1 Introduction

Figure 5.1 illustrates the basic *collection selection* process. A query is taken from a user, and then the search broker broadcasts the query to all the collections in parallel. Each collection returns results that are most relevant to the query. The results are processed, and the results are returned to the user as a set of collections ordered by relevance to the query.

The queries are deliberately kept short because the system is intended for use on a deep web system where users most commonly use short queries, or as is the case in some collections, the interface restricts the number of query terms allowed. Better and more relevant results would be expected from longer queries.

The experiment uses a sample of n documents from every collection. A search engine is used to select the top n documents from each collection and then this sample is added to the term-collection matrix. The inputs of the Experiment are a query at run-time. A sample set of documents from each collection in INEX are extracted at run-time, after the query has been given.

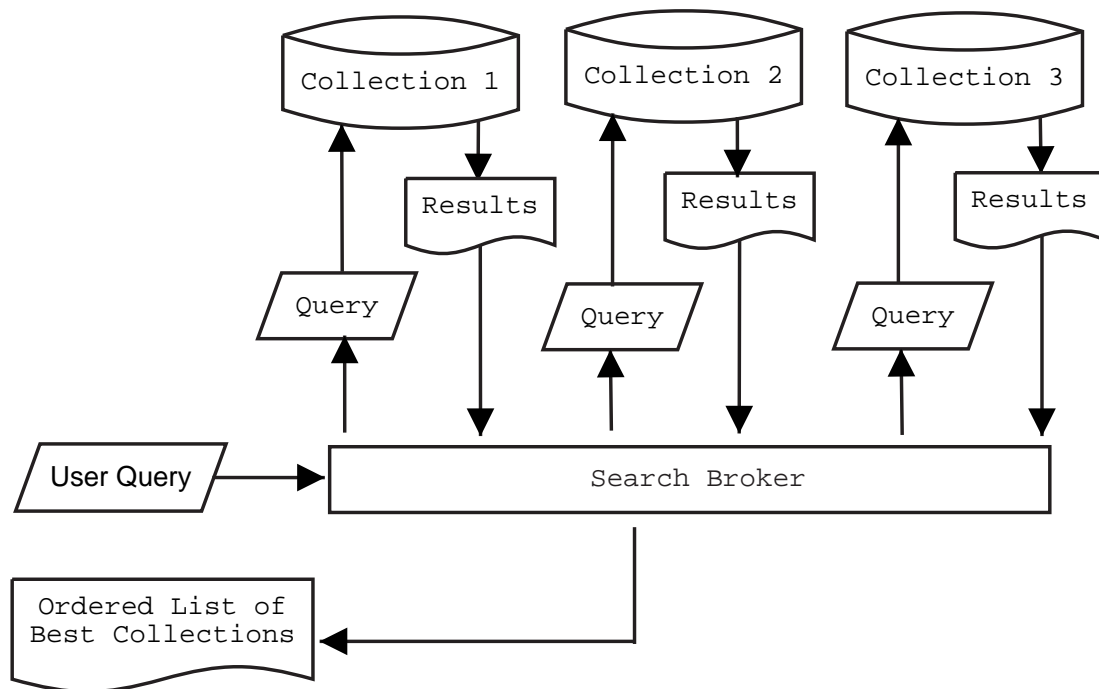


Figure 5.1: Collection Selection Using A Search Broker

5.2 Testbed

5.2.1 Document Set

Experiments were carried out on the *Initiative for the Evaluation of XML Retrieval* (INEX) collection. The collection consists of 12,232 papers from eighteen IEEE computing related journals between the years 1995 to 2001 and it is 497 megabytes in size.

The documents in INEX collection have already been pre-classified by editors into one of eighteen journal collections and allow well defined collection selection activities. Also there exists a set of queries for which the INEX documents which have been judged as relevant or irrelevant, namely topics 91 to 126 in the relevance assessments.

The XML markup was ignored in our experiments, we only use the **title** tags of the topics, ignoring the **description**, **keywords**, or **narrative** tags.

Code	Collection Name
an	IEEE Annals Of The History Of Computing
cg	IEEE Computer Graphics And Applications
co	Computer
cs	Computing In Science And Engineering
dt	IEEE Design And Test Of Computers;
ex	IEEE Intelligent Systems
ic	IEEE Internet Computing
it	IT Professional
mi	IEEE Micro
mu	IEEE Multimedia
pd	IEEE Concurrency
so	IEEE Software
tc	IEEE Transactions On Computers
td	IEEE Transactions On Parallel And Distributed Systems
tg	IEEE Transactions On Visualization And Computer Graphics
tk	IEEE Transactions On Knowledge And Data Engineering
tp	IEEE Transactions On Pattern Analysis And Machine Intelligence
ts	IEEE Transactions On Software Engineering

Table 5.1: The INEX Collections Names and Their Abbreviations

Figure 5.2 shows the size of each INEX collection in bar-chart format. *IEEE Transactions On Computers* has the largest size at 66.5M, and *IT Professional* has the smallest at 4.7M.

Figure 5.3 shows the number of files in each INEX collection in bar-chart format. *Computer* has the largest size at 1909 files, and *IEEE Transactions On Visualization And Computer Graphics* has the smallest at 233 files.

5.2.2 GP-XOR

Gardens Point XML Oriented Information Retrieval Search Engine is an XML search engine written at the Queensland University of Technology, and was used as the baseline search engine for the experiments using sampled data. Any other search engine could have been used for the purpose of deep web collection selection but for the purposes of this experiment the GP-XOR is used because we can compare our results with the pre-judged INEX Topics, something not possible using other web based search engines.

The methods used in this thesis are heavily dependant upon the quality of the GP-XOR

Collection	Files	Size
an	323	13.3
cg	687	19.2
co	1909	40.7
cs	578	14.7
dt	546	13.7
ex	709	20.4
ic	552	12.3
it	252	4.7
mi	611	15.9
mu	472	11.4
pd	369	10.7
so	943	21
tc	1050	66.5
td	773	59.1
tg	233	15.3
tk	593	48.4
tp	1064	63.2
ts	578	46.2

Table 5.2: INEX collections, the number of files in each collection, and their sizes in megabytes

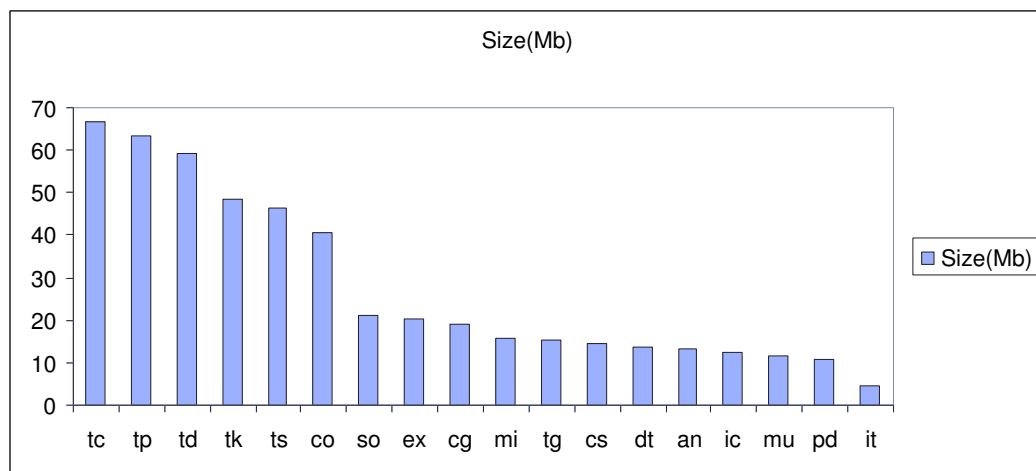


Figure 5.2: Size of INEX Collections in Megabytes

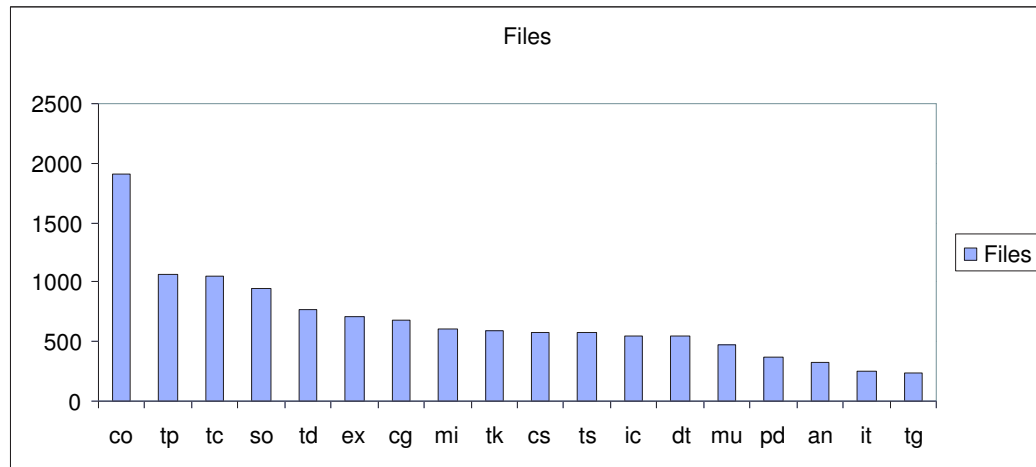


Figure 5.3: Number of files in each INEX Collection

search engine. If this engine returns a poor set of results, then it is not possible for any methods to return a good set of results because it can only work with what it has been given. This is a common problem with collection selection, and makes working with the deep web difficult.

5.2.3 Sampling

The methods used in deep web collection selection need to be tolerant of small sample sizes and able to work efficiently with collections where no direct access is available.

It must be noted that when sampling is used with a collection, the results can only ever be as good as the documents returned by the search broker, meaning that this technique is heavily dependant on the quality of the sample results returned by the broker.

Out of over 12232 documents, we take 10 documents per collection, or 1.4% of the combined collection size. However this number may change depending on the quality of the results returned by the collection indexer. In a better quality collection or with a better collection interface this sample size may be smaller.

5.3 The algorithms considered

The results of the INEX topics were compared to Singular Value Decomposition, Baseline Distributed Retrieval, and Probabilistic Self Recovery/Query Focused Retrieval. We now give a short description of Baseline Distributed Retrieval, Probabilistic Self Recovery/Query Focused Retrieval, and Singular Value Decomposition processes.

5.3.1 Baseline Distributed Retrieval

The Baseline Distributed Retrieval system is described in Xu and Croft [XC99] and represents a standard benchmark for distributed information retrieval systems. The system consists of a set of heterogenous collections, with each collection containing similar documents. A collection selection index is used to summarise each collection as a whole. A language model is used to select the most relevant collection to the query.

The language model is defined as:

- $|D|$ the size of the document D in words
- n the vocabulary size of the specified document
- $f(D, w_i)$ the number of occurrences of the word w_i in document D

The above definition returns the frequency with which word w_i is used in the text of D when observed with an un-limited amount of data, and is calculated for each term in each collection, and the collections are returned in descending order of score compared to the query.

$$p_i = \frac{f(D, w_i) + 0.01}{|D| + 0.01n}$$

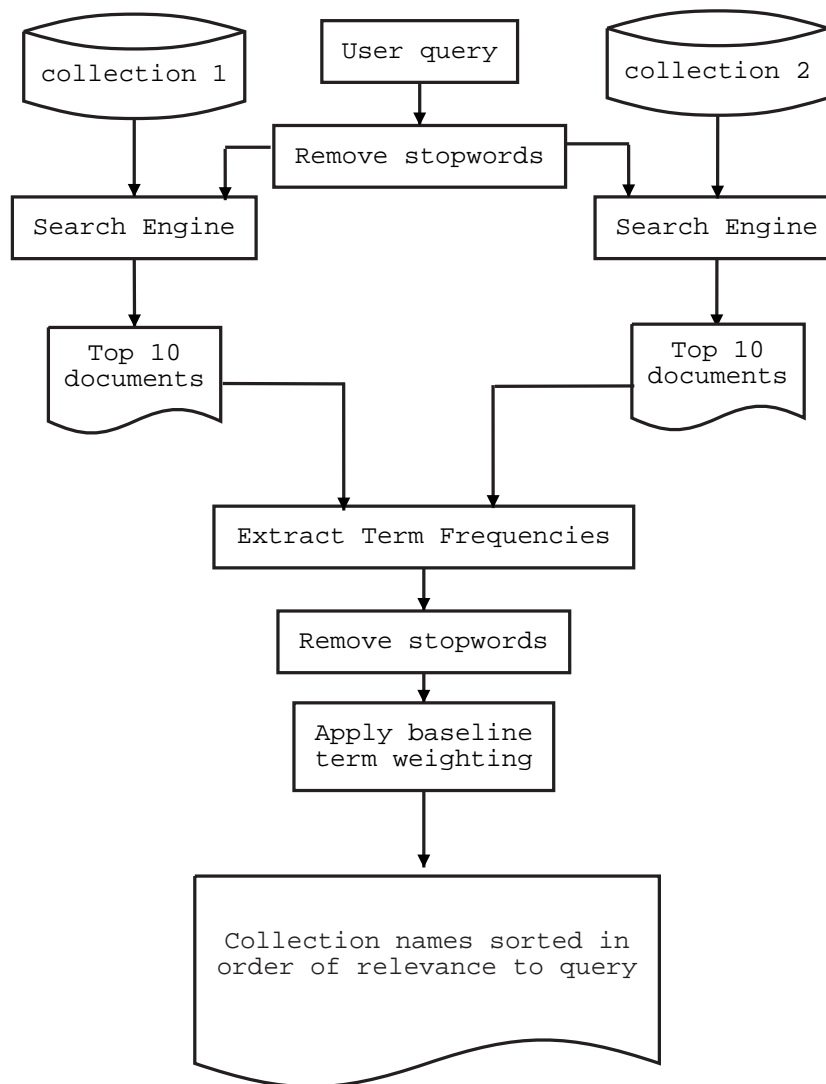


Figure 5.4: Baseline Collection Selection

Figure 5.4 illustrates the Baseline collection selection process. The following is the algorithm for Baseline collection selection:

- I Take a set of query terms (e.g. $q = [q_1..q_n]$);
- II Take a set of search engines $e = [e_1..e_{18}]$ for each collection (in this case we use the INEX Indexer for all 18 collections);
- III Execute the query terms on each search engine;
- IV Take the top set of 10 documents from each search engine (These will be the top 10 documents from INEX in each of the

```

    m = 18 collections);
V Remove stopwords;
VI Apply Baseline term weighting to calculate score for each
    collection;
VII Sort the collections in descending order according to
    their scores;

```

5.3.2 Query Focused Probabilistic Self Recovery

The Probabilistic Self Recovery/Query Focused system is described by Kwo in [Kwo90]. The probabilistic retrieval model ranks documents based on their probability of being relevant to a query using available information. An estimate is made of how relevant each document is.

Two formula are applied to each set of terms when finding the probability of a query matching a document. The first is Self Recovery Term Weighting. The second is Query-focused Ranking.

- d_{ik} times term k is in d_i
- q_{ak} times term k is in q_a
- r_{ak} probability of term k present, given q_a is relevant
- s_{ak} probability of term k present, given q_a is not relevant
- L_i the sum of d_{ik} for all term k in d_i (length of d_i)
- L_a the sum of q_{ak} for all term k in q_a (length of q_a)
- F_k the sum of d_{ik} for all d_i (times k term is in all the document collection)
- N_w the sum of L_i for all d_i (length of all the document collection)

Query-focused Ranking

Given a query, we can use the following equation to estimate a weighting score for a document d_i (or a collection for sampling technique):

$$w_{i/q} = \sum_k \frac{d_{ik}}{L_i} \cdot w_{ak}$$

Self Recovery Term Weighting

Where w_{ak} is calculated using "Self recovery term weight", which is described as follows:

$$r_{ak} = \frac{q_{ak}}{L_a}$$

$$s_{ak} = \frac{F_k}{N_w}$$

$$w_{ak} = \ln \frac{r_{ik}}{1 - r_{ik}} + \ln \frac{1 - s_{ik}}{s_{ik}}$$

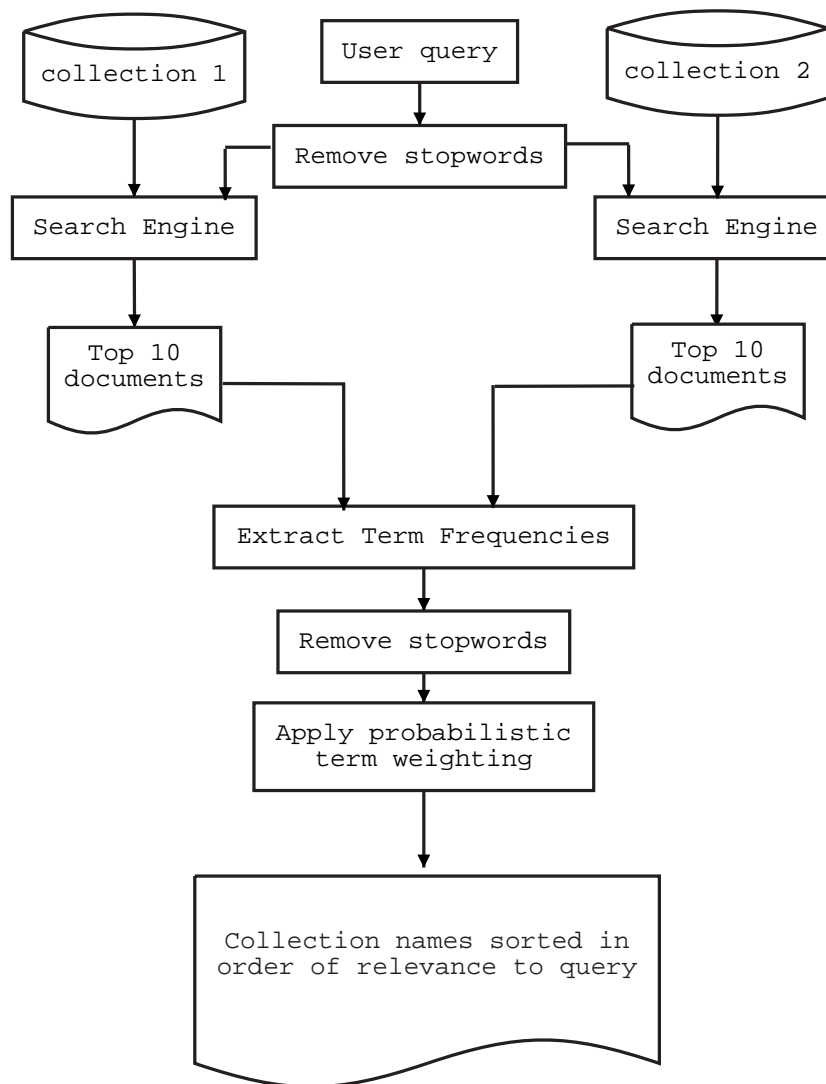


Figure 5.5: Probabilistic Collection Selection

Figure 5.5 illustrates the Probabilistic collection selection process.

The following is the algorithm for Probabilistic collection selection:

- I Take a set of query terms; e.g. $q = [q_1..q_n]$;
- II Take a set of search engines $e = [e_1..e_{18}]$ for each collection, in this case we use the INEX Indexer for all 18 collections;
- III Execute the query terms on each search engine;
- IV Take the top set of 10 documents from each search engine. These will be the top 10 documents from INEX in each of the $m = 18$ collections;

- V Remove stopwords;
- VI Apply Query-focused ranking and Self Recovery term weighting to the matrix;
- VII Sort the collections in descending order according to the document score values;

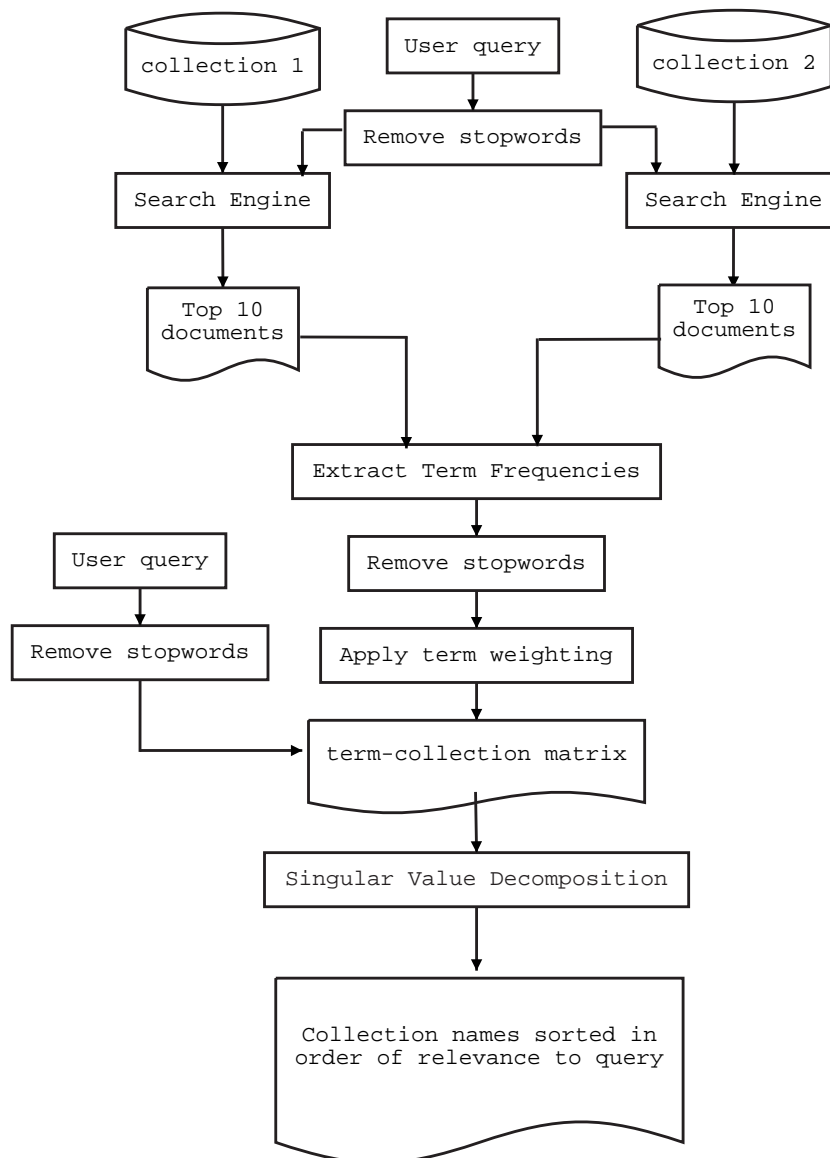


Figure 5.6: Sampled Singular Value Decomposition

5.3.3 Singular Value Decomposition

Figure 5.6 illustrates the Singular Value Decomposition method described in Chapter 4.

The following is the algorithm for Singular Value Decomposition collection selection:

I take a set of query terms; e.g. $q = [q_1..q_n]$;

II Take a set of search engines $e = [e_1..e_{18}]$ for each collection, in this case we use the INEX Indexer for all 18 collections;

-
- III Execute the query terms on each search engine;
 - IV Take the top set of 10 documents from each search engine. These will be the top 10 documents from INEX in each of the $m = 18$ collections;
 - V Create collection-term matrix A . For every document d_i , record word frequency for each word in the document and add it to the total collection score;
 - VI Remove stopwords;
 - VII Normalise the matrix;
 - VIII Add query terms to the A collection-term matrix as a query column q . Each term will be represented as 1 in the appropriate collection and term coordinate;
 - IX Run singular value decomposition on the A matrix to produce a collection correlation matrix. Find singular value decomposition $A = U \Sigma V^T$, where the orthogonal matrices V and U are formed by the eigenvectors of the matrices $A^T A$ and AA^T respectively;
 - X Sort the collections descending order according to the values in the query row in the matrix $A^T A$;

Chapter 6

Collection Selection Scoring

The goal of this research is to return a minimal set of high quality collections. It also does not matter what documents we select, it is what collections we select that matters.

A difficult problem with evaluating collection selection is the comparison of different collection selection algorithms against each other. For this reason we use the Inex Topics 91 to 126 which have been pre-judged by human editors.

A topic file looks like:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE inex_topic SYSTEM 'topic.dtd'>
<inex_topic topic_id='91' query_type='CO' ct_no='7'>
<title>
Internet traffic
</title>
<description>
What are the features of Internet traffic?
</description>
<narrative>
A relevant document/component describes the features of the internet
traffic, i.e. it contains information on traffic evolution, its
measurement and its possible congestion
</narrative>
<keywords>
internet, web, traffic, measurement, congestion
</keywords>
</inex_topic>
```

The above file is for query number 91. The query for this is in the <topic> tags; Internet traffic. The topic_id value is 91. All other information is ignored.

The assessment file for query number 91 has a portion that looks like:

```
<file file='ic/2000/w2014' > <path exhaustiveness='1'
specificity='1' path='/article[1]'/>

<path exhaustiveness='0' specificity='0' path='/article[1]/
fno[1]'/>

<path exhaustiveness='0' specificity='0' path='/article[1]/
doi[1]'/>

<path exhaustiveness='0' specificity='0' path='/article[1]/
fm[1]'/>

<path exhaustiveness='0' specificity='0' path='/article[1]/
bdy[1]'/>

<path exhaustiveness='1' specificity='1' path='/article[1]/
bm[1]'/>

<path exhaustiveness='0' specificity='0' path='/article[1]/
bm[1]/app[1]'/>

<path exhaustiveness='1' specificity='1' path='/article[1]/
bm[1]/app[2]'/>

<path exhaustiveness='0' specificity='0' path='/article[1]/
bm[1]/app[2]/apt[1]'/>

<path exhaustiveness='1' specificity='3' path='/article[1]/
bm[1]/app[2]/sec[1]'/>

<path exhaustiveness='0' specificity='0' path='/article[1]/
bm[1]/app[2]/sec[2]'/> </file>
```

The above is a small excerpt of the assessment file for query number 91. The <file> tag gives the journal name(ic) and the year. The path contains exhaustiveness, specificity and path values. The exhaustiveness and specificity values are a number between 0 and 3, with 0 meaning that the file is the least exhaustive or specific, and 3 meaning the file is very exhaustive or specific. *Exhaustiveness* describes the extent to which the document component discusses the topic of request. [GKP03]. *Specificity* describes the extent to which the document component focuses on the topic of request. [GKP03]. We add each of the exhaustiveness and specificity values together to get the score for each file. The path value is a pointer to the XML components of the file, for example /article[1]/fno[1] points to the file number of the article, /article[1]/bdy[1] points to the body section of the article, and /article[1]/bm[1]/ app[2] points to the bibliography section of the article.

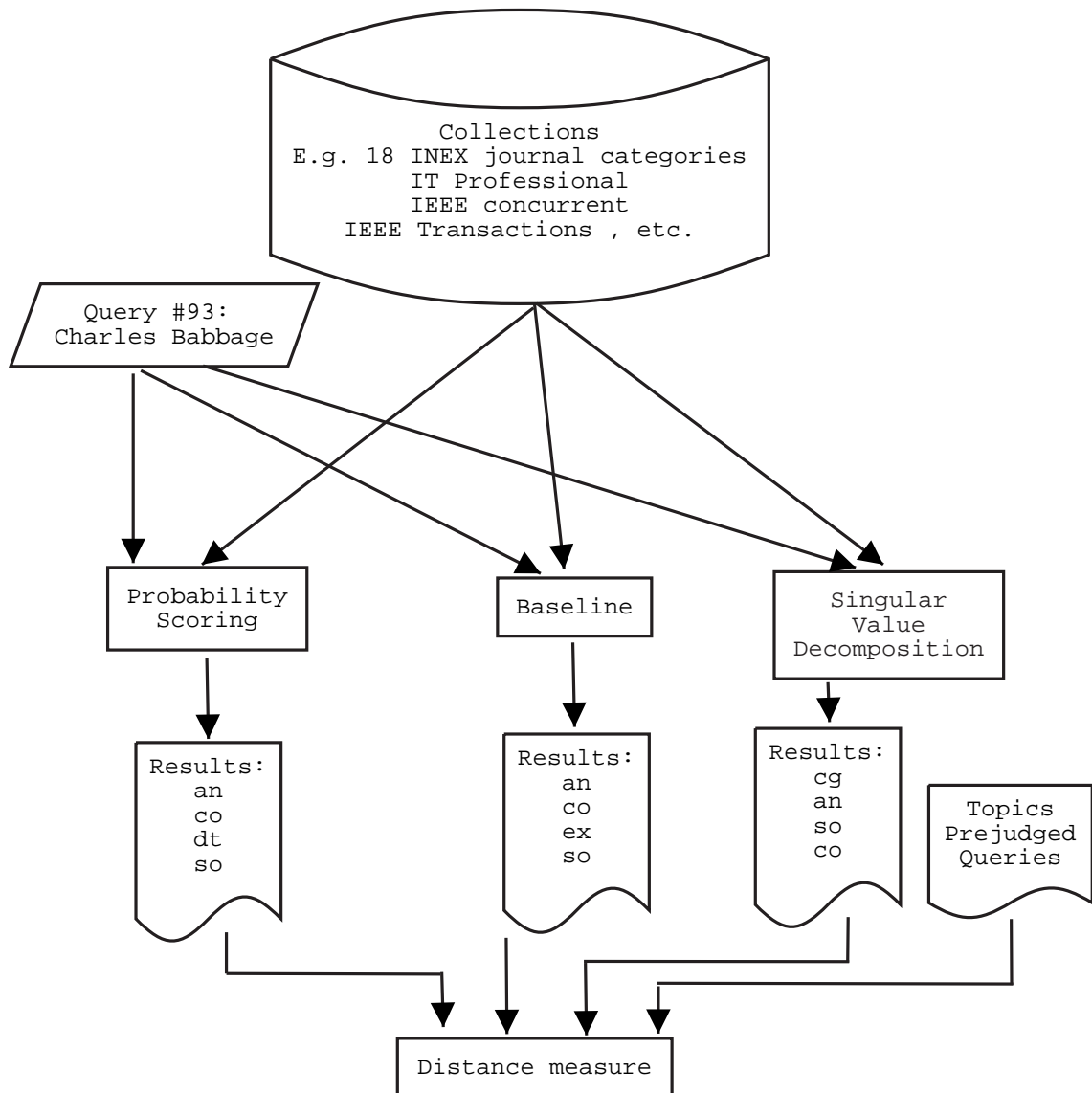


Figure 6.1: Collection Selection Scoring

Figure 6.1 illustrates the *collection selection scoring* process. This involves comparing different ranking techniques against each other. A query is taken, and each of the three collection selection techniques generate an ordered set of collections. Measurements are then taken comparing these results with the human-judged topics. The definition of the measurements can be found in the following chapter. Also, the results from these measurements are presented in the following chapter.

Chapter 7

Results Analysis

In this chapter we present the results of our comparison experiments along with a sample result table from each experiment. In section 8.1 we present the Cumulative Distance results. In section 8.2 we present the Average Precision results.

The experiments use the INEX collection. The INEX collection contains the set of IEEE Computer Society journal publications from the year 1995 to 2001. The collection consists of 12,232 papers from eighteen IEEE computing related journals and is 497 megabytes in size.

7.1 Cumulative Distance Measure

Our first result analysis technique is Cumulative Distance Measure. This technique allows us to see how different machine based results are from the human judged results. This allows us to see how well our algorithms are performing against the human judged benchmark.

Using the INEX assessments list for each query, we generated an ordered list of collections for each query based on cumulative exhaustiveness and specificity. We then generated an ordered set of collections for each query using baseline, probabilistic, and singular value decomposition. We then compared these ordered results against each other. The closer the results are to the human judged results, the better the algorithm. For a given algorithm, the distance for a collection is the absolute value of *correct position* – *actual position*.

In the Table 7.1 we compare exhaustiveness and specificity with probabilistic, baseline and SVD methods. Probabilistic and baseline methods both performed well in the first two instances. A zero indicates that there were no exhaustiveness or specificity rankings given to the journal.

exhaustiveness	specificity	probabilistic	baseline	SVD
ic	ic	ic	ic	co
co	co	co	co	pd
it	it	pd	pd	cs
mi	mi	cg	ex	it
tk	tk	dt	so	mu
mu	mu	ex	cs	ic
pd	cg	so	an	cg
cg	pd	cs	mu	tc
cs	cs	mu	mi	so
tc	so	an	dt	ts
td	tc	mi	it	mi
so	ts	it	cg	td
ts	td	tk	tk	ex
an	tg	tc	tc	tg
tg	an	ts	tg	tk
0	0	td	ts	tp
0	0	tg	tp	an
0	0	tp	td	dt

Table 7.1: Sample comparison of the Topic 91 results

In the Table 7.2 we show the distances between exhaustiveness and the probabilistic, baseline and SVD methods. Probabilistic and baseline methods both performed well in the first two instances. In some cases SVD performs better than both baseline and probabilistic methods. A blank space means that the journal was never in the exhaustiveness list.

probabilistic	baseline	SVD
0	0	1
0	0	5
4	4	6
4		1
	7	1
	3	5
5	7	1
1	2	2
3	5	3
4		3
7	8	7
9	4	1
8	8	
4	4	1
2	0	10
5	3	
2		3
	7	

Table 7.2: Distance measurements from exhaustiveness compared to probabilistic, baseline and SVD methods

Figure 7.1 and 7.2 show a comparison of SVD, baseline and probabilistic across the 18 levels of comparison. The numbers on the y axis represent the cumulative distance measures (no weighting applied). The numbers on the x axis represent the number of results compared. So number 2 means that the first 2 results across the 36 queries are compared to each other. The number 18 means that all 18 distances are compared to each other across all 36 queries. As you can see, the algorithms produce similar results.

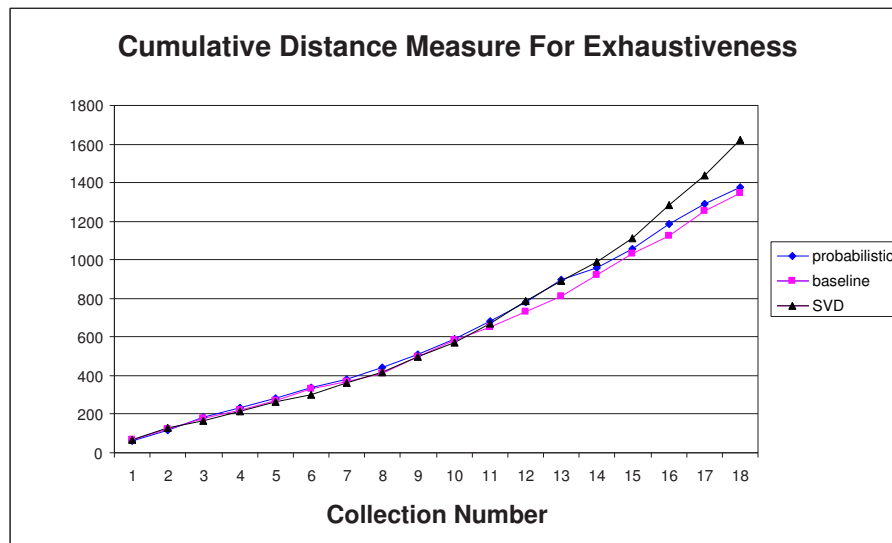


Figure 7.1: Cumulative Distance Measure For Exhaustiveness

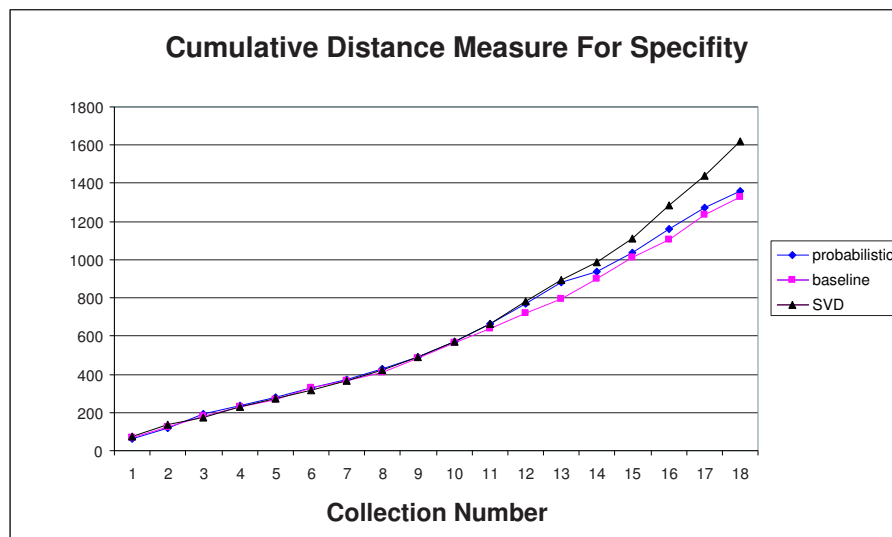


Figure 7.2: Cumulative Distance Measure For Specificity

	probabilistic	baseline	SVD
1	63	68	70
2	119	125	127
3	187	178	163
4	234	222	217
5	282	273	262
6	335	329	304
7	381	369	360
8	443	414	416
9	508	498	495
10	592	583	569
11	679	653	671
12	783	734	784
13	898	813	893
14	956	923	987
15	1056	1031	1110
16	1185	1123	1286
17	1289	1255	1438
18	1378	1347	1620

Table 7.3: Cumulative Distance Measure for Exhaustiveness

For example, if you were to use GP-XOR results and apply baseline, probabilistic and SVD to them, then take the top 2 results as being the best collections, you would have a distance of prob=119, baseline=125, SVD=127.

7.2 Average Precision Measurement

Precision is a standard IR measure of performance. It is defined as the number of relevant documents retrieved divided by the total number of documents retrieved, and is a measure of number of correct hits. See Equation (2.3).

Collection selection precision differs, because only the top collections retrieved are important.

Table 7.4 shows the precision measures for top 3 results of topics 91 to 126 when compared to the Exhaustiveness values.

Table 7.5 shows the average precision measures for top n results of topics 91 to 126 when compared to the Exhaustiveness values.

	probabilistic	baseline	svd
91	0.6667	0.6667	0.3333
92	0.6667	0.6667	0.6667
93	0.6667	0.6667	0.6667
94	0.3333	1.0000	0.3333
95	0.6667	0.6667	0.0000
96	0.6667	0.6667	0.0000
97	0.6667	0.6667	0.6667
98	0.0000	0.0000	0.0000
99	0.3333	0.3333	0.0000
100	0.0000	0.3333	0.0000
101	0.3333	0.0000	0.0000
102	0.0000	0.3333	0.3333
103	0.3333	0.3333	0.0000
104	0.6667	0.6667	0.0000
105	0.0000	0.0000	0.3333
106	0.3333	0.3333	0.0000
107	0.3333	0.3333	0.0000
108	0.6667	0.6667	0.3333
109	0.3333	0.3333	0.0000
110	0.0000	0.0000	0.3333
111	0.0000	0.0000	0.6667
112	0.6667	1.0000	0.6667
113	0.3333	0.3333	0.3333
114	0.0000	0.0000	0.0000
115	0.3333	0.0000	0.0000
116	0.3333	0.3333	0.6667
117	0.3333	0.3333	0.3333
118	0.3333	0.3333	0.0000
119	0.3333	0.3333	0.0000
120	0.0000	0.0000	0.0000
121	0.3333	0.3333	0.0000
122	0.3333	0.3333	0.0000
123	0.3333	0.3333	0.0000
124	0.0000	0.0000	0.0000
125	0.6667	1.0000	0.3333
126	0.0000	0.0000	0.6667

Table 7.4: Precision measurements for Top 3 Topic 91 Results compared to Exhaustiveness

Top n results	probabilistic	baseline	svd
1	0.2500	0.2222	0.0556
2	0.2917	0.2917	0.1528
3	0.3333	0.3704	0.2130
4	0.4028	0.4375	0.2986
5	0.4333	0.4611	0.3389
6	0.4676	0.4769	0.3657
7	0.4524	0.4683	0.3968
8	0.4722	0.4826	0.4097
9	0.5062	0.5154	0.4414
10	0.5139	0.5222	0.4472
11	0.5152	0.5227	0.4672
12	0.5162	0.5255	0.4838
13	0.5171	0.5128	0.4829
14	0.5040	0.5139	0.4742
15	0.4944	0.5037	0.4722
16	0.4878	0.4913	0.4722
17	0.4804	0.4804	0.4690
18	0.4676	0.4676	0.4676

Table 7.5: Exhaustiveness Precision Average Across 18 collections

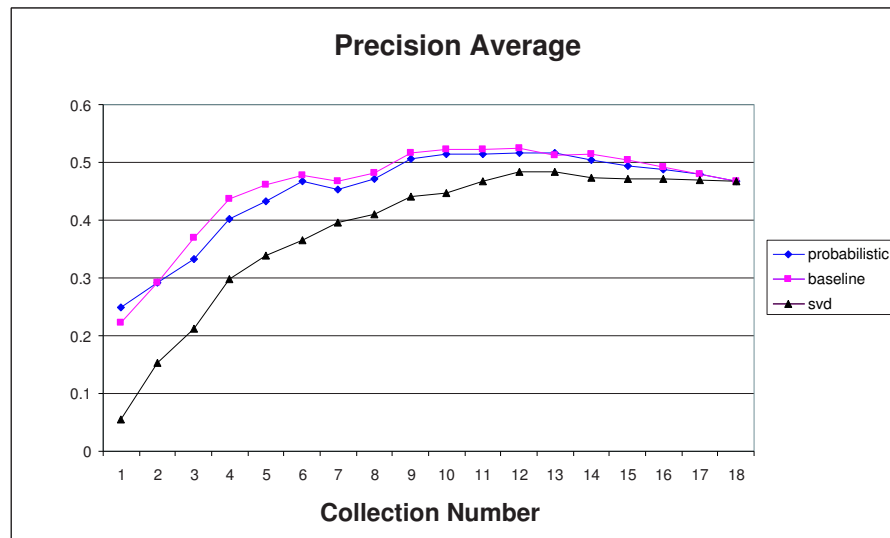


Figure 7.3: Average Precision

Figure 7.3 shows the average precision measures for top n results of topics 91 to 126 when compared to the Exhaustiveness values. The precision ranges between 0(worst) and 1(best).

7.3 Conclusion

A problem we encountered was that many of the results were the same for the top two positions, and varied afterwards. This is an important problem with collection selection because it is the top few items that matter the most, and presenting a long list of recommended collections to the user is probably not going to be useful as the top two or three are all that matter.

Overall the baseline method performed best, however using precision over the first one and two queries probabilistic performed better or as well as the baseline method. This indicates that probabilistic is the better method when returning a small number of results.

Although much faster to calculate, singular value decomposition performed poorly in the average precision metric, and produced average performance in the cumulative distance metric.

Another interesting feature is that certain collection selection methods worked well for some queries and poorly for other queries. SVD tended to work well where probabilistic and baseline worked badly.

Chapter 8

Discussion and Conclusion

The deep web contains a vast number of electronic collections which cannot be indexed by traditional search engines. Due to the number of collections available, methods need to be found which will automatically identify the best collections to use for a particular query. Previous collection selection methods such as GLOSS and CORI required communication between search broker and collections, or software running concurrently on broker and collection. These methods are effective when dealing with cooperative collections, however few deep web collections can be called cooperative. In this thesis we have reviewed three different collection selection algorithms that work with un-cooperative collections. We have also developed a sampling technique which can be used to rank the collections based on relevance to a query. It also transforms some traditional information retrieval based techniques to this area. In addition, the thesis tests these techniques using INEX collection for total 18 collections (including 12232 XML documents) and total 36 queries. The experiment shows that the performance of sample-based technique is satisfactory in average.

This work uses probe queries similar to those in [Cra01], except our probe queries are performed at query time, and can work with any server that can return an ordered set of documents for a query.

The success of the methods used in this thesis is heavily dependant upon the quality of the search broker used to obtain results from the collection. If this broker returns a poor set of results, then it is not possible for these methods to return a good set of results because it can only work with what it has been given.

Another problem we encountered was that of building a flexible search broker between the user and the collections. Because many of the collections on the deep web only expose a search interface that returns HTML results, significant time had to be spent adding each collection to the searchable set of collections. A standard query interface between the user and the collection had to be written, and a screen scraper had to be manually configured to work with each collection, parse the results, and return them to the user. Automating this process will make it possible to add many more collections to the set of searchable collections.

Work still needs to be done to find the optimal term weighting scheme, and on the optimal sample size taken from each collection.

Bibliography

- [BDJ99] Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. Matrices, vector spaces, and information retrieval. volume 41, pages 335–362. Society for Industrial and Applied Mathematics, 1999. 12
- [Ber01] Michael K. Bergman. The deep web: Surfacing hidden value. *The Journal of Electronic Publishing*, 7(1), 2001. 1, 7
- [BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998. 7
- [BYRN99] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. Modern information retrieval. ACM Press / Addison-Wesley Boston, MA, 1999. 19
- [CBH00] Nick Craswell, Peter Bailey, and David Hawking. Server selection on the world wide web. In *Proceedings of the fifth ACM conference on Digital libraries, San Antonio, Texas, United States*, pages 37–46. ACM Press, 2000. 2, 17
- [CC00] French-J.C. Powell A.L. Callan, J. and M. Connell. The effects of query-based sampling on automatic database selection algorithms. In *Technical Report CMU-LTI-00-162*, Carnegie Mellon University, 2000. Language Technologies Institute, School of Computer Science. 2
- [CCD99] Jamie Callan, Margaret Connell, and Aiqun Du. Automatic discovery of language models for text databases. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 479–490. ACM Press, 1999. 17
- [CLC95] J. P. Callan, Z. Lu, and W. Bruce Croft. Searching Distributed Collections with Inference Networks . In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, Seattle, Washington, 1995. ACM Press. 1, 2, 4, 14

- [CM00] Z. Chen and X. Meng. Yarrow: A real-time client-side meta-search learner. In *Proceedings of the 2000 AAAI Workshop on Artificial Intelligence for Web Search (AAAI'00)*, Austin, Texas, pages 12–17, 2000. 17
- [Cra01] Nicholas Eric Craswell. *Methods for Distributed Information Retrieval*. PhD thesis, The Australian National University, 2001. 17, 57
- [DDL⁺90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. 12
- [DTZ00] Daryl J. D'Souza, James A. Thom, and Justin Zobel. A comparison of techniques for selecting text collections. In *Proceedings of the 11th Australasian Database Conference(ADC'2000)*, pages 28–32, Canberra, Australia, 2000. 2
- [FPC⁺99] 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. In *Research and Development in Information Retrieval*, pages 238–245, 1999. 2, 7
- [GGM95] Luis Gravano and Héctor García-Molina. Generalizing GLOSS to vector-space databases and broker hierarchies. In *International Conference on Very Large Databases, VLDB*, pages 78–89, 1995. 2, 11
- [GGMT94] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 126–137. ACM Press, 1994. 15
- [GGMT99] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. GLOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2):229–264, 1999. 1, 2, 4, 15
- [GKP03] Mounia Lalmas Gabriella Kazai and Benjamin Piwowarski. Inex 03 relevance assessment guide, September 2003. 46
- [HC02] Jiawei Han and Kevin Chang. Data mining for web intelligence. *Computer*, 35(11):64–70, 2002. 1, 4, 7
- [HT99] David Hawking and Paul Thistlewaite. Methods for information server selection. *ACM Trans. Inf. Syst.*, 17(1):40–76, 1999. 2, 16

- [ITL] Seppo Pohjolainen Ivar Tammeraid, Juri Majak and Tero Luodeslampi. Linear algebra. <http://www.cs.ut.ee/toomas.l/linalg/>. 25
- [KL03] John King and Yuefeng Li. Web based collection selection using singular value decomposition. In *IEEE/WIC International Conference on Web Intelligence (WI'03)*, pages 104–110. IEEE, 2003. 4
- [Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999. 7
- [Kwo90] K. L. Kwok. Experiments with a component theory of probabilistic information retrieval based on single terms as document components. *ACM Trans. Inf. Syst.*, 8(4):363–386, 1990. 38
- [LCC96] Z. Lu, J.P. Callan, and W.B. Croft. Applying inference networks to multiple collection searching. Technical Report TR96–42, University of Massachusetts at Amherst. Department of Computer Science, 1996. 2, 14
- [Li01] Yuefeng Li. Information fusion for intelligent agent based information gathering. In Ning Zhong, Yi Yu Yao, Jiming Liu, and Setsuo Ohsuga, editors, *Web Intelligence: Research and Development, First Asia-Pacific Conference, WI 2001, Maebashi City, Japan, October 23-26, 2001, Proceedings*, volume 2198 of *Lecture Notes in Computer Science*, pages 433–437. Springer, 2001. 2
- [MB00] Filippo Menczer and Richard K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the web. *Machine Learning*, 39(2/3):203–242, 2000. 18
- [MLY⁺99] Weiyi Meng, King-Lup Liu, Clement T. Yu, Wensheng Wu, and Naphtali Rishe. Estimating the usefulness of search engines. *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Australia*, pages 146–153, 1999. 2
- [NK98] Kwong Bor Ng and Paul P. Kantor. An investigation of the preconditions for effective data fusion in information retrieval: A pilot study, 1998. 9
- [TL01] Theodora Tsirikika and Mounia Lalmas. Merging techniques for performing data fusion on the web. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 127–134. ACM Press, 2001. 2

-
- [VGJL95] E. Voorhees, N. Gupta, and B. Johnson-Laird. The collection fusion problem, 1995. 2
- [Wat91] David S. Watkins. *Fundamentals of Matrix Computations*. John Wiley and Sons, Inc, 1991. 22
- [XC99] Jinxi Xu and W. Bruce Croft. Cluster-based language models for distributed retrieval. In *Research and Development in Information Retrieval*, pages 254–261, 1999. 36