Query Based Stemming

by

Elizabeth Tudhope

University of Waterloo

Department of Computer Science

Technical Report CS-96-31

Waterloo, Ontario, Canada, 1996

© Elizabeth Tudhope 1996

Abstract

In information retrieval the relevancy of a document to a particular query is based on a comparison of the terms appearing in the query with the terms appearing in the document. Morphological variants of words (i.e. *locate, locates, located, locating*) often carry the same or similar meaning. Such terms should be considered equivalent for information retrieval purposes. Stemming is a simple application of natural language processing that is commonly applied at index time to reduce morphological variants to a common root form.

This thesis first examines several approaches to stemming. Some of the problems associated with the use of stemming as a query expansion technique are then discussed. The construction of equivalence classes of words for stemming at query time is presented as a possible alternative method to address some of these problems.

Acknowledgments

I would like to thank my supervisor, Frank Tompa for his sound advice, suggestions, and attention to detail.

I would also like to thank my readers Forbes Burkowski and Rick Kazman for the time they took to carefully read my thesis and offer encouragement and suggestions.

A very special thanks goes to Charlie Clarke for always having an open door. His unending patience, encouragement, and advice is greatly appreciated.

I am grateful for the hardware, software, technical and academic support provided by University of Waterloo MultiText project.

I gratefully acknowledge funding provided by Natural Sciences and Engineering Research Council, Information Technology Research Center and the University of Waterloo Department of Computer Science.

Contents

List of Figures

Chapter 1

1. Motivation

1.1. Problem Setting

Since the 1940's the problem of information storage and retrieval has attracted increasing attention [WMB94]. The growth of scientific literature, the computer revolution, and more recently the advent of the World Wide Web has produced an information explosion. Presently, it is estimated that the amount of information in the world doubles about every 20 months [WMB94]. The accuracy of these statistics may be debatable, but they underscore the rapid advance towards an information driven society.

There are two main challenges when dealing with huge volumes of data. The first is *storage*, which deals with filing the data efficiently. The second is *retrieval*, which deals with providing efficient and effective access to the data [WMB94]. In this thesis we focus on one method for improving effective access to electronic document collections.

It was not long ago that information retrieval meant going to a filing cabinet and pulling the appropriate file, or perhaps consulting the local library's card catalogue to find an

appropriate item. This traditional method of storing documents on paper (or in books) is very expensive in terms of the physical space and the time required to locate and access information.

It is now commonplace for home computers to come equipped with drives that can store a gigabyte or more of data. A gigabyte of storage is approximately equivalent to a thousand books. The fact that this quantity of information can be stored on a device that is smaller than the average book makes *electronic* storage extremely attractive.

The current interest in information retrieval has grown from the need for accurate and timely access to a growing information base. Information retrieval (IR) can be defined loosely as the study of methods and structures to represent and access information [WMB94]. One structure that has historically played an important role in the access to information is the index.

Indexes in the back of books make it possible to find sections of interest without resorting to a linear scan of the books pages. Often it is helpful to have an index to a collection of books (i.e. Works of Shakespeare, the Bible). Using manual techniques, the task of compiling an index or concordance can take a lifetime: in 1875, Mary Cowden Clarke wrote in the preface to her Shakespeare concordance, “Sixteen years of hard work, but delightful work, sufficed to complete the manuscript” [Cla1894]. In contrast, the same index could be compiled electronically in seconds. When electronically compiling an exact-word index, all words in the document can be viewed as keywords and no human intervention is required.

Looking for information in a book without an index can be a frustrating and time consuming activity. For the average book it is possible to scan each page and with the help of chapter and section headings, zoom in on the desired information. Electronic text collections are often gigabytes in size. Manually scanning collections of this magnitude is prohibitive, even with mechanical assistance [ELK91]. Just as an index is an important tool

for accessing print material, accurate and comprehensive indexing is critical to the success of a computerized information retrieval system.

There are a variety of index methods that have been used in modern IR systems. Three of the more popular index structures are inverted files, signature files and bitmaps. The method that most closely resembles a conventional concordance is the inverted file structure.

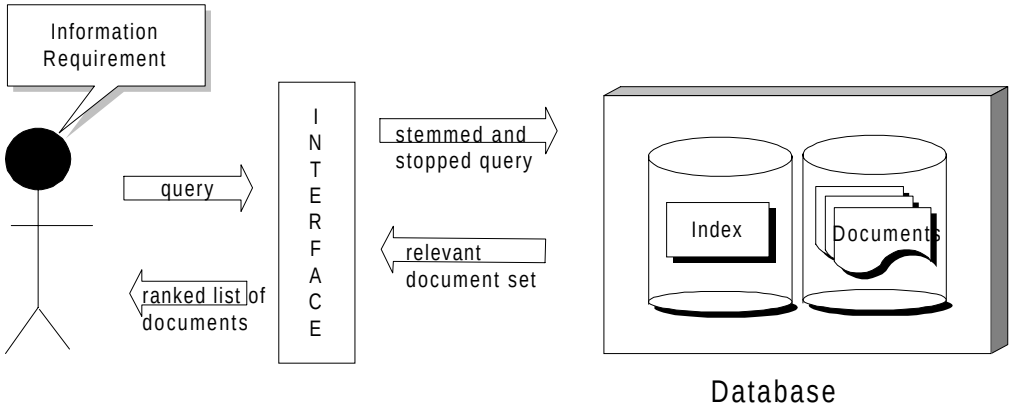


Figure 1: Typical IR System.

Figure 1 illustrates a typical IR system. The data in our typical IR system is assumed to be a collection of separate documents. Each document is represented by a set of terms. These terms may be the actual text of the document, an abstract, or a set of keywords. Queries entered in the system will return a list of documents that the system judges as relevant to the information need expressed by the query.

Various systems have diverse definitions of document. For example, an index to *The Complete Works of Shakespeare* could reference each play/poem or each Act or each Scene or even each verse as a separate document. The choice of document unit should be one that logically groups pieces of text in a way that suits the intended application of the system.

An inverted file index contains an entry for each term. Each term's entry stores a list of pointers to the occurrences of the term in the document collection. The pointer is often a unique identifier for the document (document id) in which the term occurs. These pointers are sometimes referred to as postings, and the term entry as a postings list.

For example, Figure 2 shows a selection of text from *Green Eggs and Ham* by Dr. Seuss, where each line is considered to be a document. Figure 3 shows the inverted file that would be generated for the document collection. Figure 4 is an inverted file generated for the same document collection, but with indexing at the word rather than document level.

Document	***Text***
1	I could not, would not, on a boat.
2	I will not, will not, with a goat.
3	I will not eat them in the rain.
4	I will not eat them on a train.
5	Not in the dark! Not in a tree!
6	Not in a car! You let me be!

Figure 2: Sample document collection.

The granularity of an index is the degree of accuracy that is used when indicating the location of a term. A course-grained index would identify a document block (where such a block contains several documents). A moderate-grained index would store locations in terms of documents. A fine-grained index would return a paragraph, sentence of even the exact location of each word. The coarser the grain of the index, the less storage is required. This is because even though a term may appear multiple times in a region, only one posting is required. However, one of the disadvantages of a course grain index is false matches for queries with multiple terms. A match will occur whenever all the desired query terms appear in the same block, even if the various terms may actually appear in different and unrelated documents within the block. Indexing at the word level requires more storage, but has the

advantage of being able to handle queries that involve proximity of terms. For example if the index was built at the word level, the postings in the inverted file could easily be used to determine whether the phrase, *let me be* occurs in the document collection.

In addition to granularity, a decision has to be made about what defines an indexable term. The simplistic approach is to index *all* alphabetic and alphanumeric strings *unaltered*. This approach has several serious drawbacks. Indexing terms exactly as they appear in the document means that their case will be preserved. This means that *Not* and *not* will have separate entries in the inverted file (see entries 13 and 14 in Figure 4). Additionally, a search will fail if there is not a case match between the query and the term as it appears in the document. The solution to this problem is to index all terms as lower-case only (case folding). The result is that *Not*, *not*, *nOt*, *noT* will all be indexed as *not*. Similarly, all query terms will be converted to lowercase before consulting the inverted file. There are instances such as acronyms and proper names when case is important. For example *time* vs. *Time* (the magazine), *zeppelin* vs. *Zeppelin*, and *of* vs. *OF* (designation for old French in the *Oxford English Dictionary*). Situations where case is important can be handled by performing case-sensitive post-processing of the search results.

To save space, numeric data is often not included in an index. However, in some domains it is important for numbers to be indexed. For example in a movie database users will expect the capability to search for a list of a particular movie type (i.e. Action/Adventure) released in a particular year. In such cases numeric information should be indexed.

Entry Number	***Term***	***Document (Postings List)***
1	a	1,2,4,5,6
2	be	6
3	boat	1
4	car	6
5	could	1
6	dark	5
7	eat	3 4
8	goat	2
9	I	1,2,3,4
10	in	3,5,6
11	let	6
12	me	6
13	not	1,2,3,4
14	Not	5,6
15	on	1,4
16	rain	3
17	the	3,5
18	them	3,4
19	train	4
20	tree	5
21	will	2,3,4
22	with	2
23	would	1
24	You	6

Figure 3: Inverted file entries for text of *Figure 2* indexed at the document level.

Entry Number	***Term***	***Document (Postings List)***
1	a	(1,7) (2,7) (4,7) (5,7) (6,3)
2	be	(6,8)
3	boat	(1,8)
4	car	(6,4)
5	could	(1,2)
6	dark	(5,4)
7	eat	(3,4) (4,4)
8	goat	(2,8)
9	I	(1,1) (2,1) (3,1) (4,1)
10	in	(3,6) (5,2) (5,6) (6,2)
11	let	(6,6)
12	me	(6,7)
13	not	(1,3) (1,5) (2,3) (2,5) (3,3) (4,3)
14	Not	(5,1) (5,5) (6,1)
15	on	(1,6) (4,6)
16	rain	(3,8)
17	the	(3,7) (5,3)
18	them	(3,5) (4,5)
19	train	(4,8)
20	tree	(5,8)
21	will	(2,2) (2,4) (3,2) (4,2)
22	with	(2,6)
23	would	(1,4)
24	You	(6,5)

Figure 4: Inverted file entries for text of *Figure 2* indexed at the word level.

A stop list is a mechanism that is often used to decrease the size of an index. Stop words are frequently occurring words that carry very little information value. Frequently occurring terms such as *the, of, and, to* etc. have low discriminating value, when used individually. Such terms are likely to occur in almost every document in the database. Typically the 10 most frequently occurring words in English account for 20 to 30 percent of the tokens in a document [Fox92]. Ignoring these terms speeds up document processing, reduces the index size and normally does not adversely effect retrieval. An exception would be a search for the phrase *to be or not to be* or *The Who*. In systems that use a stop list, none of the query terms would appear in the index, and these queries would fail.

The ideas behind a typical IR system, such as the one we have just described, are quite simple. The system consists of a store of documents and a means for the user to issue a request for information. The answer to the user's information request is a set of documents likely to be relevant to the user's needs. The conventional role of an IR system is to inform users of the existence (or non-existence) of documents related to their request [Van79]. In contrast to traditional database systems, an exact match between the user's request and data (documents) is difficult to achieve because of the flexibility of expression in natural language. In a database system a query can be formulated that completely specifies the information needed. However, in IR systems the specification of what constitutes a relevant document is often ambiguous and incomplete.

Deciding the relevance of a document is a relatively straightforward (although not deterministic nor well-defined) intellectual task for humans. However, computers need a model on which to base relevance judgments. Much of the research in IR has focused on developing such models.

The majority of IR systems compare query and document terms on a lexical, rather than on a semantic level. In order to decide whether a document matches a query, the system must determine whether terms present in the query appear in the document. The problem is that word occurrence in a document does not necessarily mean that the word matches precisely or completely the concept intended by the query. There are two main

problems with basing IR on keyword search. The first problem, called synonymy, is that relevant documents will not be retrieved if they do not contain the terms specified in the query. A search for documents containing the word *cinema* will not necessarily find documents containing synonyms such as *theater* or *movie*. Other problems are homography and polysemy. Homography refers to words that have the same spelling but differ in origin and meaning. Polysemy refers to words that have many meanings. Ambiguous words in a query may cause non-relevant documents to be returned. For example: a search for *walks in Boston* may return documents containing *At home in Boston ... Johnston walks 4 in the bottom of the fifth*, as well as documents about scenic neighborhood walks. These problems arise because many words can represent one concept, and one word can represent several concepts.

A query is normally expressed in a query language and contains a list of terms that are relevant to the user's information need. When formulating a query, the user must try to predict the word combinations that are likely to occur in relevant documents while simultaneously avoiding words that are likely to occur in non-relevant documents. Since it is not possible to predict all possible uses of language, it is highly unusual for a user to formulate a query that both retrieves all relevant documents, and eliminates all non-relevant documents. Normally, some relevant documents will be missed because they include few or none of the query terms, and some non-relevant documents that nevertheless contain terms used in the query will be retrieved.

The difficulty in representing the concepts present in queries and documents is a central problem in IR. Presently there is no satisfactory solution. One intermediary solution is to provide tools that help users select better search terms.

1.2. Retrieval Evaluation

Central to information retrieval is the idea of relevance. The goal of an IR system is to retrieve all the relevant documents while retrieving as few non-relevant documents as

possible. Retrieval performance is normally evaluated by using the interrelated measures of *precision* and *recall*. Precision is defined to be the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Recall is defined to be the ratio of the number of relevant documents retrieved to the total number of relevant documents.

$$precision = \frac{\text{number of relevant documents retrieved}}{\text{total number of documents retrieved}}$$

$$recall = \frac{\text{number of relevant documents retrieved}}{\text{total number of relevant documents}}$$

The ideal IR system would achieve 100% recall and 100% precision for all queries. However, in practice precision and recall tend to vary inversely. A very specific query formulation tends to produce high precision, but it also generally results in low recall. Conversely a broader query formulation is likely to retrieve a greater pool of documents, but the portion relevant is normally smaller, resulting in lower precision [Sal86]. Most attempts at improving one variable tend to have a negative effect on the other.

Calculation of recall requires knowledge of the total number of relevant documents in the collection. For a small document collection this is possible. But for larger collections determining the number of relevant documents may not be practical. Recall figures for larger collections are often calculated by estimating the number of relevant documents. Using sampling techniques is one method for doing this. Another method is to perform a series of searches using various retrieval techniques. The top results of these searches are combined to produce the relevant documents. This technique, called pooling, is based on the assumption that the combination of independent retrieval techniques will retrieve all/most relevant documents.

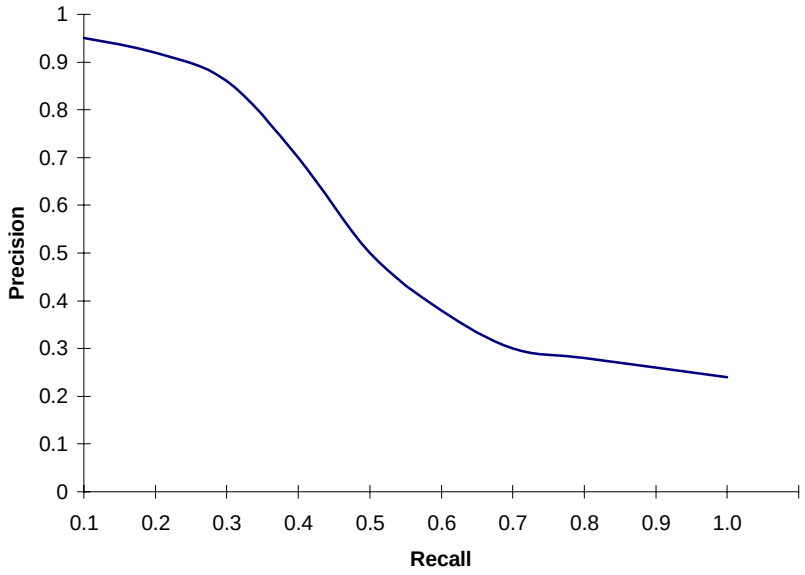


Figure 5: Typical average precision vs. recall graph.

The following chart and graph illustrate an example of a recall-precision computation for a hypothetical query that retrieves a set of 14 documents, including all 7 of the relevant documents in the collection. The documents are listed in ranked order.

Rather than determining relevance on a boolean, "yes or no" answer based on all the terms appearing in the document, a more flexible approach is to compute the probability of relevance based on the number/frequency of query terms that appear in the document. The more terms that occur, the greater the probability that the document is relevant. A document that contains even one of the terms is viewed as a potential answer, but documents that contain all or most of the terms will receive the highest ranks. The ranking approach to information retrieval retrieves documents in decreasing ranked order of likely relevance to the user's query.

Rank	***Document ID*** ***✓indicates relevant***	***Recall***	***Precision***
1	DOC900425-0174✓	0.1428	1.0
2	DOC900712-003✓	0.2857	1.0
3	DOC900611-0080	0.2857	0.6667
4	DOC900518-0197✓	0.4286	0.75
5	DOC880929-0026	0.4286	0.6
6	DOC881102-0259✓	0.5714	0.6667
7	DOC901126-0005	0.5714	0.5714
8	DOC900131-0219✓	0.7143	0.625
9	DOC900128-0036	0.7143	0.5555
10	DOC207342-254✓	0.8571	0.6
11	DOC900102-2314	0.8571	0.5455
12	DOC275289-1034	0.8571	0.5
13	DOC912954-8932✓	1.0	0.5385
14	DOC708416-0921	1.0	0.5

Figure 6: Computation of precision-recall values for ranked query results.

The problem with precision-recall graphs such as the one in Figure 7 is that important information is not represented. Information such as the actual number of documents that were retrieved and the size of the document collection are not present on the graph. Also, the data points are discrete, but the graph is continuous. Multiple precision values can exist for a given recall level, making the graph difficult to interpret.

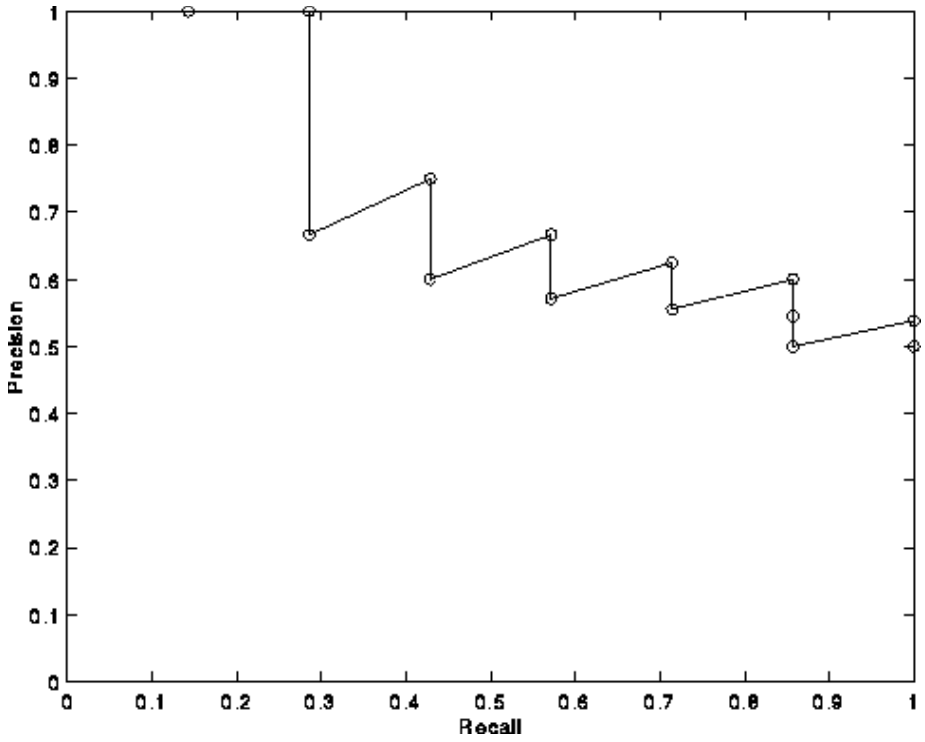


Figure 7: Graph of precision vs. recall for data in Figure 6.

Smooth graphs such as Figure 8 are normally used in place of the raw precision vs. recall plots. The curve in Figure 8 is achieved by starting at the highest point for the highest level of recall and drawing a horizontal line leftward from each peak point of precision to a point vertically below the next peak point of precision (see Figure 9). This interpolated curve is representative of the best performance a user can expect to achieve. For example the interpolated precision at recall 0.10 (after 10% of all relevant documents for that query have been retrieved) is the maximum precision at all recall points greater than or equal 0.10. The interpolated average precision is often used for combining the results from multiple queries and computing average precision-recall values.

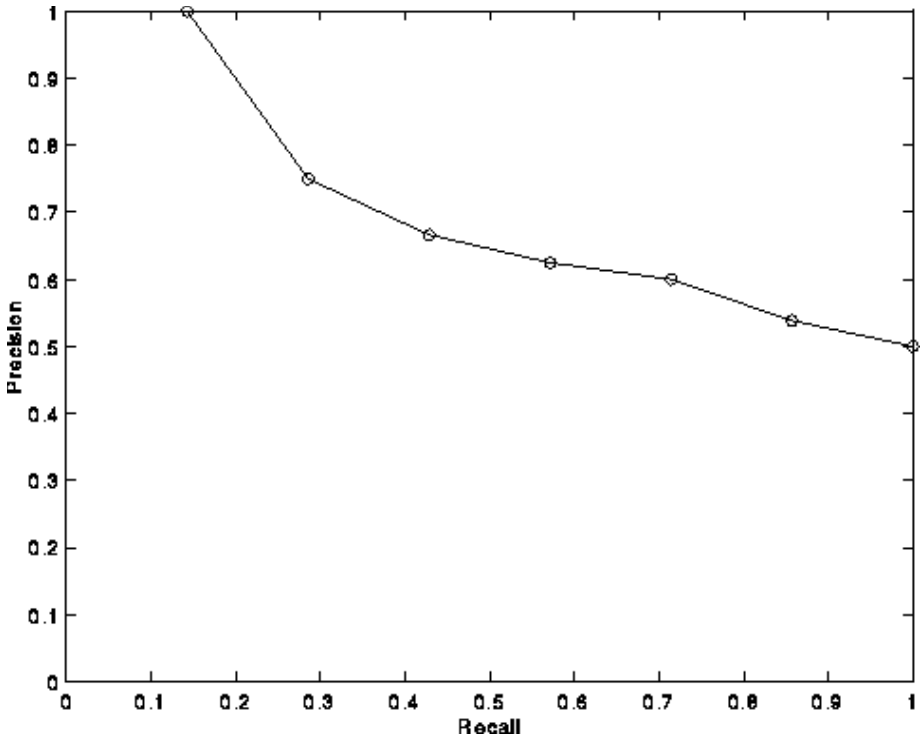


Figure 8: Interpolated precision-recall curve for Figure 7.

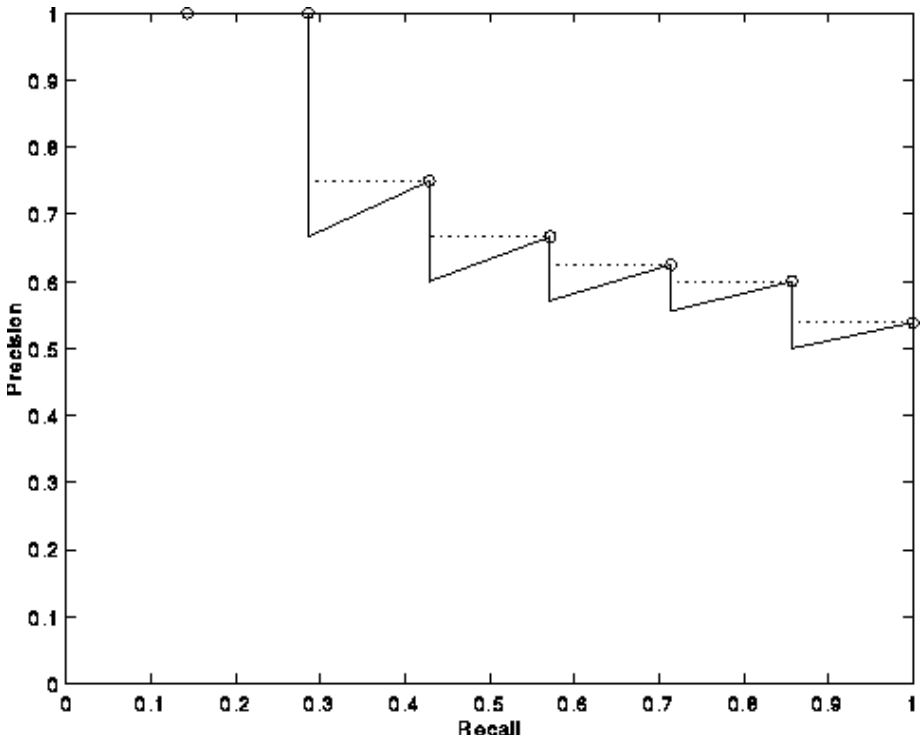


Figure 9: Method for producing interpolated precision-recall curve.

The formula that measures the performance a user can expect from a system can be defined by taking the average over a series of sample queries.

$$\text{Recall}_{RL} = \frac{1}{num}\sum_{i=1}^{num}\frac{\text{Number of Relevant Documents Retrieved}_i}{\text{Total Number of Relevant Documents}_i}$$

$$\text{Precision}_{RL} = \frac{1}{num}\sum_{i=1}^{num}\frac{\text{Number of Relevant Documents Retrieved}_i}{\text{Total Number of Documents Retrieved}_i}$$

where *num* is the number of queries

Since the recall and precision values for each individual query are uniquely defined, the values for Recall_{RL} and Precison_{RL} are also uniquely defined. These values can be used to calculate average precision values for arbitrary recall intervals. One of the standard measures of system effectiveness is to calculate average precision values for each of 11 recall values from 0 to 1.0. When these values are plotted they usually result in a graph similar to Figure 5. The left-hand portion of the graph corresponds to narrow/specific query formulations where precision is high and recall is low. The right-hand portion of the graphs corresponds to broad/general queries where a large number of documents are retrieved, but precision suffers.

Another popular measure is the average precision (non-interpolated) over all relevant documents. This precision is calculated after each relevant document is retrieved. All precision values are then averaged together to get a single number for the performance of the query. For example, for the data in Figure 6 the average precision is 0.74. Conceptually this is the area underneath the precision-recall curve. The values can then be averaged over all queries for an overall system performance measure.

Similarly, precision can be measured after *X* documents (relevant or not) have been retrieved. If fewer than *X* documents are retrieved in total, then all missing documents are assumed to be non-relevant. Experimental results often list the precision for various values of *X (*e.g. 5, 10, 15, 30).

1.3. Problem Statement

In information retrieval the decision of whether a document is relevant for a given query is generally determined by the number and frequency of terms that the document and query have in common. Query expansion is a method of increasing the number of documents matched by a query. The addition of well selected terms to the original query provides the opportunity for more relevant documents to match. The additional relevant documents are retrieved as a result of matches with documents that contain terms related to, but not included in, the original query specification. There are a variety of methods that can be used for query expansion. These include the addition of term variants (morphological, phonetic, typographical), synonyms, and hypernyms/hyponyms. In this thesis we will be exploring query expansion using morphological variants of the query terms.

For example, given the query *Recycling in Waterloo*, obvious morphological variants of *recycling*, such as *recycled, recycles* and *recycle*, would not be considered matches with the term *recycling* unless some form of natural language processing is performed. As this example illustrates, morphological variants of a common term often have a similar meaning. To address the problem of variation in terms, algorithms have been developed to reduce term variants to a root form. This process is referred to as *stemming*, which syntactically approximates the lexicographical notion of *lemmatization*.

Although in many cases morphological variants of a term represent a similar concept, this is not always the case. For example, the following chart lists some terms that share a common root, but do not necessarily represent the same concept.

Root	***Terms***
busi	busy, busied, busies, business, businesses
commun	commune, communicate, communication, communism, community
orient	oriental, orientalism, orientalize, orients, oriented, orientating, orientation, orientability

Figure 10: Examples of terms that share a common root (based on Porter Stemmer).

In IR systems, stemming is usually done when the documents are indexed. In such systems the index for a document collection is based on the stems (roots) of words, rather than on the full text of the term. As seen in Figure 10, a stem usually corresponds to several full terms. The effect of stemming is to expand the original query to include morphological variants of each of the query terms. The disadvantage is that not all morphological variants share the same semantic meaning. So, although useful terms are added to the query, there is the potential for the addition of irrelevant terms to the query.

1.4. Thesis Outline

In this thesis we examine the use of a stemmer for building equivalence classes of words. The resulting classes of words can be used as the basis for a query expansion tool. This tool can be used to provide a facility for leveraging the knowledge of the user during the query expansion process. The goal is to expand the query to include only morphological variants that have the same semantic sense as the original query terms.

The structure of this thesis is as follows: first we present an overview of stemming as a method for query expansion. Then we look at some of the limitations/problems of current stemming techniques. Next, an architecture for a system to build equivalence classes of

words for use in query based stemming is presented. Finally, we present an analysis of the effectiveness of applying stemming at query time to avoid some of the inherent problems of index time stemming.

Chapter 2

2. Background

2.1. Introduction to Stemming

Stemming is a syntactic approximation to a morphological process, and it is used in many IR systems to reduce different word forms to common roots [Frakes92a]. For example the Porter stemming algorithm [Port80] reduces *runs, running, runner* to the common root *run*. One way to view stemming is as the construction of equivalence classes of words. Each class consists of words that have been grouped together based on sharing a common root [CX95]. In our *run* example, *runs*, *running* and *runner* would be grouped together.

Stemming is usually done during the document indexing phase. The terms in the document are stemmed, and the index is built based on the resulting stems. Since each stem typically represents several full terms, the process has the desirable effect of reducing the index size. Figure 11 illustrates the compression rates of various stemmers on different document collections.

IR experiments often use test collections that consist of a document database, queries and a set of relevance judgments. Traditionally, the actual size of test collections has been quite small. Figure 12 gives a brief summary of the contents of the test collections used to produce the compression data in Figure 11.

Stemmer	***Cranfield***	***National Physical Laboratory***	***INSPEC***	***Brown Corpus***
INSPEC [Field75]	32.1	40.7	40.5	47.5
Lovins [Lov68]	30.9	39.2	39.5	45.8
RADCOL [LRK73]	32.1	41.8	41.8	49.1
Porter [Port80]	26.2	34.6	33.8	38.8

Figure 11: Percentage compression using conventional stemming algorithms [LPTW81].

Test Collection	***Number of Words***	***Subject Area***
Cranfield	4,486	Aeronautics
National Physical Laboratory	11,989	Electronics, computing, physics and geophysics
INSPEC title, abstract and free index term fields	19,798	Electronics, computing, physics and control engineering
All words occurring more than once in the Brown corpus	26,067	Very wide range including newspaper cuttings on sports and politics, scientific journal articles, religion and hobbies

Figure 12: Text collections used as source for compression experiment [LPTW81].

The decrease in index size that results from stemming depends on the characteristics of the document collection and on the granularity of the index, as well as the stemming algorithm used. Indexes built at the word or paragraph level will not be compressed as much as indexes built at the document level, since fewer term variants are likely to occur. The more aggressive the stemming algorithm, the more terms that share a common root, and as a result the greater the index compression. The above data indicates that the Porter and Lovins Stemmer are the more aggressive stemming algorithms. These are described in the next section.

As described in Chapter 1, stemming has the effect of expanding the original query to include morphological variants of the query terms that occur in the document collection. Use of these related word forms improve the system's ability to match the query to the document vocabulary, leading to an increase in the number of documents retrieved. Figure 13 shows an example of how the query *trends and developments in retirement communities* might be expanded by the use of the Porter Stemmer. The expansion of most of the terms is relatively consistent with the meaning of the original query terms. However, the expansion of the term *communities* adds several terms that are not appropriate for this query. If documents in the collection contain several such misleading terms, they will be returned to the user in spite of their non-relevance to the original query.

The disadvantage of stemming is that for each useful term added, some non-useful or misleading terms may also be included. These misleading term expansions cause non-relevant documents to be retrieved, and may negatively influence document ranking. Stemming is an effective method for increasing recall. However, the problem of non-relevant terms being added to the query needs to be addressed.

Query Term	***Terms Added by Use of the Porter Stemmer***
trends	trend trended trending
developments	development develop developers developing
retirement	retire retires retired retiring
communities	commune communes communicate communicated communicates communication communicator communism communities community

Figure 13: Selections from query expansion of *trends and developments in retirement communities.*

2.2. Common Approaches

A variety of methods have been suggested as effective stemming algorithms. These include suffix removal, strict truncation, word segmentation, n-grams, and linguistic approaches to morphology [LPTW81]. The stemmers commonly used in IR systems reduce words to stems by the removal of suffixes. Two of the more popular suffix removal algorithms were suggested by Lovins [Lov68] and Porter [Port80].

Iterative methods for stemming algorithms are based on the observation that more than one suffix is often appended at the end of a word. Such a method attempts to peel the suffixes off one at a time. A word such as *successfulness* would have *-ness* removed on the first iteration, and *-ful* on the second. Alternatively, longest-match algorithms involve a single iteration only, during which the longest recognized suffix is removed. A longest match system includes a master list of recognized suffixes and compound suffixes, called a suffix dictionary. Using a longest match stemming algorithm, where the suffix dictionary contains the suffixes *-ful*, *-ness* and *-fulness*, the word *successfulness* would be

reduced to the root *success* by the removal of *-fulness* [LPTW81]. Longest match algorithms are easier to code, but they require a much larger dictionary (suffix list) to accommodate all the possible combination of suffixes. Some stemmers implement a combination of these approaches, known as an iterative longest match algorithm [Frakes92a].

The first longest match stemmer was developed by Lovins [Lov68]. The Lovins Stemmer takes a two phased approach. The first phase consists of the actual stemming algorithm. This involves identifying the stem by removing the longest possible suffix that matches a pre-defined list. The second phase handles spelling exceptions that arise. The term "spelling exceptions" is used by Lovins to refer to cases where a stem may be spelled in more than one way. Figure 14 gives examples of the types of spelling exceptions that can occur. The two main approaches for dealing with these types of exceptions are recoding and partial matching.

Recoding involves making changes to stems that have identified spelling variations, so that they will ultimately conflate to the same class. The recoding procedure is incorporated into the stemming process and takes place immediately following suffix removal. The order that the recoding rules are applied is important. An example of recoding rules is given Figure 15 [Lov68].

The recoding procedure makes the assumption that most of the exceptions are predictable and can be handled by a small set of transformational rules. However, it is not possible to predict all exceptions, and there will be situations where terms are accidentally transformed. Such mistakes are not expected to be frequent enough to significantly effect the stemming process, much less the performance of later document retrieval.

producer → produc
production → product

consumed → consum
consumption → consumpt

induced → induc
induction → induct

attending → attend
attention → attent

expanding → expand
expansion → expans

input → input
inputting → inputt

registering → register
registration → registr

resolved → resolv
resolution → resolut

circle → circl
circular → circul

matrix → matrix
matrices → matric

hypothesized → hypothes
hypothetical → hypothet

analysed → analys
analyzed → analyz
analytical → analyt

Figure 14: Examples of spelling exceptions [Lov68].

1.	Remove one of double b,d,g,l,m,n,p,r,s,t	inputting⇒inputt⇒input
2.	iev ⇒ ief	believes⇒believ⇒belief
3.	uct ⇒ uc	induction⇒induct⇒induc
4.	umpt ⇒ um	consumption⇒consumpt⇒consum
5.	rpt ⇒ rb	aborption⇒absorpt⇒absorb
:		
24.	end ⇒ ens except following s	extended⇒extend⇒extens
:		
28.	her ⇒ hes except following p,t	
29.	mit ⇒ mis	admitting⇒admitt⇒admit⇒admis
30.	end ⇒ ens except following m	
:		
33.	yt ⇒ us	
34.	yz ⇒ ys	analyze⇒analyz⇒analys

Figure 15: Sample of transformational rules used in recoding stem terminations.

An alternative approach for dealing with such exceptions is partial matching. The partial matching method assumes that rules for the exceptions are indicative of changes that *may* occur. Partial matching is not included as part of the stemming procedure. Instead, it is applied to the index that is created from the output of the stemmer. Retrieval from the index using this method does not require exact matches. Within certain pre-defined limits, such as illustrated in Figure 16, partial matches are retrieved from the index. These matches are then further processed to produce the final result.

1 start with the stem S_1 of a query term

2. search the index for all stems that begin with S_1 minus its last two letters

3. discard all stems that are more than two characters longer than S_1

4. the result is a set of terms that will then be processed further to determine the final list of matches for the query term

Figure 16: Example of a partial match algorithm [Lov68].

Partial matching algorithms are clearly more flexible than the use of predetermined recoding rules. This flexibility may not produce fewer wrong matches, but it has the potential for producing more correct matches The disadvantage to this method is an increase in index size, as well as an increase in retrieval overhead.

The Porter stemmer is based on the iterative approach and uses a five step process consisting of a set of condition/action rules. Despite involving more phases, code for the Porter algorithm is actually more compact than for the Lovins algorithm [Frakes92a]. Whereas Lovins has 260 suffixes (many of them compound) in its suffix dictionary, the Porter Stemmer recognizes only 60 simple suffixes.

The first phase of the Porter algorithm is the removal of plurals and past participles. The next three phases involve suffix stripping based on a pre-defined list of recognized suffixes, as well as context sensitive rules. Simple suffixes are removed in a series of steps that are dependent on the form of the remaining stem (measured in syllables). The final phase of the algorithm takes care of tidying up stem endings (e.g. removal of double constants controll⇒control), much like the recoding rules described for the Lovins algorithm. A more detailed presentation of the Porter algorithm appears in Appendix A.

There are some obvious problems associated with these methods of conflation. All of these stemming methods ignore word meaning. This leads to errors when words with different meanings are conflated, or when words with similar meanings are not conflated.

Example : Words with different meanings that are conflated using the Porter Stemmer

responsive and *responsibility*
oriental and *orienteering*
factory and *factorial*

Example: Words with similar meanings that are not conflated using the Porter Stemmer

matrix and *matrices*
absorb and *absorption*
angle and *angular*

The stems that are produced by suffix removal algorithms are often not valid words. For example the Porter Stemmer returns *locat* as the stem for the word *located*. This makes it difficult to use the output from a stemmer for purposes other than retrieval (e.g., word frequency or dictionary lookup). Interactive techniques that require user input for term selection (i.e., for query expansion) are made more difficult when the full text of the words is no longer available (for an example see [HWW86]).

2.3. Evaluating Stemming Algorithms

Establishing a quantitative measure of the effectiveness of a stemming algorithm is a difficult task. One possible measure is to compare the stems produced by the automatic algorithm with those produced manually by a person [LPTW81]. However, this makes the assumption that the stem produced by the human is correct and the one most effective for retrieval. In IR, producing the linguistically correct root is not as important as grouping terms together that will increase retrieval effectiveness.

There have been several studies that analyze the impact of stemming algorithms on IR performance. A good overview of the various studies can be found in [Frakes92a]. None of the more popular stemmers seem to decrease the overall retrieval effectiveness, but the beneficial effects of stemming vary and are sometimes quite small. The average performance of the common stemming algorithms is quite similar [Frakes92a][Krov93]. Collections with short documents generally show greater increases in performance over collections with longer document lengths. Krovetz reports an increase of 15-35% in

precision at various levels of recall when stemming was used with the CACM and NPL test collections [Krov93], which both contain short documents and queries.

Most research on stemming has concentrated on producing new algorithms. Much of the focus has been on domain/subject specific stemmers (e.g. those for medical applications [PP78][UD83]) or linguistic based stemming algorithms (i.e. those based on derivational and inflectional information [Hull96][Krov93]).

	No Stemming	***S Stemmer***	***Porter Stemmer***	***Lovins Stemmer***
Cranfield[1]				
number of terms added	10	16	28	39
total number of documents retrieved	736	837	886	943
Medlars[2]				
number of terms added	11	18	28	40
total number of documents retrieved	296	356	398	444
CACM[3]				
number of terms added	13	22	47	58
total number of documents retrieved	894	1253	1403	1459

Figure 17: Average query expansion and documents retrieved by stemming [Har91].

When stemming does not provide meaningful improvements, it is not because retrieval is unaffected. As Figure 17 illustrates, experiments have shown that the number of documents retrieved is consistently increased by the use of stemming. However, for each of the stemming techniques, it is not uncommon to find that the number of queries

[1]Aeronautics domain, 1400 documents, 225 queries.

[2] Medical domain, 1033 documents, 33 queries.

[3] Computer Science domain, 3204 documents, 64 queries.

that show improvements in performance is rivaled by the number of queries that show a degradation in performance (see Figure 18). Thus, the increase in matching documents needs to be offset by an increase in the discriminating power of the ranking algorithm. Very little research has been done on the interaction of stemming and ranking algorithms.

	Queries with improvement in performance	***Queries with decrease in performance***
Cranfield (255 queries)		
Lovins vs. full word	51	44
Porter vs. full word	49	37
S stemmer vs. full word	34	32
Medlars (30 queries)		
Lovins vs. full word	8	9
Porter vs. full word	8	11
S stemmer vs. full word	6	7
CACM (64 queries)		
Lovins vs. full word	23	12
Porter vs. full word	20	10
S stemmer vs. full word	12	7

Figure 18: Comparison of effect of stemmers on query (based on top 10 documents retrieved) [Har91].

	No Stemming	*S Stemmer*	*Lovins Stemmer*	*Porter Stemmer*
Cranfield				
Average precision for three intermediate points of recall	0.377	0.397	0.388	0.402
Total relevant retrieved				
At 10 documents retrieved	650	654	655	666
At 30 documents retrieved	946	958	984	972
Medlars				
Average precision for three intermediate points of recall	0.522	0.538	0.574	0.539
Total relevant retrieved				
At 10 documents retrieved	188	187	190	185
At 30 documents retrieved	387	389	412	394
CACM				
Average precision for three intermediate points of recall	0.304	0.323	0.318	0.319
Total relevant retrieved				
At 10 documents retrieved	153	159	171	169
At 30 documents retrieved	278	293	324	313

Figure 19: Comparison of retrieval performance of stemmers on Cranfield, Medlars and CACM collections [Har91].

Like IR systems more generally, stemming experiments are evaluated by measures such as average precision and recall. The common methodology is to:

1. Propose a new stemming method which is designed to improve retrieval performance.
2. Find an experimental text collection with queries and known relevant documents.
3. Run experiments using the new strategy and a baseline obtained from a standard approach.
4. Compute traditional evaluation measures such as recall and precision.

The average measures do not always describe adequately the overall performance of an algorithm. There is a great deal of additional information that can be discovered by analyzing the performance of individual queries across methods. This is particularly important with stemming, where there does not seem to be a significant difference in average performance between the commonly used algorithms.

Hull suggests that there are three major patterns which lead to high variance in precision [Hull96]:

1. Many related documents are ranked higher by some method(s).
2. The query has a few relevant documents, and as a result scores are sensitive to changes in the rankings of a few relevant documents.
3. Relevant documents tend to have low ranks due to the difficulty of the query. Some methods manage to rank a few documents well, which boosts their evaluation measures.

The first scenario is the preferred behavior for a retrieval method. A method that produces scenarios two and three may be valuable, but such results are a less reliable indicator than improvement in the ranking for a large number of the relevant documents.

Analysis of average performance figures does not explain why or when one stemming algorithm works better than another. In order to understand the difference in stemmer performance, the analysis needs to examine specific examples where the stemming algorithm makes a significant difference. Hull suggests that a detailed analysis of the results of queries that produce the first scenario is valuable for making general conclusions about the performance of different experimental methods. This type of analysis is also key to motivating improvements to current stemming algorithms.

Hull has proposed that in addition to the conventional evaluation measures a method that involves ranking the scores for each algorithm on a query by query basis be used. Evaluation results can be summarized by computing the average rank for each method.

Statistical methods can be applied to the ranking data to determine if they represent a significant difference between algorithms. Since ranked-based analysis is a relative measure, it should not be used alone, but rather as a complement to traditional evaluation measures [Hull96].

Chapter 3

3. Query Based Stemming: Indexing Phase

3.1. Introduction to Query Based Stemming

The addition of useful terms that help to represent the user's information need seems an intuitive way to improve retrieval effectiveness. But, as we saw with stemming, when expansion occurs inappropriately, retrieval effectiveness is often adversely effected. Inappropriate expansion can occur in a variety of ways, depending on the expansion technique being used. When term co-occurrence is used, terms may be added that co-occur with a sense of the term other than the sense used in the query. For example the query, *fishing on the bank of the Mississippi*, could have terms such as *financial*, *money* etc. added based on co-occurrence with the word bank. The result is a skew of the query towards documents that deal with the financial institution sense of the word *bank*. With stemming, problems often occur when morphologically similar words which are not conceptually equivalent are included (e.g., author expanded to include authority and authoritarian).

A weakness inherent in most query expansion techniques is that expansion terms are selected based on how strongly they relate to *one* of the query terms. A term-term similarity measure is computed, and terms that are above a certain threshold are added to the query. However, a large number of terms that would be viewed as consistent with the concept expressed by the query often do not meet the threshold for term-term similarity [QF93]. Although this similarity measure is a good starting point, a better criterion for query expansion is consistency with the overall query concept.

The weakness of the conventional implementation of stemming at index time is that the query expansions are pre-determined. The retrieval system does not have the ability to alter the expansion dynamically based on query context. As a result a given term will always be expanded in the same way, without regard for the concepts represented by the query in which it occurs.

When an index stores only the word stems, it fixes at index time an equivalence relationship between all words that share the same stem. Specifically, posting information is stored based on stems, and the original terms are lost to the query and ranking processes. This introduces the disadvantage that all occurrences of words with the same stem are treated identically. Since *commune* and *community* have the same stem, they are viewed as equivalent. The system does not distinguish that *commune* is a variant of the original query term *community*, and that a document containing *commune* may not be as good a match as a document containing the original *community*

Croft has identified stemming as "one of the main sources of occasional bad mistakes…" in information retrieval [Croft95]. Clearly what is needed is a facility that reduces the number of bad mistakes caused by stemming, while retaining the effective gains. This can be accomplished by only expanding a query to include variants that are both *consistent* and *close* in meaning with the intent of the query.

No automatic technique has been found for separating out useful term variants (for a given query) produced by stemming, from the non-useful ones [Har88]. Filtering out bad term expansions has proved a difficult task to perform automatically (see section 3.2).

A problem with fully automatic query expansion, such as stemming, is that it tends to diminish the role of the user. Leaving the user out of the interaction can lead to confusion when modifications are made to the query which are not consistent with its original intent. For example a user looking for articles about *The Lakers*, does not want to be presented with a large number of articles about lakes and streams that resulted from unsolicited query expansion.

It is worth considering the user as more than a mechanical receptor of data produced by the system. The user could be a potential source of intelligence that can be leveraged to strengthen the quality and effectiveness of a retrieval system. What has not been explored in stemming research is whether the users' knowledge can be incorporated effectively to determine when to expand query terms, and what appropriate term expansions should be.

An approach that has been suggested but not explored is the idea of moving stemming from index time to query time. When stemming is moved to query time, there is an opportunity to filter the terms that will be added to the query. In query based stemming all decisions about word conflation are made dynamically when the query is formulated, rather than statically at index time [CX95].

Query based stemming greatly increases the flexibility of stemming. The full word form is used for indexing, and stemming becomes part of query processing. An aggressive stemmer is used to identify the words that could be conflated, and then user judgments are used to initiate the conflation. When a query is entered, the equivalence class for each query term is used to generate a set of options for expanding the query.

The advantages of performing stemming at query time are that the user of the system can be consulted as to the applicability of particular word forms, and queries can be

restricted to search for a specific word form. For example, when looking for articles about specific drug related deaths, the word *over-dose* is a good discriminating term. However, the word *over-doses* is less useful, as it will tend to be used in more general stories. This is an example of a situation in which a user could decide that it is better not to expand a term. These advantages can be significant in cases where small differences in word forms can result in large differences in the relevance of the retrieved documents. Query based stemming can be used as a tool to offer guidance to users wishing to improve their queries in a full-text environment.

One disadvantage to query expansion is that queries tend to become longer, and that can potentially increase processing time. Although the query will become larger when stemming is performed at query time, the effective term expansion actually decreases, because not all expanded terms will be selected.

3.2. Previous Experiments With Selective Stemming

The problem with using stemming as an *automatic* method for query expansion is that some inappropriate terms are added. Defining what constitutes an inappropriate, term and developing methods to automatically exclude them from the query expansion is a difficult problem.

Ideally stemming should only be applied in situations where it results in improved performance. Donna Harman performed a series of studies where she tried to perform selective stemming based on the collective results of earlier stemming research [Har91]. Her study was an attempt to predict and apply stemming when it is advantageous.

The first approach was based on the observation that shorter queries and collections with shorter length documents seem to benefit the most from stemming. Intuitively this makes sense since making a match between queries and documents is difficult when they contain few terms. Tests were performed against the Cranfield collection where only queries with fewer than ten terms (≅ 54% of the collection) were stemmed. The results

proved to be no better than without stemming, but worse than automatically stemming all terms.

The next approach was based on the hypothesis that stemming should only be applied to *important* terms in the database. Since terms that are widespread in the database are not generally good discriminators, adding morphological variants of these terms is likely to degrade the query. The results of the experiments involving the stemming of only important terms showed no improvements over full stemming. Term distribution in the database is apparently not a good indicator of whether a term in a specific query will benefit from stemming.

The third approach that was tested was altering the ranking algorithm so that terms that were added via stemming were weighted as less important than the original query terms. When added terms were heavily down weighted, the results were similar to no stemming. Without down weighting, terms with many variants tended to dominate the query. No compromise was found.

Automatic selective stemming has proved to be a difficult problem. This opens the question whether or not human interaction in the selective stemming process might improve the results.

Harman performed some initial experiments to explore the feasibility of a system that would allow the user to provide the necessary separation of useful and non-useful terms produced by stemming [Har88]. The goal of these experiments was to estimate how much improvement could be gained by expanding a query with term variants that were chosen by a user. No users were involved in this experiment, but user filtering was simulated. Initially the user's choice was simulated by choosing all term variants that appeared in relevant documents retrieved by the non-expanded query. The method had a positive, but limited, effect on retrieval. The second method used was to simulate a perfect user's choice. The perfect user would choose all the term variants that appear in relevant documents in the collection. This method showed significant (≈ 8%) improvements. The

conclusion of this study was that there is evidence that a user could provide the necessary filtering of expansion terms based on stemming and that substantial improvement in retrieval effectiveness should be possible.

3.3. Overview of Inter-Stem System Architecture

The premise behind the Inter-Stem system is to allow users to play an active role by allowing them to determine what query terms should be expanded and which of the potential term expansions are most appropriate.

This system is based on the idea that stemming really acts as the construction of equivalence classes of words. Each class contains a set of words that appear in the document collection and share a common stem. For example the words *beam*, *beams*, *beamed* and *beaming* appear together in a class because they all share the common stem *beam*.

The role of the stem is to be the key by which terms in the classes are accessed, rather than a representative that replaces the original terms. The system acts as a front end interface to be used with existing retrieval systems (illustrated in Figure 20). The current implementation of the system assumes that the back end retrieval system can perform exact match, case independent searches.

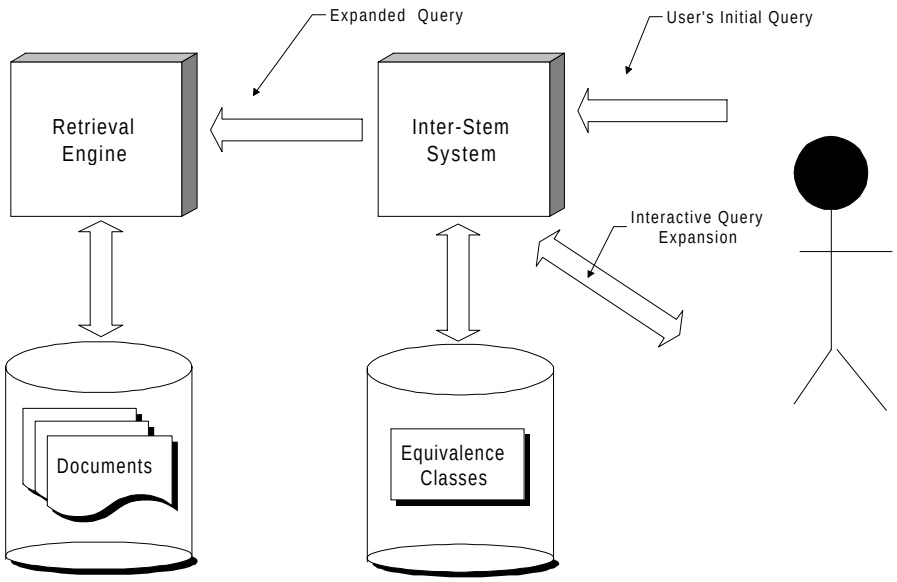


Figure 20: The Inter-Stem System.

There are two main components to the Inter-Stem system: the equivalence class generator, and the query expansion module. The equivalence class generator is the tool that processes the document collection and builds classes of words based on stemming. Similar to stemming at index time, the Inter-Stem System forms equivalence classes that are based on the actual terms that appear in the document collection. Classes are built by pre-processing the document collection and parsing it into terms. Each of these terms is then stemmed and placed into an appropriate class based on the resulting stem. Figure 21 illustrates the phases involved in the construction of the equivalence classes. These classes will then be used by the query expansion module for interactive query expansion. The query expansion module is designed to read in queries expressed in a query language (the current implementation accepts queries written in GCL [CCB95a]) and expand terms identified by the user. The proposed expansions are then presented to the user, who determines which expansions will be added to the query. Once the user is satisfied with the expanded query, it is passed along to the retrieval engine for processing.

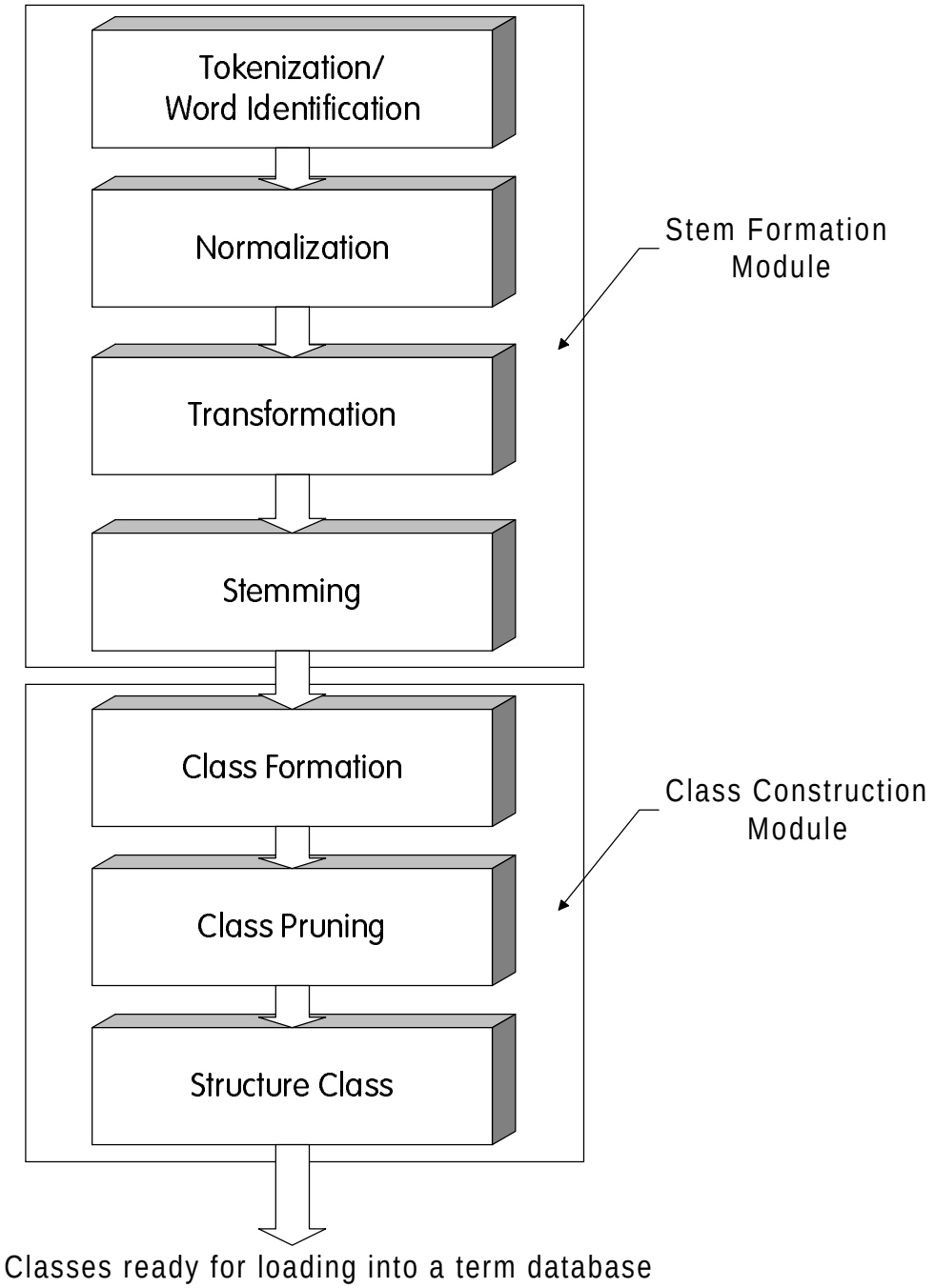


Figure 21: Phases for construction of equivalence classes

3.4. Stem Formation Module

The first module of the equivalence class generator is responsible for identifying terms and producing the associated stem for inclusion in the equivalence classes. The input to the stem formation module is a collection of documents. The output is a stream of normalized term and stem pairs.

3.4.1. Tokenization/Word Identification

The first phase of the class construction process is to parse the document collection into tokens. The purpose of the tokenization stage is to eliminate noise (i.e., data not to be included in term expansion), while identifying terms to be included in the equivalence classes. The strategy adopted for parsing the document collection is to adopt a conservative definition of what constitutes a word in the document. Thus for example, SGML tags, numbers, and acronymns (e.g. *<DATE>, 01/02/92, IBM)* are not considered words. This approach was decided upon in order to avoid many uninflectable terms. It was felt that the system would lose credibility if the user was presented with noisy lists for term selection.

The tokenizer parses the text into the token types listed in Figure 22. `Word` and `word with hyphen or apostrophe` are identified as distinct tokens in anticipation that some transformations will be necessary on tokens of the latter type before they are passed along to the stemmer.

The purpose of the present system is to build equivalence classes based on words. However, it is conceivable that equivalence classes based on SGML or numbers could be useful. For example, SGML tags such as <doc>, <document>, etc. might be viewed as equivalent for query expansion purposes (especially queries based on structure or proximity). For this reason, SGML and numbers are identified as separate tokens, although they are not processed in the present implementation. The decision was made that words that contain mixed case (other than initial capital letter) or embedded numbers

are most likely noise and should be discarded. Words that are all upper case are also discarded. The majority of words that appear in all upper case are either acronyms or proper names (i.e. USA, NAFTA, WATCOM, APPLE, IBM) and would not be suitable for inclusion in the equivalence classes.

Token Name	***Properties***
word	Series of either all lowercase alphabetic, or initial uppercase followed by all lower case alphabetic.
word with hyphen or apostrophe	A sequence of ***words*** (as defined above) connected by one or more non-sequential apostrophes or one or more non-sequential embedded hyphens.
word with embedded caps	A **word** (as defined above) but containing one or more upper case letters in a position other than the initial character.
word with embedded number	A **word** (as defined above) that contains one or more numbers.
number	A string of digits that may include one or more non-sequential decimal points and/or one or more non-sequential commas.
markup	Anything enclosed in < >, including the angle braces themselves.
other	Does not satisfy any of the previous token definitions.

Figure 22: List of token types.

3.4.2. Normalization

The normalization process is used to eliminate variations in the data with which the user need not be concerned. Any variation in the data that is not supported by the targeted back end retrieval system is a candidate for normalization. A typical normalization operation is the conversion of all tokens to lower case. Without this normalization operation *Not* and *not* would be considered distinct words (see discussion and example in Section 1.1). If the back-end system is restricted to performing case insensitive searches, there is no need to preserve case in the equivalence classes. Furthermore, case-folding decreases the number of distinct tokens to be passed on to the transformation phase. Some retrieval systems treat hyphens as whitespace. If the retrieval system has proximity operators, hyphenations may be preserved in the classes so that proximity operators can be implemented in the query expansion stage to mimic hyphenation. However, if proximity operators are not available, or not used, hyphenated words may be normalized by decomposing them into their constituent words.

3.4.3. Transformation

The role of this phase is to perform any transformations necessary for the token to be accepted and processed by the stemmer. The amount of transformation necessary is dependent on the stemmer, and this phase can be customized to the stemmer being used. The current implementation of Inter-Stem uses the Porter Stemmer, which was chosen because it is an aggressive stemmer and publicly available.

The Porter Stemmer does not have the ability to handle words that contain any kind of punctuation. This includes hyphens and apostrophes. Rather than modify the stemmer, the tokens of the type `word with hyphen or apostrophe` are modified before being sent to the stemmer. For example the word *wanderin'* is transformed to *wandering*.

3.4.3.1. Apostrophe Transformation

An apostrophe is used to indicate the possessive, the omission of one or more letters from a word, and plurals. The purpose of the transformation stage is not to replace missing letters, but to add the token to the most appropriate equivalence class and to determine the most appropriate word form to be passed along to the stemming phase.

A list of common occurrences of apostrophes was development by analyzing a large[4] test collection. This collection was tokenized using the method described in Section 3.4.1. All the terms of type `word with hyphen or apostrophe` that contain one or more apostrophes (some terms contain hyphens only) were sorted into groups and a list of the most frequently used forms of apostrophes was compiled. The results of this analysis appear in Appendix B. Based on this analysis the following transformational rules were developed.

Suffixes

1. **in', 'ing → ing**
   (*examples* : fixin', blowin', spec'ing, ho-ho-ho'ing)
2. **' , 's, n't , 'll, 'em, 've, 'd, 're, 'n → removed**
   (*examples* : sink'n, nuke'em, maitre'd, they're)

Prefixes

1. **a', o', j', l', n', d' → removed**
   (*examples* : a'roaming, o'clock, j'ai, l'ville, d'etats)
2. **e' → e**
   (*examples* : e'tude, e'vent, e're)

[4] Approximately 2 gigabytes of text.

Other

For all other cases the apostrophe was merely removed. Through various trials it was determined other processing introduced as many errors as were fixed.

3.4.3.2. Hyphen Transformation

A hyphen is used between the parts of a compound word, or between the syllables of a word when the word is divided at the end of a line of text. Once again, in order to develop an algorithm for transforming words containing hyphens, a list of common occurrences of hyphenated words was developed. Several transformations of hyphenated words were considered.

The list of common occurrences of hyphenation was developed by analyzing the same large test collection. Some examples of the hyphenated words that were considered are: *Computer-aided*, *over-the-counter*, *m-i-c-k-e-y m-o-u-s-e*, *re-election* and *table-top*. There does not seem to be any clear algorithmic way of identifying which part of a compound word should be used to determine the target equivalence class. In the case of *re-election*, the second term, *election* is most appropriate for determining class membership. For *table-top*, the first term is more appropriate. Since word order cannot be used to determine which part of the term should be sent to the stemmer, it was decided to allow hyphenated words to be added to multiple classes.

Example :

computer-aided appears in both the *computer* and *aided* equivalence classes

over-the-counter appears in the *over*, *the*, and *counter* classes.

3.4.4. Stemming

The stemming phase takes each term produced by the transformation phase and produces its stem. The choice of stemming algorithm determines the composition of the classes produced, so the algorithm should be chosen carefully. An aggressive stemming algorithm was chosen to aid in building comprehensive equivalence classes. A weak stemming algorithm would tend to build small, limited classes. Since the user will be deciding what terms from the class will be added to the query, the introduction of more than one concept in a class does not introduce the problems that aggressive algorithms pose in traditional applications of stemming.

3.5. Class Construction Module

The second module of the equivalence class generator is responsible for taking the list of normalized term and stem pairs produced by the stem formation module and constructing equivalence classes. These classes can then be loaded into a database for use by the query expansion module.

3.5.1. Class Formation

When processing the term/stem pairs, the stem is used as the key to determining in which equivalence class the normalized term will appear. If a term already appears in a class it is not added. A class may optionally contain frequency or other statistical information about the occurrences of its members. The class formation process produces a list of terms, beginning with the stem, such as:

agenc agencies agency agencys
brass brass brasse brassed brasses
catch catch catches catching catchings

Notice that not every item need be a proper word. Stems are often not proper words, but since they are used only for access and are not presented to the user, this is not a concern.

This phase also handles such idiosyncrasies as the cases where *a* stems to *b*, but *b* stems to *c*. The Porter Stemmer occasionally makes such incomplete conflations. This phase is responsible for ensuring that *a*, *b*, and *c* all appear in the same class. An algorithm for collapsing classes in this manner is presented by Croft and Xu [CX95].

3.5.2. Class Pruning

In this phase a decision is made whether or not to eliminate some of the classes. For example, the word *m-i-c-k-e-y* would appear in the *m*, *i*, *c*, *k*, *e*, and *y* classes. It is doubtful that one letter classes will be useful for expansion, and thus can be eliminated. Classes that consist wholly of stopwords, or that have only one member with a very low frequency are also candidates for pruning.

3.5.3. Structure Class

In order to speed up access to the equivalence classes at query time, Inter-Stem loads then into a text database. By annotating the classes with markup, the classes can be structured. Initially a simple set of tags have been used, but these may be modified later to introduce more internal structure to the classes.

Example:

<stem>***agenc***</stem><term> agencies agency agencys</term>
<stem>***brass***</stem><term>brass brasse brassed brasses</term>
<stem>***catch***</stem><term>catch catches catching catchings</term>

Depending on the size of the classes it may be beneficial for the terms to be ordered. However, this is probably better performed dynamically at query time, since the optimal order of terms is dependent on the query context (see Chapter 4).

3.6. Sample Equivalence Class Generation

The following takes the example text in Figure 23 and traces the processing performed by the equivalence class generator.

<doc><doc-no>SHAK-PLAY-ROMEO-AND-JULIET</doc-no>

<act><act-number> 1 </act-number><scene><scene-number> 1 </scene-number>

<direction>Enter Sampson and Gregory, with swords and bucklers, of the house of bucklers </direction>

<speaker>**Sampson**:</speaker><speech>Gregory, on my word, we'll not carry coals.</speech>

<speaker>**Gregory:**</speaker><speech>No, for then we should be colliers.</speech>

<speaker>**Sampson:**</speaker><speech>I mean, an we be in choler, we'll draw.</speech>

<speaker>**Gregory**:</speaker><speech>Ay, while you live, draw your neck out of collar.<speech>

<speaker>**Sampson**:</speaker><speech>I strike quickly, being moved.</speech>

<speaker>**Gregory:**</speaker><speech>But thou art not quickly moved to strike.</speech>

<speaker>**Sampson**:</speaker><speech>A dog of the house of Montague moves me.</speech>

<speaker>**Gregory:**</speaker><speech>To move is to stir, and to be valiant is to stand. Therefore, if thou art moved, thou runn'st away.</speech>

<speaker>**Sampson**:</speaker><speech>A dog of that house shall move me to stand. I will take the wall of any man or maid of Montague's.</speech>

<speaker>**Gregory**:</speaker><speech>That shows thee a weak slave; for the weakest goes to the wall.<speech>

<speaker>**Sampson**:</speaker><speech>'Tis true; and therefore women, being the weaker vessels, are ever thrust to the wall. Therefore I will push Montague's men from the wall and thrust his maids to the wall. </speech>

<speaker>**Gregory**:</speaker><speech>The quarrel is between our masters, and us their men.</speech>

<speaker>**Sampson**:</speaker><speech>'Tis all one. I will show myself a tyrant. When I have fought with the men, I will be cruel with the maids--I will cut off their heads. <speech>

<speaker>**Gregory:**<speaker><speech>The heads of the maids?<speech>

<speaker>**Sampson:**<speaker><speech>Ay, the heads of the maids, or their maiden-heads. Take it in what sense thou wilt.</speech>

<speaker>**Gregory**:</speaker><speech>They must take it in sense that feel it.</speech>

<speaker>**Sampson**:</speaker><speech>Me they shall feel while I am able to stand; and 'tis known I am a pretty piece of flesh.</speech>

<speaker>**Gregory**:</speaker><speech>'Tis well thou art not fish; if thou hadst, thou hadst been Poor John. Draw thy tool! Here comes the house of Monagues.

... </doc>

Figure 23: Sample text, an excerpt from Romeo and Julliet [Shaks]

3.6.1. Tokenization

The purpose of the tokenization phase is to eliminate noise and identify candidate terms for the equivalence classes. Figure 24 shows how the sample text would be broken into tokens. Notice that *SHAK-PLAY-ROMEO-AND-JULIET* is of token type `other`. All tokens of type `other` are discarded. In this example, duplicate tokens have been eliminated for the sake of brevity.

Token Type	***Tokens***
number	1
markup	<doc> <doc-no> </doc-no> <act> <act-number> </act-number> <scene> <scene-number> </scene-number> <direction> </direction> <speaker> </speaker> <speech> </speech>
word with hyphen or apostrophe	we'll run'st Montague's maiden-heads ‘Tis ‘tis
word	Enter Sampson and Gregory with swords bucklers of the house Capulet on my word not carry coals No for then we should be colliers I mean an in choler draw Ay while you live your neck out collar strike quickly being moved But thou art to strike A dog Montague moves me To move is stir valiant stand Therefore if away that shall will take any man or maid That shows thee a weak slave wall weakest goes true therefore women weaker vessels are ever thrust push men from his maids The quarrel between our masters us their all one show myself tyrant When have fought civil cut off heads Take it what sense wilt They must that feel Me they am able known pretty piece flesh well art fish hadst been Poor John Draw thy tool Here comes two Montagues
Other	SHAK-PLAY-ROMEO-AND-JULIET

Figure 24: Results of tokenization of sample text from Figure 23.

3.6.2. Normalization

Only tokens of type `word` or `word with hyphen or apostrophe` are processed by the normalization phase. The normalization process is used to eliminate variations in the data that need not be of concern to the user. The assumption is made that the back-end system has the ability to handle hyphens, and as such they are not normalized. In this example the only normalization that takes place is the conversion of all tokens to lower case. Terms that appear in boldface in Figure 25 are the terms that were affected by the normalization process.

we’ll
run'st
montague's
maiden-heads
‘tis
‘tis
enter
sampson
and
gregory
with
swords
bucklers
of
capulet
the
house
on
my
word
not
carry
coals
no
for
then
we
should
be
colliers
i
mean
an
in
choler
draw
ay
while
you
live
your
neck
out
collar
strike
quickly
being
moved
but
thou
art
to
a
dog
of
montague
moves
me
to
move
is
stir
valiant
stand
therefore
if
away
that
shall
will
take
any
man
or
maid
that
shows
thee
a
weak
slave
wall
weakest
goes
true
therefore
women
weaker
vessels
are
ever
thrust
push
men
from
his
maids
the
quarrel
between
our
masters
us
their
all

one
show
myself
tyrant
when
have
fought
civil
cut
off
heads
the
take
it
what
sense
wilt
they
must
that
feel
me
they
am
able
known
pretty
piece
flesh
well
art
fish
hadst
been
poor
john
draw
thy
tool
here
comes
two
montagues

Figure 25: Output results of normalization of sample text from *Figure 23*.

Conversion to lowercase reduces the number of distinct terms. For example, after normalization the following terms are now considered equivalent :

'Tis and *'tis*
A and *a*
To and *to*
Therefore and *therefore*
That and *that*
The and *the*
Take and *take*
Draw and *draw*
Me and *me*
They and *they*

3.6.3. Transformation

Because the current implementation of the Inter-Stem system uses the Porter Stemmer, which does not have the ability to handle words that contain any kind of punctuation, the tokens, *we'll, run'st, montague's, maiden-heads* require transformation. The purpose of the transformation is to remove the punctuation with the goal of adding the token to the most appropriate equivalence class. The following transformations occur.

Token	***Rule Applied***	***Result***
we'll	`ll→removed	we
run'st	remove apostrophe	runst
montague's	`s→removed	montague
maiden-heads	each component sent to the stemmer	maiden heads
‘tis	remove apostrophe	tis

3.6.4. Stemming

The stemming phase takes the term produced by the transformation phase and produces its stem. The output of this phase is two items, the normalized term and a stem. The list below is the output produced by the stemming phase for our example text. The stem appears in boldface.

we we’ll
runst run’st
maiden maiden-heads
head maiden-heads
maiden-head maiden-heads
montagu mantague’s
tis ‘tis
enter enter
sampson sampson
and and
gregori gregory
with with
sword swords
buckler bucklers
of of
the the
capulet capulet
hous house
on on
my my
word word
not not
carri carry
coal coals
no no
for for
then then
we we
should should
be be
collier colliers
i i
mean mean
an an
in in
choler choler
draw draw
ai ay
while while
you you
live live
your your
neck neck
out out
of of
collar collar
strike strike
quickli quickly
be being
move moved
but but
thou thou
art art
to to
a a
dog dog

of of
montagu montague
move moves
me me
move move
i is
to to
stir stir
stand stand
therefor therefore
if if
awai away
that that
shall shall
move move
will will
take take
ani any
man man
or or
maid maid
that that
show shows
thee thee
a a
weak weak
slave slave
weakest weakest
goe goes
true true
therefor therefore
women women
weaker weaker
vessel vessels
ar are
ever ever
thrust thrust
push push
men men
from from
hi his
maid maids
the the
quarrel quarrel
between between
our our
master masters
u us
their their
all all
on one
show show
myself myself
tyrant tyrant
when when
have have
fought fought
civil *civil*
cut cut
off off
head heads
the the
take take
what what
sens sense
wilt wilt
thei they
must must
that that
feel feel
me me
thei they
am am
abl able
known known
pretti pretty
piec piece
flesh flesh
well well
fish fish
hadst hadst
been been
poor poor
john john
draw draw
thy thy
tool tool
here here
come comes
two two
montagu montagues

Figure 26: Output results of the stemming phase applied to the normalized terms in *Figure 25*.

Notice that the way the Porter Stemmer handles the words *is*, *us*, *his* and *one* is not necessarily as expected. Also, the terms *weaker* and *weakest* are unchanged after stemming.

3.6.5. Class Formation

The class formation phase takes the stream of pairs of normalized terms and their stems produced by the stemming phase, and groups terms into classes based on the stem of the normalized term. Below is listed the classes produced for our sample text. The stem, which represents the class appears in boldface.

a a
abl able
ai ay
all all
am am
an an
ani any
ar are
art art
awai away
be be being
been been
between between
buckler bucklers
but but
carri carry
choler choler
coal coals
collar collar
collier colliers
come comes
cruel cruel
cut cut
dog dog
draw draw
enter enter
ever ever
feel feel
fish fish
flesh flesh
for for
fought fought
from from
goe goes
gregori gregory
hadst hadst
have have
head heads maiden-heads
here here
hi his
hous house
i i s
if if
in in
it it
john john
known known
live live
maid maid maids maiden-heads
maid-head maiden-heads
man man
master masters
me me
mean mean
men men
montagu montague montagues montague's
move move moved moves
must must
my my
myself myself
neck neck
no no
not not
of of
off off
on on one
or or

our our
out out
piec piece
poor poor
pretti pretty
push push
quarrel quarrel
quickli quickly
runst run'st
sampson sampson
sens sense
shall shall
should should
show show
slave slave
stand stand
stir stir
strike strike
sword swords
take take
that that
the the
thee thee
thei they
their their
then then
therefor therefore
thou thou
thrust thrust
thy thy
ti ‘tis
to to
tool tool
true true
tyrant tyrant
u us
valiant valiant
vessel vessels
wall wall
we we we’ll
weak weak
weaker weaker
weakest weakest
well well
what what
when when
while while
will will
wilt wilt
with with
women women
word word
you you
your your

Even on this short piece of text, a few useful classes have begun to form. For example:

```
maid maid  maids maiden-heads
montagu montague montagues montague's
move move moved moves
```

However, some shortcomings are also apparent. Since the Porter Stemmer does not handle the suffixes *-er* and *-est*, the terms *weak, weaker* and *weakest* end up in different equivalence classes.

3.6.6. Class Pruning

The class pruning phase is where a decision is made whether or not to eliminate some of the classes. Classes can be eliminated if they are not considered useful for query expansion. Classes that are candidates for pruning are *and, an, be i, of, or, the, to,* and *with.*

3.6.7. Structure Class

In order to speed up access to the equivalence classes they are placed in a text database. Before placing the classes in the database the classes should be annotated with markup tags as described in section 3.5.3. Because of the small size of the equivalence classes in this example, ordering of the classes is not necessary.

Chapter 4

4. Query Expansion Using The Equivalence Classes

For large document collections, the equivalence classes produced have the potential to become quite large (see example in Appendix C). Selection of appropriate terms from such large classes can be a difficult task. Ideally a facility should be provided where a user can choose the level of expansion (using discrete measures such as low, medium, high or by a sliding scale) and be presented with an appropriate list based on their expansion needs. In order for such a feature to be possible, an appropriate ordering scheme must be developed.

4.1. Class Ordering

Ordering of classes is also an important basis for developing tools that allow the user to specify the degree of expansion required. Classes should be ordered in a way that will aid the user in selecting appropriate terms for expansion. If the selection of terms is a difficult process because of the size and organization of the classes, query based stemming will not be practical.

A variety of methods can be used to structure the equivalence classes in a manner that facilitates term selection. Classes can be fully ordered, either statically at class formation time or dynamically based on query context. In many cases the information that will be used in determining class order can be computed and stored at index time. Nevertheless, class ordering is best performed dynamically at query time because the optimal ordering of terms is usually dependent on the query term and context. With a fully ordered list the user has the option of selecting individual terms or selecting a cut off point, where all terms below the cut off point are discarded. Another option is to partition the class based on a classification scheme and organize the discrete groups into a partial order. The order of remaining terms could then be changed based on initial selections from the partial order. This process could be repeated in a drill down manner, resulting in hierarchical clusters.

4.1.1. Lexical Distance

One method for dynamically creating a total class order is by lexical distance from the query term. There are a variety of formulas that could be used for such a calculation. A naïve approach is to count the "edit distance" [HD80] between the candidate term and the query term. Transformation calculations can be limited to the addition of characters, or may include complex interchanges and removal of characters. For example, Figure 27 shows the ordering of a sample class based on lexical distance.

Term	***Transform***
run	none (query term)
runs	addition of *-s*
runner	addition of *-ner*
running	addition of *-ning*

Figure 27: Ordering of sample class *run* based on lexical distance.

However, determining order solely by the number of transformations performed can be problematic when words that are lexically close are semantically distant. Continuing with the *run* example above, the word *rune* is lexically very close to *run*, but the word *running* is semantically closer. Similarly the following pairs of terms represent different concepts despite being syntactically similar :

past	paste	head	heading
on	one	attache	attached
arm	army	suite	suited

A similar problem with lexical distance calculations occurs for query terms that represent multiple concepts. For example, Figure 28 represents the ordering for a sample class based on the ambiguous query term *orient*. The ordering of the *orient* class by lexical distance proved not to be helpful. This is because the term *orient* has multiple semantic meanings. The addition of *-er* or *-al* require comparable transformation, but each suffix is associated with a different sense of the term *orient*. As a result, in such cases, the lexical distance measure produces an interleaving of the various senses of the query term. However, Figure 29 illustrates that the effectiveness of ordering the *orient* class by lexical distance is greatly improved when the query term is *oriental*.

The problems with polysemous terms, such as *orient*, may be reduced by the use of partially ordered lists. A partial order involves presenting the user with a short list of terms, and then based on initial selections, ordering the rest of the list. For example, if the query term is *orient* and the user selects *oriental* from the initial partial order, the variants of *oriental* will rise to the top of the order and the impact of interleaving will be greatly reduced.

Term	***Transform***
orient	none (query term)
orients	addition of *-s*
oriental	addition of *-al*
oriented	addition of *-ed*
orientals	addition of *-als*
orienteer	addition of *-eer*
orienteers	addition of *-eers*

Figure 28: Ordering of sample class *orient* based on lexical distance.

Term	***Transform***
oriental	none (query term)
orientals	addition of *-s*
orient	removal of *-al*
orients	removal of *-al* addition of *-s*
oriented	removal of *-al* addition of *-ed*
orienteer	removal of *-al* addition of *-eer*
orienteers	removal of *-al* addition of *-eers*

Figure 29: Ordering of sample class oriental based on lexical distance.

The computation of lexical distance can further be refined by considering transformations based on suffixes rather than just individual characters. Transformations for plurals, possessives, and past tense (e.g. *-s, -ed, -ing*) should be considered low cost operations. Since derivational variations (e.g. the addition of *-ize* and *-ship* as in general→generalize, court→courtship) usually have a greater effect on semantics than inflectional variation, derivational transformations should have a greater cost. A list of suffixes, such as the one in Appendix D, could be incorporated into a lookup table where each entry includes an associated cost.

As a further refinement, the removal and append operations may assigned different costs. For example, the following two transformations on the word communicate require the same number of character operations.

communicate → communicational	remove *-e* add *-ional*	*6* character operations
communicate → commune	remove *-icate* add *-e*	*6* character operations

One transformation requires more character removals, while the other requires more character additions. In this example, the character removal operations resulted in a greater semantic change than the character addition operations.

In summary, the development of a lexical distance algorithm is a non-trivial activity and needs to be sensitive to the peculiarities of the language.

4.1.2. gram

Another method that could be used for creating a total order of the equivalence class is the n-gram method. The digram (n=2) method is a measure used to compute the similarity of terms based on unique pairs of consecutive letters held in common [AB74]. The technique

is not limited to digrams and can be implemented for small values of n, thus the name n-gram.

Once the unique digrams for the word pair have been identified, a similarity measure is computed as follows:

$$S = 2C(A+B)$$

where :

A = the number of unique digrams in the first word

B = the number of unique digrams in the second word

C = the number of unique digrams shared by *A* and *B*

The following example uses digrams to compute the similarity between *statistical* and *statistically*.

statistical → st ta at ti is st ti ic ca al

unique digrams = st ta at ti is ic ca al

10 digrams, 8 unique

statistical → st ta at ti is st ti ic ca al ll ly

unique digrams = st ta at ti is ic ca al ll ly

12 digrams, 10 unique

statistical and *statistically* share 8 unique digrams

In the above example, the similarity measure for *statistical* and *statistically* is :

$$S = 2 * 8 / (8+10) = .89$$

The n-gram measure can be used to compute a similarity measure between the query term and each term in an equivalence class. The class could then be presented totally ordered by decreasing similarity.

An alternative is to present the user with a partial list consisting of the top 5-7 terms based on the similarity measure. The rest of the terms could then be ordered based on similarity to the selections from this initial list.

The n-gram method can also be used to create clusters of terms by creating a similarity matrix for all terms in the class and applying a single link clustering algorithm [Ras92].

4.1.3. Term Co-occurrence

The term co-occurrence measure is based on the assumption that words that should be conflated are likely to occur in the same documents or text windows. Croft and Xu suggest that a variation of the EMIM measure [Van79] may be used to determine the significance of word form co-occurrence [CX95].

$$em(a,b) = \frac{o(a,b)}{n(a) + n(b)}$$

where

o(a,b) is the number of times both *a* and *b* occur in a text window of predetermined size (e.g. $\{<a_i b_j> \mid distance\ (a_i,b_j) < window\ size\}$ where *window size* may equal some unit such as a document)

n(a) is the number of times *a* occurs in the corpus

n(b) is the number of times *b* occurs in the corpus

This measure can be repeatedly calculated with *a* equal to the query term and *b* equal to each member of the equivalence class. The class can be linearly presented by decreasing *em* value, or as a complex hierarchy based on the results of computing *em* values for each pair in the class.

4.1.4. Term Importance

A static method for determining term order of an equivalence class is to base the order on term importance. The *inverse document frequency*, *noise* and *term discrimination* are all measures that attempts to identify term importance. These are static measures, in that they are independent of query context. A class can thus be linearly ordered by decreasing term importance prior to query time.

Measures of importance based on term frequency and distribution would not be possible if terms tended to occur randomly and with consistent frequency across a document collection. Using term frequency to determine term significance is based on the theory that an author will tend to repeat certain words as they develop their ideas. Thus, repetition of a word is an indication of emphasis and significance, up to some threshold. High frequency terms are usually low discriminating stop words (e.g. *the, of, and to, a* etc.).

According to Zipf's law, when the terms in a body of text are arranged by decreasing order of frequency of occurrence, the following holds [Zipf45]:

frequency * *position in list* ≅ constant

Many automatic indexing techniques will ignore terms with frequency of occurrence above a certain threshold. Similarly, very low frequency words may occur so infrequently in the collection that their presence may not be considered to affect performance in a significant way. Terms below a low frequency threshold are often discarded by automatic indexing programs. However, eagerness to discard terms should be tempered with the knowledge that the elimination of high-frequency words can result in loss of recall and the elimination of low-frequency terms may negatively effect precision. The use of absolute measures of frequency for measuring term importance is considered naïve and more sophisticated measures exist [Sal83].

There are two main characteristics of a good discriminating term. First, the term must be representative of the information content of the document. Second, the term should help to distinguish the document from other documents. A measure of term importance needs to compute relative frequency to identify terms that occur with significant frequency in some documents, but with relatively low frequency in the collection as a whole.

The common measures for term importance are *inverse document frequency*, *noise* and *term discrimination*. The inverse document frequency measures term importance as proportional to frequency of occurrence in a document and inversely proportional to the number of documents that the term occurs in. The noise measure identifies broad non-specific terms, that are evenly distributed across the document collection. The signal measure is an inverse function of noise and is used to identify the importance/content value of a term. Ordering the terms within the equivalence classes by decreasing signal value has the effect of favoring terms that distinguish one or two specific documents. The term discrimination value attempts to measure the degree to which a term will help to distinguish documents from one another. Ordering the terms within the equivalence classes by decreasing discriminating value has the effect of favoring terms that discriminate the document they occur in from other documents in the collection.

4.1.5. Concept Clusters

Since actual word meanings have a strong correlation with relevance judgments, it would be desirable to form concept clusters within each class. This approach views the classes as discrete conceptual groups, rather than as a linear list of ordered terms. A representative term from each cluster can be presented to the user for selection. Based on their selection, the appropriate concept cluster is used for query expansion.

Since a word may have multiple senses, a term must be able to be a member of more than one concept cluster. The term *gravity* is an example of a word with multiple senses. One sense of *gravity* (great importance or significance; seriousness) dictates it should be clustered with *grave*. Another sense of *gravity* (tendency of objects to fall towards the

center of the earth) necessitates it appear in the same cluster as *gravitation*. The clustering algorithm should allow a term to appear in both groups.

The determination of word senses is a major problem both practically and theoretically. Krovetz developed modifications to the Porter algorithm that incorporate a machine readable dictionary to provide word-sense disambiguation [Krov93]. He has also developed inflectional and derivational stemmers that attempt to incorporate word-sense disambiguation. Unfortunately, there is no one piece of information in current on-line dictionaries or thesauri that can be used to disambiguate terms consistently. Krovetz, used a combination of evidence from part of speech, subcategorization, subject area codes etc. The process of analyzing evidence from multiple sources and handling exceptions is a complicated heuristic approach.

In addition, no dictionary is exhaustive and many of the document terms may not appear. Technical, domain specific terms, proper nouns, acronyms, abbreviations and hyphens are important when working with real world text, but are often not included in a dictionary. Any attempt to use a dictionary for disambiguation will therefore be incomplete.

Another method that may be applicable to the formation of concept clusters is *lexical chaining*. A lexical chain is a succession of semantically related words in a text that creates a context and contributes to the continuity of meaning [MH91]. Since word-sense disambiguation is an important component of lexical chaining, it is conceivable that it could be applied to identifying concept clusters within equivalence classes. St. Onge has developed a software tool that reads in documents and automatically groups words together (lexical chains), based on relationships found in WordNet 1.4[5] [Sto95]. If equivalence classes instead of documents were provided as input, the chains produced could be used as the basis for concept clusters.

Research into word sense disambiguation techniques and their retrieval effectiveness has recently become popular [Gre92][San94][Voo93][Voo94]. Results indicate that more

[5] WordNet is a trademark of Princeton University.

basic research is needed into the relationship between sense ambiguity, disambiguation and retrieval performance. Once mature, disambiguation techniques represent an appropriate way for clustering terms within an equivalence class.

4.2. Expansion Methods

The purpose of building equivalence classes of words is to use them in query expansion. When a term is to be expanded, the Inter-Stem tool will stem the term and retrieve the equivalence class associated with the resulting stem. Either a portion or the complete class can then be added to the query. The three main ways that query based expansion can be implemented is command-line utility, batch process or interactive session.

4.2.1. Command Line

The command-line version of the Inter-Stem system is a useful query formation aid. It allows a user to type :

```
prompt>expand my_word
```

where upon a list of the morphological variants of *my_word* from the equivalence classes is returned. The command line version is designed to be used as a separate utility that can aid in determining terms to be included in a query. The current version has a switch for choosing between low, medium and high levels of query expansion.

```
prompt>expand flood
flood floods flooded flooding
```

Figure 30: Command-line expansion of term *flood* using the default *low* setting.

```
prompt>expand H flood

flood floods flooded flooding flood's flooding's anti-flood
asia-floods bangladesh-flood bangladesh-floods bog-flooding
china-flood ethiopia-floods flash-flood flash-flooding
flood-battered flood-causing flood-contaminated flood-
control flood-coroner flood-created flood-damaged flood-fill
flood-filling flood-flow flood-free flood-hazard flood-hit
flood-induced flood-insurance flood-irrigate flood-irrigated
flood-lighted flood-like flood-lit flood-loss flood-of
flood-plagued flood-plains flood-polluted flood-producing
flood-prone flood-protection flood-ravaged flood-related
flood-relief flood-scene flood-search flood-stage flood-
stranded flood-stricken flood-struck flood-survivor flood-
swept flood-swollen flood-threatened flood-tide flood-torn
flood-warning flood-water flood-weary flooded-out flooded-
the-carburetor floods-history floods-missing floods-
vignettes fluorescent-flooded half-flooded hard-disk-
flooding hurricane-flooding light-flooded near-flood non-
flood non-flooded non-flooding now-flooded once-flooded
post-flood rain-flooded room-flooding seasonally-flooded
soviet-floods steam-flooding sun-flooded
```

Figure 31: Command-line expansion of term *flood* using the *high* setting.

4.2.2. Batch

Query based stemming can also be performed as a batch process. A file containing the text of queries (written in a predefined query language) with the terms to be expanded marked, is submitted to the Inter-Stem batch facility for expansion. The batch facility will return a text file with the queries modified to contain all the morphological variants of the marked terms. The results will normally be submitted directly to the retrieval engine, but the user may choose to edit the results before submission.

The batch mode has the advantage that it can include understanding of the query language and make expansions that are consistent with the syntax of the language being used. Options such as expansion level and handling of hyphens can be specified by parameters and will be applied to all expansions uniformly.

Since the results of the batch process will normally be submitted directly to the retrieval engine, more detailed control than the course grained low, medium and high levels of expansion provided by the command line version are needed. The choice of levels of expansions need to be detailed and range from very low expansion, such as plurals only, to expansion that includes all terms in the equivalence class.

Figure 32 is a sample query written in the query language GCL (overview of syntax appears in Appendix F), ready for submission to the Inter-Stem batch process. Terms to be expanded are delimited by dollar signs. Figure 33 is the expanded version of the query produced by the Inter-Stem system.

```
FEMA = “fema”+((“federal”<>“emergency”<>“management” <>”agency”)
            <[4])
DISASTER = $disaster$ + $flood$ + $earthquake$ + $hurricane$ +
            $tornado$
QUERY = FEMA^DISASTER
@rank QUERY
```

Figure 32: Query marked for batch processing.

```
FEMA = “fema”+((“federal”<>“emergency”<>“management” <>”agency”)
            <[4])
DISASTER0 = “disaster” + “disasters”
FLOOD0 = “flood” + “floods”+ “flooded” + “flooding”
EARTHQUAKE0 = “earthquake” +“earthquakes” + “earthquaking”
HURICANE0 = “hurricane” + “hurricanes”
TORNADO0 = “tornado” + “tornadoed” + “tornadoes”
DISASTER = DISASTER0 + FLOOD0 + EARTHQUAKE0 + HURRICANE0 +
            TORNADO0
QUERY = FEMA^DISASTER
@rank QUERY
```

Figure 33: Results of batch processing applied to Figure 32.

4.2.3. Interactive

Thirdly, and ideally, query based stemming should be integrated into a query interface for interactive query formulation. Such a facility would provide the user with even greater control over the expansion process. The interactive method allows for expansion of terms to be handled *individually*, unlike the batch process where all terms and queries are handled uniformly.

In addition to customizing the degree of expansion on a term by term basis, the user can choose to have the class presented as either a total order, cluster of terms, or as a hierarchy. The interactive method allows the presentation or suppression of additional information such as frequency of occurrence and term importance. Too much information can be overwhelming, but the option to display extra information or re-order the equivalence classes presents flexibility not available with other methods.

A feature that could be very useful is the ability to view and temporarily merge equivalence classes that have lexically similar stems. This would provide the user with a mechanism to add terms that do not appear in the same class due to inadequacies of the underlying stemming algorithm. For example, it would be useful to be able to merge and then order the *deter* and *deterr* classes. Term selection would then be performed on the merged class.

deter deter deters deterring

deterr deterrence deterrent deterrents

Features that would be helpful in an interactive stemming system are facilities for the iterative refinement of the query involving the perusal of preliminary results, manual editing, refinement and re-submission. One of the aims of an interactive stemming interface is to make the query formulation process more iterative. Different types of information and feedback could be used at different stages of the formulation process. In the early stages users may not fully understand how to express their information need or be familiar with the terminology used in the document collection. Interactive query expansion requires a term

selection stage where the system presents the query expansion terms to the user in some reasonable order, preferably with the terms that are most likely to be useful appearing first.

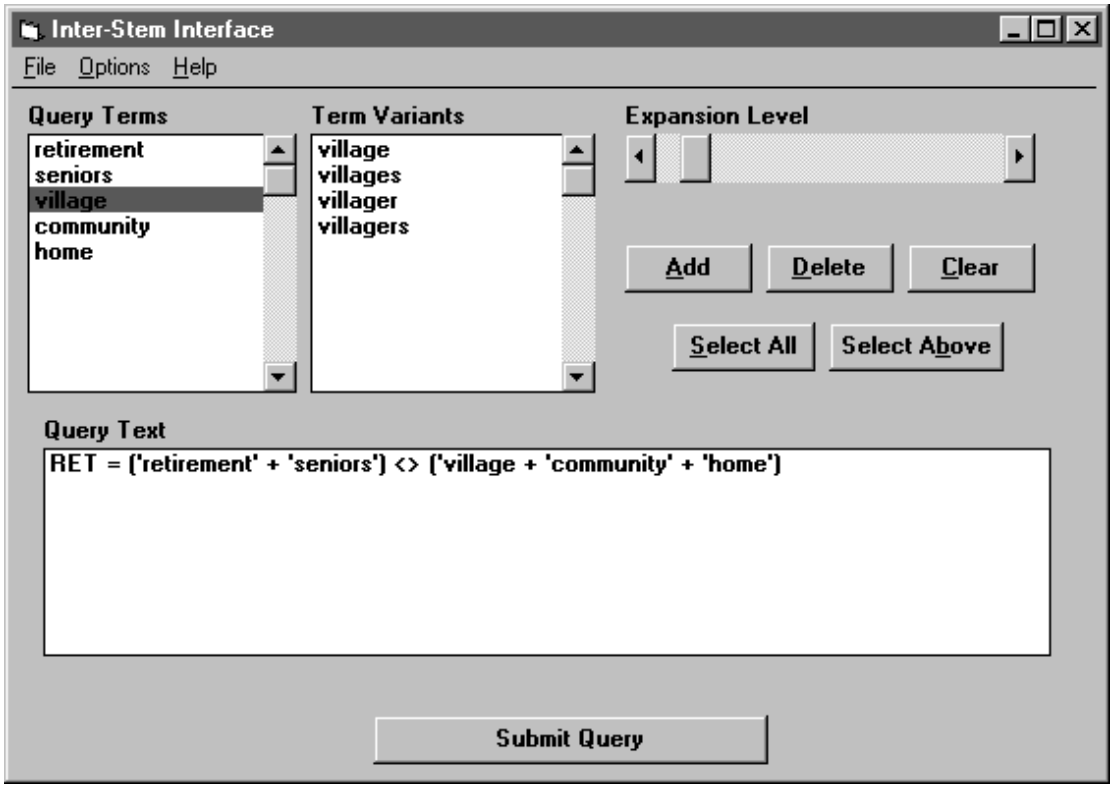


Figure 34: Basic interface for query based stemming.

Figure 34 is a preliminary design for a very basic interactive stemming interface. The query terms box is automatically populated based on the text of the query. When a user selects a query term (in the example *village* is selected), a list of morphological variants appears in the term variants list box. A slider is provided for increasing and decreasing the expansion level. Various push buttons are provided for term selection. When the submit button is pressed the query is run, and the total number of hits and the titles of the top ten documents are displayed in a scrollable window. The scroll bar allows the user to scroll

through all the documents returned. The full text of a document can be viewed by double clicking on a document title.

Query based stemming could be combined with relevance feedback. Relevance feedback is an interactive retrieval tool that is often used to retrieve more documents by expanding the number of terms in a query. Systems that implement relevance feedback bring back an initial set of documents that are presented to the user for relevance judgments. The original query is then expanded with terms from the relevant documents. These terms could act as a filter on the morphological variants of the original query terms (i.e., only variants that occur in the relevant documents are used for expansion). In experiments by Harman, filtering of variants based on feedback documents was shown to produce performance improvements comparable to relevance feedback [Har88].

Chapter 5

5. Evaluation of Query Based Stemming

In order to evaluate the effectiveness of user mediated, query based stemming a suitable test environment is needed. The test environment should include queries, a collection of documents and relevance judgments.

Since its introduction, the Text REtrieval Conference (TREC) has provided a large heterogeneous test collection, which has often been used to test various query expansion methods. Early TREC results showed some support for the use of query expansion techniques. Whether these results are applicable to on-line retrieval systems is questionable due to the detailed and lengthy user need statements that were used for query formulation. In Trec-4 the user need statements have been shortened, making it an ideal test environment for query expansion techniques. For this reason the TREC-4 collection was chosen for our query based stemming experiments.

5.1. Trec

TREC is an annual conference with the goal of bringing research groups together to discuss information retrieval research involving large document collections. TREC is co-sponsored by the Advanced Research Project Agency (ARPA) and the National Institute of Standards and Technology (NIST). The motivation for introduction of TREC is to provide a very large test collection to stimulate interest in information retrieval research and encourage interaction between academia, industry and government. The annual TREC conference provides a forum where research groups can compare results from experiments against a common document and topic collection using standard evaluation methodology.

5.1.1. Overview

Information retrieval has a long history of experimentation. Unfortunately, individual research groups often perform experiments using their own sets of data and without common evaluation methods. This makes it difficult to compare the effectiveness of different techniques. The Cranfield experiments in the mid sixties introduced the importance of creating common test collections [CMK66] (cited in [Har95]). The popularity of the Cranfield collection led to the introduction of other similar collections, such as the CACM and NPL collections. However, many of the experiments performed against these collections do not use the same evaluation techniques, and there is very little comparison of results across systems or collections. In addition, the size of these test collections are not realistic representations of current *real-world* information retrieval environments. Techniques that have been shown to perform well on small test collections do not necessarily scale up to larger corpora. Other techniques such as automatic thesauri are impractical and produce poor results when tested against small collections, but perform quite well when used with larger collections.

5.1.2. The Task

There are two kinds of tasks in TREC, *adhoc* and *routing*. For each task a series of information needs are defined in natural language. These are the basis for query construction. The two main tasks are subdivided into three categories based on query construction technique.

Automatic

The system automatically analyzes the information need and formulates a query based on this analysis. No manual modifications are allowed.

Manual

Queries are manually constructed (software tools are allowed for assistance) based on the defined information need. Once the query is constructed *no* modifications are allowed.

Interactive

An initial query is constructed by either manual or automatic means. The query is run against the document collection (or subset), and based on the content of the documents retrieved the query may be modified. This iterative process many be repeated as many times as needed.

In the adhoc query task, the collection to be queried is both known and static. The information need statements are translated by each team into queries to be entered into their system and run against the collection. The retrieval results are submitted for evaluation.

In the routing task, the information need is static, but the collection to be queried is dynamic. Participants are given a set of query topics and a set of training documents with relevance judgments. As in the adhoc task, the topics are used to formulate queries. These queries are run against the training documents and modified/tuned for optimal results. The final tuned version of the queries are then run against a new set of documents, and the results are submitted for evaluation. The routing task is meant to simulate information filtering such as would be done by a personal news filtering agent/service.

The results for each task are submitted and run through a common evaluation package. The use of a common evaluation routine allows participants to compare the effectiveness of different retrieval and query formulation techniques.

5.1.3. The TREC Test Collection

The TREC test collection consists of documents, definitions of information needs and relevance judgments (lists of document that satisfy the defined information needs). The document collection for TREC is distributed on a set of CD-ROMS, each of which contain about 1 Gigabyte of data that has been compressed. The contents of the collection is changed for each conference. For TREC-4, the test document collection included:

Training Data for the Routing Task

Disk 1

Federal Register (1994)
Information from Computer Select Disks
IR-Digest (archives)
Virtual Worlds
Selections from Usenet Newsgroups

Data for the Adhoc task

Disk 2

Wall Street Journal (WSJ) 1990 - 1992
Associated Press (AP) Newswire 1988
Federal Register 1988
Information from Computer Select disks

Disk 3

San Jose Mercury News 1991
AP Newswire 1990
U.S. Patents 1993
Information from Computer Select disks

The source of documents are chosen with the aim of producing a heterogeneous retrieval environment. The collection contains documents of varying lengths, writing style, and editing (spelling and typographical errors are left in the source) as well as various vocabularies/domains.

Figure 35 illustrates the diversity of the TREC document collection. The U.S. patents and IR digest contain predominately long documents, while the Computer Select collection consists of mostly short articles. The lengths of documents in the Federal Register vary widely (1,398 average vs. 403 median). For contrast, entries have been appended to the table for some of the older collections. Notice that most of the older test collections are considerably smaller than even one of the components of the TREC collection.

Collection	*Size in Megabytes*	*Average Terms Per Document*	*Median Terms Per Document*	*Total Records*
Federal Register (1994)	283	456	390	55,554
IR Digest	7	2,383	2,225	455
News Groups	237	340	235	102,598
Virtual Worlds	28	416	225	10,152
AP Newswire (1988, 1990)	482	472	443	158,240
Federal Register (1988)	211	1,398	403	19,860
Computer Select (Disks 2 & 3 combined)	552	315	150	75,180
San Jose Mercury News (1991)	290	285	227	90,257
U.S. Patents (1993)	245	4,777	3,922	6,711
Wall Street Journal (1990 - 1992)	247	≈377	≈218	≈74,520
TREC Total	2582			593,527
Cranfield	1.5	88	79	14,000
CACM		24.26		3,204
MED		54.69		1,033
NPL		19.96		11,429

Figure 35: [6]Comparison of Test Collections[Har93][Har96].

Each TREC document is organized in a standard format using SGML style tags. Each data file may contain multiple documents. The following is a sample document from the San Jose Mercury News portion of the collection.

[6] Terms were computed using no stopwords, stemming or tokens beginning with a number.

```
<DOC>
<DOCNO>SJMN91-06002002</DOCNO>
<ACCESS>06002002</ACCESS>
<CAPTION>Drawing; DRAWING: Dan Hubig</CAPTION>
<DESCRIPT>OPINION; NEWSPAPER; WRITING</DESCRIPT>
<SECTION>Editorial</SECTION>
<HEADLINE>THE NEW YEAR IN TABLOID HEADLINES</HEADLINE>
<MEMO>Commentary
Tony Kornheiser writes for the Washington Post.  </MEMO>
<TEXT> (Let me digress. I had the buona fortuna of driving home
in Thursday's snow behind a procession of drivers suffering snow
paralysis. There's less than 1 inch on the roads, and they're
going so slowly, I think I'm in an Ingmar Bergman movie. What,
none of these bozos has ever seen snow before? Just my luck to
get on the road as the new "Only Drivers From the Tropics Allowed
6-8 p.m." rule goes into effect. Six miles, 65 minutes. I'd have
gladly turned into Rock Creek Park and risked skidding headfirst
into a tree for the sheer thrill of actually moving. It's snow,
people, not nuclear winter. If you're behind the wheel of a car,
it's an indication that you intend to go somewhere.)
…
</TEXT>
</DOC>
```

Figure 36: Sample TREC document.

5.1.4. Query Topics

Instead of providing a query set (e.g., in the form of boolean expressions), the TREC collection contains a set of query topics that are presented as user need statements. This allows for a variety of query construction methods to be used.

The topics are designed to imitate a real user's information requirement. In earlier TRECs the information need statements were long, detailed and highly structured. In TREC-4 a change was made to short concise topics that better reflect how *real* users would state their need for information. Keeping the topics concise encourages the development of techniques for expanding statements of information requirements that are *too short*.

The query topics are developed by the same group of people who are responsible for determining document relevancy. Each assessor develops 10 query topics. These topics are then tested against a subset of the document collection. The final selection of topics is made based on the projected number of relevant documents, clarity, and contribution to a mix of broad and narrow topics. Each topic is formatted using a standard method, as illustrated below.

Samples of Adhoc Query Topics

```
<top>
<num> Number:  218
<desc> Description:
How have mini steel mills changed the steel industry?
</top>

<top>
<num> Number:  223
<desc> Description:
What was responsible for the great emergence of
"MICROSOFT" in the computer industry?
</top>

<top>
<num> Number:  231
<desc> Description:
Should the U.S. Government provide increased support
to the National Endowment for the Arts?
</top>
```

5.1.5. Relevance

Relevance judgments are a very important part of a test collection. For each query a comprehensive list of relevant documents in the collection must be provided.

There are several possible methods for compiling such a list. One method is to compile full relevance judgments for each topic across all documents. In the case of the

TREC-4 adhoc task, providing judgments on 50 topics across approximately 593,527 documents would require in excess of 29 million relevance judgments. Clearly this would be a very time consuming and expensive process. Another option is to take a random sample of documents, and perform relevance judgments only on the sample. The size of the sample needed to find a suitable number of documents per topic would be very large. A third option is the pooling method, which is based on providing relevance judgments for a set of the documents retrieved using the various participating systems. This has been used successfully for TREC and for other collections (NPL and INSPEC).

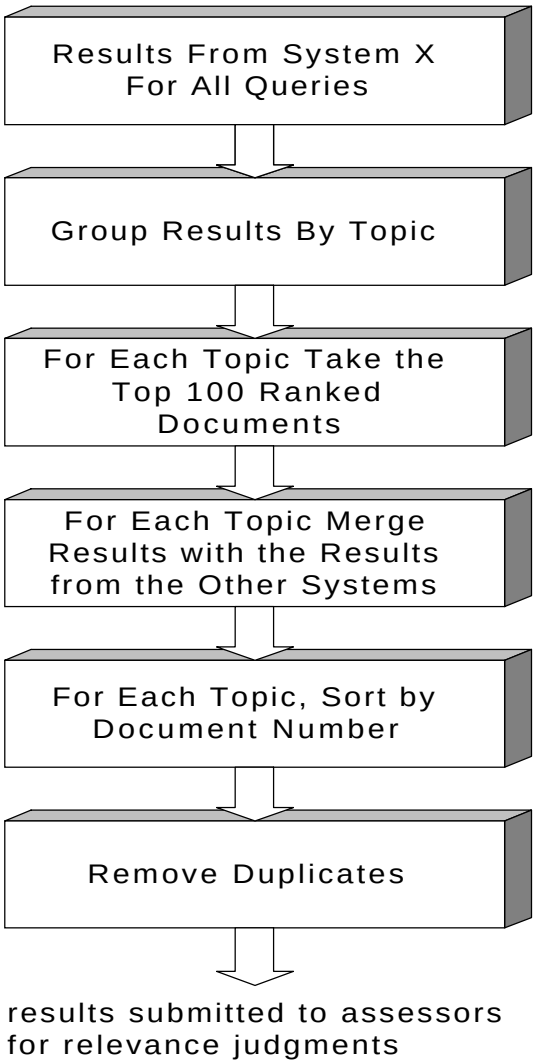


Figure 37: TREC Pooling Methodology

Figure 37 illustrates the pooling methodology used by TREC. The results produced by the each participating system are submitted for evaluation. These results are then divided into query topics. For each topic the top 100 ranked documents (out of the possible 1000 submitted) are selected. The results by query from all the other systems are merged. The merged results are sorted by document number within topic and duplicates are removed. The resulting list of documents by topic are submitted to the assessors for relevance judgments. Results are judged by the assessor who created the topic, as well as two additional assessors.

In TREC-4 there were 74 adhoc runs. Each query can potentially return 1,000 documents, resulting in a potential of 7,400 documents for each query. Once duplicates are removed, on average there are approximately 1,770 documents produced by the pooling method. Of these, approximately 7.5% were judged relevant.

5.1.6. Evaluation

One of the goals of TREC is to provide a common evaluation forum. Recall/precision figures are calculated for each system and a standard set of results for each system is published in the proceedings. The standard results for Waterloo MultiText's submission to Trec-4 is provided as a sample in Appendix E.

5.2. MultiText Retrieval Engine

The current version of the Inter-Stem system is designed to work with the retrieval engine of the MultiText Project at the University of Waterloo. A summary of the basic syntax of GCL, the query language used by this system, can be found in Appendix F.

GCL determines matches based on the shortest substring model. The text in the database is viewed as a continuous sequence of terms and tokens. A boolean subset of the language is used for the TREC queries. In a traditional boolean system, the query

```
"information" AND "retrieval"
```

would retrieve all *documents* that contain both terms. In GCL, the result is the smallest interval of text that contains both query terms.

The concept of shortest substring satisfying the query Q, can be more formally defined as an interval represented by the pair (α,Ω) of integral positions in the text, such that the substring beginning at α and ending at Ω satisfies the query and no containing interval does so [CCB95c]. For the basic boolean operations:

1. An interval (α,Ω) is a solution to the query Q_1 **AND** Q_2, if the interval satisfies Q_1 *and* the interval satisfies Q_2.
2. An interval (α,Ω) is a solution to the query Q_1 **OR** Q_2, if the interval satisfies Q_1 *or* the interval satisfies Q_2.
3. An interval (α,Ω) is a solution for Term ***T*** if the interval contains the term *T*.

Based on these definitions, a large number of intervals may satisfy a query. For example, the query *"to" AND "be"* performed against the text in Figure 38 can be satisfied by the intervals (1,2) (2,5) (5,6) (1,6) (1,5) (1,4) (1,3) (2,6) (3,6) (4,6). Although, many intervals satisfy the query, only intervals that do not contain within them a satisfying interval are viewed as solutions.

```
To be or not to be
1  2  3  4   5  6
```

Figure 38: Example text database

The ranking method used by the MultiText retrieval system takes advantages of the shortest substring model of GCL. The ranking algorithm is based on two key assumptions:

1. The smaller the solution interval of text, the greater the probability that the text is relevant.
2. The more solution intervals contained in a document, the greater the probability that the document is relevant.

Based on these two assumptions, the rank of a document that contains intervals $(\alpha_1,\Omega_1),(\alpha_2,\Omega_2), \ldots,(\alpha_n,\Omega_n)$ is [CCB95c]:

$$DOCUMENT\ SCORE = \sum_{i=0}^{N} S(\alpha_i, \Omega_i)$$

where

$$S(\alpha_i, \Omega_i) = \begin{cases} \dfrac{A}{(\alpha_i, \Omega_i)} & \text{if } |(\alpha_i, \Omega_i)| \geq A \\ 1 & \text{if } |(\alpha_i, \Omega_i)| \leq A \end{cases}$$

The constant *A* represents the threshold, below which, assumption one no longer holds. Based on earlier TREC experience, *A* is assigned a value of 16 [CCB95c].

The document score is the sum of the interval scores, where the shorter the interval the higher the score. A higher score represents greater probability that the document is relevant. Documents are returned ranked by decreasing score.

5.3. Equivalence Class Construction for the TREC Collection

The TREC ad hoc collection was used to build the equivalence classes for our query based stemming experiments. The ad hoc collection consisted of approximately 2 Gb of data across 2,255 text files. Figure 39 lists some of the statistics compiled during class construction. Additional statistics are included in Appendix G. This data indicates that many of the equivalence classes are large in size.

Collection Stats	
Unique words	808,273
Unique words containing apostrophes	66,743
Unique words containing hyphens	267,254
Words that occur only once in the collection	367,038
Equivalence Class Stats	
Equivalence classes	630,267
Size of largest class	5,699
Size of smallest class	1
Classes of size 1	517,847
Classes of size between 1 and 100 terms	111,207
Classes of size greater than 100 terms	1,213

Figure 39: Class construction statistics.

The largest class is represented by the stem *non* and includes 5,699 distinct terms. To illustrate how this class became so large, a sample of terms from the class is included in Figure 40. The use of *non* as a prefix in hyphenated words and the inclusion of these hyphenated words in the equivalence classes accounts for the extremely large class size. Frequent use of hyphenation and the resulting effect on class size had not been anticipated. A selection of other large equivalence classes were examined to see if the inclusion of hyphenated words inflated class sizes (see Figure 41 for some examples). Examination of large classes showed that over 90% of the terms were hyphenated.

non non nons non' alumina-non anti-non-a aromatic-non-aromatic chag-non co-non-polynomial context-non-local ethylene-propylene-non-conjugated i-non-competitive low-to-non-irritating metal-non-metal microorganism-non-decomposable my-non-deterministic-call no-non-sense non-a non-abandonment non-abbreviated non-ablating non-ablative non-aborigines non-abortion non-abraded non-abrading non-abrasive non-abrupt non-absolute non-absorbable non-absorbent non-absorbing non-absorptive non-abstract non-abused non-abusers non-abusive non-academic non-academicians non-academics non-academy non-accelerated non-accented non-acceptability non-acceptable non-acceptance non-access non-accessible non-accident non-accidental non-accommodating non-accomplishment non-account non-accountability non-accountable non-accountant non-accountants non-accounting non-accounts non-accredited non-accruing non-accumulating non-accusatory non-achromatic non-acid non-acidic non-acoustic non-acquiescence non-acquisition non-acquisitiveness non-acrylic non-act non-acting non-action non-actionable non-actions non-activated non-active non-active-matrix non-active-multiplex-matrix non-activity non-activity-induced non-actor non-actor-proof non-actors non-actors' non-actress non-actuated non-acute non-acutely non-adaptive non-addicting non-addictive non-address non-addressable non-addressed non-adhered non-adherence non-adherent non-adherents non-adhering non-adhesive non-adidas non-adjacent non-adjoining non-adjustable non-adjusted non-adjustment non-administrative non-admission non-admissions non-admittance non-adobe non-adobe-licensed non-adsorbed

Figure 40: Selections from the *non* equivalence class.

Sample of terms from the *driven* equivalence class

driven driven driven' acquisition-driven action-driven activity-driven advertiser-driven advertising-driven agenda-driven air-driven alcohol-driven anxiety-driven application-driven applications-driven applicaton-driven attribute-driven auger-driven automatic-driven axially-driven backseat-driven' banjo-driven batch-driven batch-file-driven battery-driven belt-driven bicycle-driven bottom-line-driven brand-name-driven budget-driven bureaucrat-driven business-driven button-driven cache-driven capital-driven career-driven case-driven cash-driven cast-driven casual-yet-driven celebrity-driven channel-driven character-and-plot-driven character-driven chauffeur-driven

Sample of terms from the *book* equivalence class

book book booked booking bookings books book' book's booking' books' books's ad-booking address-book address-book-size advance-booking advanced-booking already-booked alternative-book appointment-book asset-to-book-value back-of-the-book band-booking bank-to-book bankruptcy-book baseball-book big-book black-book blank-book blue-book book-assembly book-author book-autographing book-balancing book-banning book-based book-binding book-bindings book-borrowing book-building book-building' book-burning book-burnings book-burying book-buyers book-buying book-centered book-closure book-complaint book-compliant book-contract book-crammed book-disk book-distribution book-edition book-end book-entry book-entry-only book-excerpts book-expo book-expo's book-filled book-first book-form book-formatting book-in book-in-progress book-industry book-inspired

Sample of terms from the *mail* equivalence class

mail mail mailed mailing mailings mails mail's mailings' advertising-mail air-mail already-mailed audits-by-mail brochure-mailings bulk-mail bulk-mailed bulk-mailing bulk-mails business-mailing by-mail by-mail-order-only cargo-mail chain-mail check-mailing complete-mail corporate-e-mail courier-mail customer-mail data-mailing dedicated-mail-management-software direct-mail direct-mailing discount-mail distribution-by-mail divorce-by-mail drugs-by-mail e-mail's e-mail-based e-mail-enabled e-mail-fetching e-mail-handling e-mail-is e-mail-only e-mail-oriented e-mail-related e-mail-style e-mail-to-fax e-mail-to-print e-mail-to-telex e-mail-to-voice e-mail-to-voice-mail e-mail-to-voice-mail e-mailed e-mailing

Figure 41: Selections from large equivalence classes.

The size of the classes clearly necessitates that some sort of structure and order be introduced to the classes. Without this structuring the task of selecting terms for query expansion is impractical. An initial naïve and inexpensive approach is to present only those terms that do not include hyphens and apostrophes. For example, if the term to be expanded was *book*, the user would be presented with the terms *book, booked, booking, books*, rather than the large list of terms shown in Figure 41. Although not optimal, this approach is adequate, pending the implementation of class structuring, since the existing MultiText index for TREC treats hyphens and apostrophes as whitespace.

Despite inflating the class size, hyphenated terms have the advantage of providing some additional context that can be helpful for query expansion. An example where hyphenated terms may be useful is in performing an expansion using the *mail* equivalence class (see Figure 41). If the user is interested in e-mail rather than conventional mail it would be useful to display or automatically expand the query to include all compound words that include e-mail as a sub-component by specifying **e-mail** or *electronic-mail**.

Another example where the context provided by hyphenated words is helpful is when searching for documents about people with "driven" personalities. Documents that contain the word *driven* as part of a hyphenated term may be relevant and a good indicator of relevance. Documents containing terms such as *career-driven, casual-but-driven, competency-driven, greed-driven, panic-driven* etc. are likely to be relevant. However, documents containing terms such as *command-line-driven, application-driven, battery-driven, chauffeur-driven* are clearly not appropriate. What is needed is a facility that allows the user to easily zoom in on the desired compound terms without being overwhelmed by the volume of data.

5.4. Query Based Stemming Experiments

The purpose of the query based stemming experiments is to examine the effects of allowing a user to determine candidate terms for expansion and control the expansion process. Of particular interest is the performance of query based stemming compared to no stemming and fully automatic stemming.

Previous research has suggested that in some cases the impact of stemming algorithms on retrieval performance is small, and thus difficult to detect. Therefore, it is important to use a large number of queries in order to make it easier to detect significant differences between methods. A large and diverse document collection is also desirable so that the results are not influenced by domain or document characteristics. As outlined earlier, the TREC collection meets these requirements.

The adhoc queries developed by the University of Waterloo MultiText project for TREC-4 were used as the basis for the query based stemming queries. These queries are actually compound queries consisting of an ordered list of sub-queries. The documents retrieved for each sub-query are ranked separately. The results are then combined into a final solution where the results of the each sub-query are ranked before the results of subsequent queries. The first query in the list is intended to be a precise expression of query topic, while later queries are successively weaker, designed to increase recall and satisfy the artificial requirement to retrieve 1,000 documents. The original queries contain limited manual expansion of query terms with morphological variants. The manual expansion of terms was removed and the resulting queries became the baseline for this study.

For the expansion experiments a copy of the baseline query was marked up, using dollar signs to delimit candidate terms for expansion. Query terms were delimited if the user thought the inclusion of morphological variants of the term would improve the query.

```
<top>
<num> Number:  250
<desc> Description:
Does available data show a positive correlation between the sales
of firearms and ammunition in the U.S. and the commission of
crimes involving firearms?
</top>
```

Figure 42: User need statement for TREC query 250

```
@output "250.output"
sales = "sale" + "sales" + "purchase" + "purchases" + "buying" +
"buy"
aguns0 = "handgun" + "handguns" + (("hand" <> ("gun" + "guns")) <
[2])
aguns1 = (("automatic" + "assault") <> ("weapons" + "weapon")) <
[2]
aguns = aguns0 + aguns1
guns0 = "firearm" + "firearms" + "weapon" + "weapons"
guns1 = "gun" + "guns" + "rifle" + "rifles" + "shotgun" +
"shotguns"
guns2 = "pistol" + "pistols" + "revolver" + "revolvers"
guns = guns0 + guns1 + guns2
crime0 = "drugs" + "crack" + "gang" + "gangs"
crime1 = "violence" + "crime" + "crimes"
crime = crime0 + crime1
q1 = aguns + (crime^guns)
q0 = sales^q1
@rank 250 q0 q1
```

Figure 43: Original query used by MultiText project in TREC-4.

```
@output "250.output"
sales = "sale"  + "purchase" +  "buy"
aguns0 = "handgun" + (("hand" <> ("gun")) < [2])
aguns1 = (("automatic" + "assault") <> ("weapon")) < [2]
aguns = aguns0 + aguns1
guns0 = "firearm" +  "weapon"
guns1 = "gun" +  "rifle" + "shotgun"
guns2 = "pistol" + "revolver"
guns = guns0 + guns1 + guns2
crime0 = "drugs" + "crack" + "gang"
crime1 = "violence" + "crime"
crime = crime0 + crime1
q1 = aguns + (crime^guns)
q0 = sales^q1
@rank 250 q0 q1
```

Figure 44: Query 250 with morphological variants removed.

```
@output "250.output"
sales = $sale$  + $purchase$ +  $buy$
aguns0 = $handgun$ + (("hand" <> ($gun$)) < [2])
aguns1 = (("automatic" + "assault") <> ($weapon$)) < [2]
aguns = aguns0 + aguns1
guns0 = $firearm$ +  $weapon$
guns1 = $gun$ +  $rifle$ + $shotgun$
guns2 = $pistol$ + $revolver$
guns = guns0 + guns1 + guns2
crime0 = $drugs$ + "crack" + $gang$ ^M
crime1 = $violence$ + $crime$
crime = crime0 + crime1
q1 = aguns + (crime^guns)
q0 = sales^q1
@rank 250 q0 q1
```

Figure 45: Query 250 marked for expansion.

The query based stemming experiments involved running six batches of queries against the TREC adhoc collection and comparing their results. Two filtered runs were developed for comparison of different term selection strategies. The six test runs are as follows :

Baseline

MultiText TREC-4 queries with morphological variants removed

Fully Expanded

Queries with all the marked terms fully expanded using equivalence classes based on the Porter Stemmer.

Filter 1

The fully expanded queries filtered to remove expanded terms that the user viewed as clearly distant from the original query term and context.

Filter 2

The fully expanded queries filtered to include only terms that the user considered close to the original query term and context.

Manual Expand

The original MultiText TREC-4 queries which include manually expanded morphological variants.

Approximate to Manual Expand

Fully expanded queries filtered to be as close as possible to the manually expanded queries.

5.5. Results

Figure 46 presents a summary of the experimental results from the six query runs. This data shows that the set of fully expanded queries retrieved the greatest number of documents. Although, the *Filter 1* run retrieved fewer documents overall, it retrieved more *relevant* documents than any other method. Similarly, while the *Filter 2* run retrieved 675 fewer documents than the *Fully Expanded* version, it includes only 3 fewer relevant documents. These results demonstrate that the filtering process was successful in decreasing the number of non-relevant documents retrieved, while maintaining or even increasing the number of relevant documents retrieved.

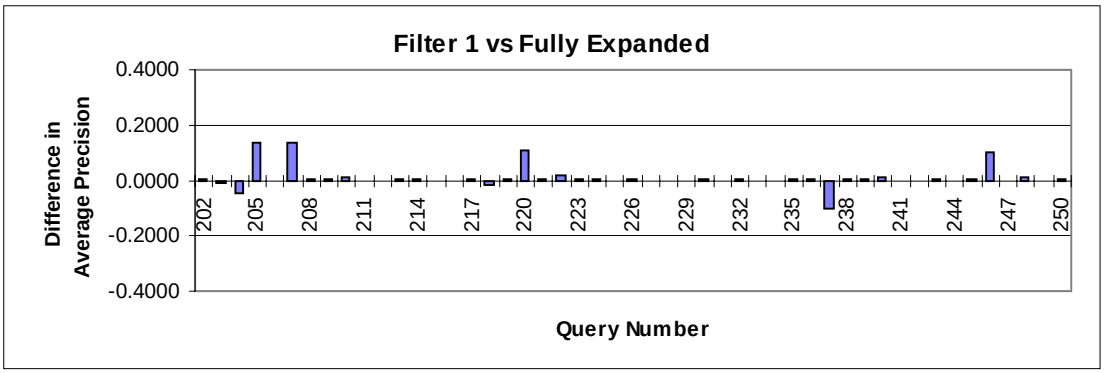
The last two columns of Figure 46 summarize the total number of queries that were positively and negatively effected by each method. The query based stemming runs compared to full stemming consistently increased the number of queries performing above baseline, and consistently decreased the number of queries performing below baseline. These values confirm that the user has the ability to filter out some of the terms that adversely effect queries. Comparing the filtering strategies, the conservative selection (Filter 2) of terms shows both the greatest increase in positively performing queries and the greatest decrease in negatively performing queries.

The results of the experimental runs are encouraging. But, they also indicate that a significant number of queries still perform below baseline, even after filtering. What is not clear from Figure 46 is if these queries perform *significantly* below the baseline, whether it is the same queries that consistently perform poorly across methods, or how the queries are effected by the filtering process. Figure 47 indicates that query based stemming performs worse than full stemming for a few queries. The standard IR measures are not able to identify whether filtering significantly changes the set of documents retrieved or if it the differences are because relevant documents are ranked higher. In order to really understand the effects of query based stemming, examples of queries where the filtering made a significant difference need to be examined.

	Number Retrieved	*Relevant Retrieved (of 6501)*	*Precision*	*Recall*	*Average Precision*
Baseline	40139	3935	.0980	.6053	.2503
Fully Expanded	44371	4338	.0978	.6673	.2741
Filter 1	44102	4401	.0998	.6770	.2793
Filter 2	43695	4335	.0992	.6668	.2679
Manual Expand	42937	4361	.1016	.6708	.2995
Approx. Manual Expand	42886	4323	.1008	.6650	.2901

	Number of Queries Better than Baseline	*Number of Queries Worse than Baseline*
Baseline	-	-
Fully Expanded	27	19
Filter 1	30	15
Filter 2	34	13
Manual Expand	32	11
Approx. Manual Expand	32	11

Figure 46: Summary of query based stemming experimental results.



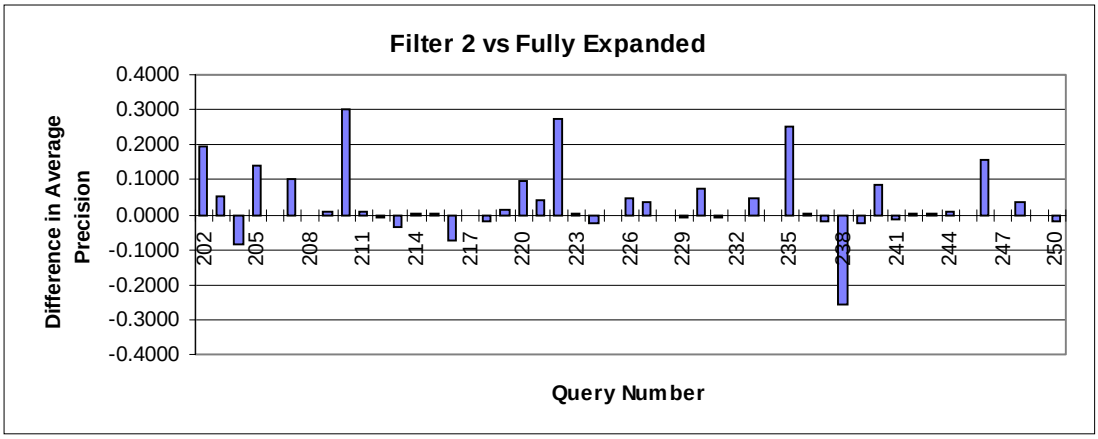


Figure 47 : Comparison of query based stemming and full stemming results.

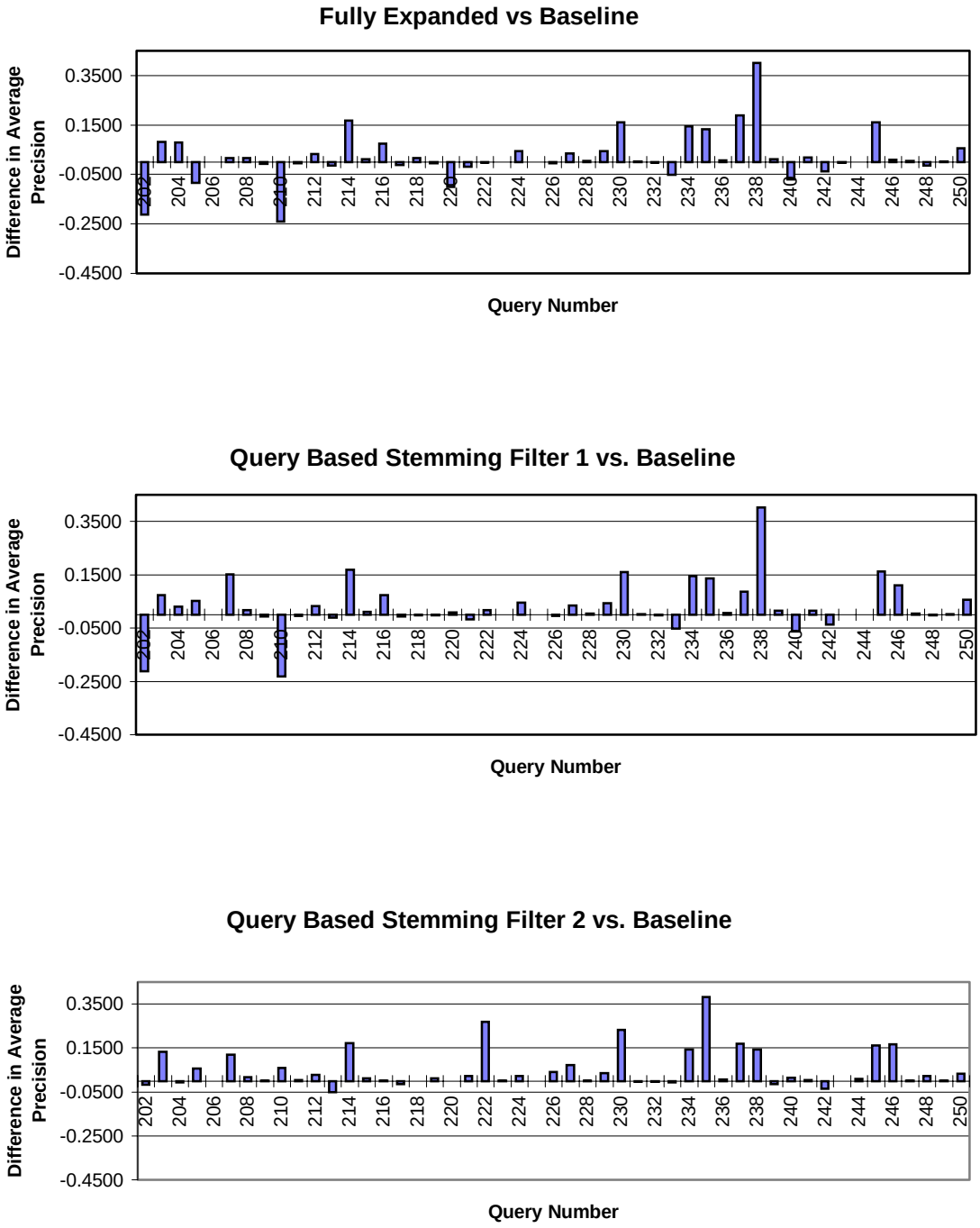


Figure 48: Comparison of stemming performance to baseline on a query by query basis

The data presented in Figure 48 compares the performance of the three stemming methods with the baseline for each query. These charts suggest that queries that performed worse than the baseline when full stemming was applied (e.g. queries 202, 210, 220, 233, 240, 242) tend to improve slightly or stay the same with the *Filter 1* strategy. The *Filter 2* strategy provided slight to dramatic (202, 210) improvements over the *Filter 1* results for these same queries.

Most queries that performed well with full stemming continued to perform well when filtering was applied. A few queries did slightly worse and a few queries did significantly (222, 240) better. The graph for *Filter 2* clearly shows that the beneficial effects of query based stemming on queries is significantly greater than the negative effects. These results support the hypothesis that the user has the ability to identify morphological variants that negatively effect retrieval.

Unfortunately, Figure 48 also shows that some queries perform worse than the baseline when query based stemming is applied. For example, the *Filter 1* strategy applied to query 202 performs significantly worse than the baseline. However, the *Filter 2* strategy for the same query performs comparable to the baseline. To determine why the performance of the filtering strategies are significantly different for this query the documents retrieved by the three methods were examined.

Of the 1,000 documents retrieved by *Filter 1* and *baseline*, 520 documents were common to both methods. Of these 520 common documents, 164 were relevant. The two pools of 480 documents unique to each method contain 19 relevant documents for the *baseline*, and 17 unique documents for *Filter 1*. This makes for a total of 183 relevant documents for the baseline and 181 relevant documents for the *Filter 1* run. Although the actual documents returned by each method are significantly different, the number and composition of relevant documents are very similar. These results show that the number of relevant documents retrieved across methods is comparable, but that the ranking of these documents distinguishes one method from the other. Confirming this observation, the *Filter 2* run which performs almost on par with the baseline contains the exact same set of relevant

documents as the poorly performing *Filter 1*. Clearly the difference in performance between the two filtering methods is strongly influenced by ranking.

To understand how the query expansion and ranking interact it is helpful to look at examples of documents that were ranked significantly differently for the various versions of query 202. The topic for query 202 is "status of nuclear proliferation treaties – violations and monitoring". The actual text of the three versions of the query appears in Appendix H.

The document in Figure 49 is an example of a document that was judged non-relevant by the assessors. The *Filter 1* version of the query ranked this document first, while the baseline query ranked the document 774 out of 1,000. Terms that appear in the baseline version of the query are marked in boldface, terms that were added in the *Filter 1* version of the query are underlined in the document. The occurrence of variants of *inspect* (7) and *missile* (2) in the *Filter 1* version contribute to the high rank of this document. The query is designed to rank documents that satisfy the macros *treaty* (e.g. treaty, agreement, SALT*)*, *nukes* (e.g. nuclear, missile, weapon etc.) and monitor (e.g. inspect, monitor, observe etc.). If this does not result in the required pool of 1,000 documents, documents that contain terms that satisfy *nukes* and one of *treaty* or *monitor* are considered relevant. The additional variants of *inspect* in the *Filter 1* version of the query results in three text intervals (highlighted) that satisfy the high priority sub-query *treaty^nukes^monitor* causing the document to be ranked highly. Since the *monitor* macro in the baseline query does not contain any variants of inspect, this document only satisfies the lower priority query of *treaty^nukes* (the four satisfying text intervals are denoted in capital letters).

<DOC>
<DOCNO> AP880719-0188 </DOCNO>
<FILEID>AP-NR-07-19-88 1939EDT</FILEID>
<FIRST>r i AM-EGermany-US 07-19 0234</FIRST>
<SECOND>AM-EGermany-US,0240</SECOND>
<HEAD>U.S. **Nuclear Missile** Inspection Teams in East Germany</HEAD>
<HEAD>With AM-Britain-US-Soviets</HEAD>
<DATELINE>BERLIN (AP) </DATELINE>
<TEXT>
U.S officials are in East Germany inspecting **nuclear missile** sites in compliance with the superpowers'Intermediate **NUCLEAR** FORCES **TREATY**, the state-run news agency AND said Tuesday. ADN, in a brief report, said that two U.S. inspection teams arrived in East Germany on Monday. One team began checking a **missile** site near the town of Waren, about 70 miles north of Berlin the same day. The East German news agency also said the other U.S. inspection team started checking a second site Tuesday at Koenigsbrueck, about 15 miles north of Dresden. According to ADN, the Soviets maintained missiles at the sites until February, when they decided to remove them early to set a positive example in **nuclear** disarmament. The inspections in East Germany come as two teams of Soviet inspectors checked **nuclear** **MISSILE** SITES IN BRITAIN, WHICH ALSO FALL UNDER THE INF **AGREEMENT**. Soviet teams have already conducted inspections at **MISSILE** SITES IN WEST GERMANY AND THE UNITED STATES THIS MONTH, AT THE SAME TIME U.S. INSPECTION TEAMS BEGAN DEPLOYING TO WARSAW PACT LOCATIONS. THE INSPECTIONS FALL UNDER THE TERMS OF THE INF **TREATY** signed in Washington last Dec. 8 by Soviet leader Mikhail Gorbachev and President Reagan. The ***TREATY*** *PROVIDES FOR THE ELIMINATION WITHIN THREE YEARS OF ALL* ***NUCLEAR*** missiles with ranges from 340 miles to 3,000 miles.
</TEXT>
</DOC>

Figure 49: Non-relevant document ranked highly by filter 1.

The document in Figure 50 was also judged non-relevant for query 202 by the assessors. The *Filter 1* version of the query resulted in this document being ranked 278, while the baseline version of the query ranked the document 916, and the *Filter 2* version did not retrieve the document in the pool of the top 1,000 documents. Terms that appear in the baseline version of the query appear in bold, terms added by the *Filter 1* version are underlined, and terms that are removed by the *Filter 2* version appear in strikeout. The

major difference between the ranking of the *Filter1* version and the other versions of the query is the inclusion of the words *monitored*(1) and *inspectors*(2). Since this document was judged non-relevant, the removal of *monitored* and *inspectors* by the *Filter 2* version of the query was beneficial.

<DOC>
<DOCNO> AP880216-0158 </DOCNO>
<FILEID>AP-NR-02-16-88 2056EST</FILEID>
<FIRST>r w AM-DismantlingWarheads 02-16 0573</FIRST>
<HEAD>Scientists Say US, Soviet **Nuclear** Warheads Should Be Dismantled</HEAD>
<BYLINE>By BRYAN BRUMLEY</BYLINE>
<BYLINE>Associated Press Writer</BYLINE>
<DATELINE>WASHINGTON (AP) </DATELINE>
<TEXT>

A group of American scientists on Tuesday disputed Reagan administration statements that it was not practical to verify the dismantlement of **nuclear** warheads under an arms control **agreement**.

Members of the Federation of American Scientists outlined at a news conference a dismantlement plan worked out earlier this month in conjunction with the Committee of Soviet Scientists Against the **Nuclear** Threat, headed by Roald Sagdayev, head of the Soviet civilian space agency.

Jeremy J. Stone, president of the U.S. group, and members Frank von Hippel and Theodore B. Taylor, said they were concerned that Defense Secretary Frank Carlucci may have raised ``misconceptions'' in defending the Intermediate-range **Nuclear** Forces pact. The pact calls for the elimination of all medium-range U.S. and Soviet missiles.

Carlucci and other administration officials, defending the **treaty** against conservative Sen. Jesse Helms, R-N.C., said it did not require the destruction of warheads from the banned missiles because verifying that step could expose secrets on how the devices are built.

The administration further argued that destroying the warheads would free up weapons-grade plutonium which could be used by terrorists. And officials said the Soviets could simply produce more **weapon** material to replace any destroyed. Although Stone, Taylor and Von Hippel support the INF pact, they took issue with Carlucci, and proposed a system under which warheads could be dismantled and the material either used in reactors or buried underground.

Under the plan, the Soviet Union and United States would each designate a dismantlement site, which would be closely ~~monitored~~ to prevent the clandestine entry or exit of **nuclear** material, said Taylor, who worked in the national laboratory at Los Alamos in 1949-56 designing **nuclear** weapons.

Nuclear warheads entering the site would be ``tagged'' for identification and ``fingerprinted'' by neutron probes, x-rays or gamma rays which the scientists said would verify that the devices contained **nuclear** materials but would not disclose their inner workings. The warheads would then be dismantled behind closed doors at the nation which owned them, and ~~inspectors~~ from the other nation would verify that **nuclear** material had left the site. In the case of American warheads, Soviet ``~~inspectors~~'' would only have to verify that a batch of nuclear warheads had entered a U.S. dismantlement facility and that, after all the warheads had been dismantled and their components destroyed beyond recognition, no intact nuclear warheads remained inside, the scientists said in a statement.

Taylor and Von Hippel recommended that weapons-grade plutonium from the warheads be mixed with radioactive wastes and buried. They recommended that highly enriched uranium also used in **nuclear** weapons be used to fuel reactors for electric power plants. The estimated cost of dismantling the warheads, about $6 billion, would be dwarfed by the value of the fuel, about $50 billion, they said. Taylor said in a paper that ``the approximately 1,000 tons of U-235 (enriched uranium) and 200 tons of plutonium in the world's more than 50,000 **nuclear** warheads could be consumed as fuel in all U.S. and Soviet **nuclear** power plants in seven years, or in all the world's **nuclear** power plants in about three years.''

Von Hippel acknowledged that the scientists had not devised a fool-proof plan to enable each side to be assured that the correct type and number of warheads were being dismantled under an arms control **agreement** involving multiple systems.</TEXT></DOC>

Figure 50: Non-relevant document ranked highly by filter 1 and not retrieved by filter 2.

This example shows that the correct choice of expansion terms is difficult to predict and dependent on the document collection. Expansion terms may be consistent with the intent of the query and still cause non-relevant documents to be ranked highly, resulting in loss of precision.

The table in Appendix I compares the average precision values for each query and method. When the overall average precision is calculated for each method, the manually expanded queries have the best precision, followed by the three query based stemming

methods. The slightly superior performance of the manually expanded queries is disappointing. This performance difference is because some deficiencies in the stemming algorithm cause variants of terms that were included in the manually expanded queries to be omitted from the automatically expanded queries. For example, the manually expanded version of query 222 contains the following macro:

```
DETERRENT = “deter” + “deters” + “deterrent” + “deterrence”
```

The fully expanded version of the same macro is:

```
DETERRENT = “deter” + “deters” +  “deterent” + “detering” +
            “deterred” + “deterring” + “deterence”
```

Notice that the term *deterent* (misspelling) occurs in the fully expanded version, but *deterrent* (correct spelling) does not. This is because the Porter stemming algorithm stems *deterrence* to *deterr*, which is another class. Interestingly, the Porter Stemmer handles *deterred* and *deterring* correctly. The class construction method should be modified to merge classes with similar stems to compensate for this weakness in the stemming algorithm. Similar stems could be identified by applying recoding rules to stem endings or by partial matching algorithms such as the one described in section 2.2.

The table in Appendix L ranks the performance of each method on a query by query basis. This is an analysis technique suggested by Hull (see section 2.4) for comparing the performance of stemming strategies across queries. All methods show variation in ranking from 1 down to 5. When judged by average rank, *manual expand* and *filter 1* have the best average, followed by the other filtering methods. Significantly more analysis with this and other collections is needed before we can arrive at definitive conclusions about the best approach to pursue.

Chapter 6

6. Conclusions

6.1. Summary

The difficulty of matching a user's need for information with the content of documents is a central problem in IR. Queries are constructed by using combinations of terms to form a representation that approximates the user's information need. Decisions on document relevance are made based on the number and frequency of terms the query and document have in common. One method to address the IR problem is to build tools to help the user select better search terms. The addition of well selected query terms aids in defining the information need and provides the opportunity for more document matches.

One approach to expanding queries is through the addition of morphological variants of the query terms by performing stemming at index time. Morphological variants are good candidates for query expansion because term variants often represent similar concepts. Past studies have shown that the addition of variants is an effective re-call enhancing device. Unfortunately, not all morphological variants share similar semantic meanings (i.e.

orientation, oriental). While the stemming process is effective through adding useful terms to a query, it also has the potential to add irrelevant terms which degrade query performance.

A weakness of the conventional implementation of stemming at index time is that the query expansion is pre-determined. All variants of a term are automatically included with no regard for consistency with the query. The addition of non-relevant terms by stemming has been identified as one of the main causes of bad mistakes in IR [Croft95].

A variety of attempts have been made to define what constitutes an inappropriately expanded term and to develop methods for excluding such terms from query expansion. Ideally, stemming should only be applied in situations where it adds terms consistent with the query and results in improved retrieval performance. To date, attempts to automatically predict these situations and selectively apply stemming have been unsuccessful.

The work in this thesis has explored the idea of moving stemming from index time to query time. The advantage to this approach is that it allows decisions about word conflation to be made dynamically when the query is formulated.

An architecture for query based stemming has been presented and implemented. The resulting system, Inter-Stem, was used to study and evaluate whether a user's knowledge can be incorporated effectively during the query formulation process to determine when and how query terms should be expanded. The results indicate that user-mediated stemming is successful in decreasing the number of non-relevant documents retrieved by stemming, and at increasing the number of relevant documents retrieved compared to the baseline (no stemming).

The query based stemming experiments involved the application of two distinct expansion strategies. The first strategy (Filter 1) was to accept all variant terms proposed by the system, except for those that were clearly semantically *distant* from the original query term and context. The second strategy (Filter 2) was to expand the query only to include those terms that were semantically *close* to the original query term and context. When the

number of queries that benefit is measured against the number of queries that suffer under each method, the conservative approach (Filter 2) outperformed the more liberal approach (Filter 1).

The conclusion of this study is that a user's knowledge can be successfully leveraged to perform selective stemming. An interactive stemmer such as Inter-Stem should be considered a worthwhile tool for query formulation and modification.

6.2. Future Work

Detailed evaluation of individual query performance indicates that the difference in performance between methods was often a result of the interaction of stemming and ranking. It was not unusual for the various approaches to expansion to retrieve a similar set of relevant documents, but for the ranks to vary widely. The ranking system used in our experiments is based on the shortest substring model. The results may differ when using a ranking method based on frequency and distribution of terms. Further study into the interaction between stemming and ranking methods is needed.

The queries for our experiments were compound queries written in a boolean subset of the GCL query language. The effect of stemming on at least one of the queries was strongly influenced by the boolean nature of the queries. Further study is needed to determine if the interaction between stemming and queries is consistent across query methods (i.e., are similar for boolean and natural language style queries).

Chapter 4 proposes a variety of methods for ordering and presenting the word classes produced by stemming. The ideas of computing lexical distance and partitioning classes into concept clusters were proposed. Further work is needed to develop these ideas before they can be included in an interactive stemming system.

As part of such research, an evaluation of the various ordering techniques is needed. The classes that were used in our experiments could be ordered using the various methods and the results compared with the actual terms included in the expanded queries.

The interactive filtering and term ordering processes suggested for stemming could be extended to apply to terms produced by relevance feedback. A study by Koenemann and Belkin concluded that user control over the addition of query terms by relevance feedback is helpful [KB96]. Users who were provided interactive control required less iterations to achieve comparable results than did users in less interactive feedback conditions. Ideas for term ordering and the use of classes of terms explored in this thesis could be applied to further improve interactive relevance feedback.

Finally, usability and extensions to the interactive version of Inter-Stem should be explored. First a system should be implemented based on the preliminary design that was presented. This can be used as the basis for a usability study to determine which features improve retrieval performance. Thereafter, modifications can be incorporated and tested to determine appropriate features for a more effective interface.

Bibliography

[Avis89] Avis, W.S. *Funk and Wagnalls Canadian College Dictionary*, Fitzhenry and Whiteside, Toronto, 1989.

[AB74] Adamson, G., Boreham, J. The Use of an Association Measure Based on Character Structure to Identify Semantically Related Pairs of Words and Document Titles. In *Information Storage and Retrieval*, 10:253-260, 1974.

[BC92] Belking, N.J., Croft, W.B. Information Filtering and Information Retrieval: Two Sides of The Same Coin? In *Communications of the ACM*, 35(12):29-38, December 1992.

[BCK95] Belkin, N.J., Cool, C., Koenemann, J. et al. Using Relevance Feedback and Ranking in Interactive Searching. In *The Fourth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology, U.S Department of Commerce, 1996.
http://potomac.ncsl.gov/TREC/t4_proceedings.html

[BSAS95] Buckley, C., Salton, G., Allan, J., Singhal, A. Automatic Query Expansion Using SMART : TREC 3. In *The Third Text REtrieval Conference (TREC-3)*. National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 500-225, 1995.

[BRP95] Byrd, R., Ravin, Y., Prager, J.. Lexical Assistance at the Information Retrieval User Interface. In *4th Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*. Las Vegas, NV: University of Nevada, 1995.

[Bur96] Burkowski, F.J. *Course Notes for CS748G*, Department of Computer Science, University of Waterloo, Winter 1996.

[CC93] Callan, J.P., & Croft, W.B. An Evaluation of Query Processing Strategies Using The Tipster Collection. In *Proceedings of ACM SIGIR International Conference on Research and Development in Information Retrieval*, pp. 347-356, 1993.

[CCG95] Charoenkitkarn, N., Chignell, M.H., Golovchinsky, G. Is Recall Relevant? An Analysis of How User Interface Conditions affect Strategies and Performance in Large Scale Text Retrieval. In *The Fourth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology, U.S Department of Commerce, 1996.
http://potomac.ncsl.gov/TREC/t4_proceedings.html

[Cla1894] Clarke, V.M. *The Complete Concordance to Shakespeare*, Bickers & Son, London, 1894.

[CCB95a] Clarke, C.L.A., Cormack, G.V., Burkowski, F.J. An Algebra for Structured Text Search and a Framework for its' Implementation. In *The Computer Journal* 38(1),43-56, 1993.

[CCB95b] Clarke, C.L.A., Cormack, G.V., Burkowski, F.J. Schema-Independent Retrieval from Hetrogeneous Structured Text. In *Proceedings of The Fourth Annual Symposium on Document Analysis and Information Retrieval* Las Vegas, Nevada pp.279-289, April, 1995.

[CCB95c] Clarke, C.L.A., Cormack, G.V., Burkowski, F.J. Shortest Substring Ranking (MultiText Experiments for TREC-4). In *The Fourth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology, U.S Department of Commerce, 1996.
http://potomac.ncsl.gov/TREC/t4_proceedings.html

[CMK66] Cleverdon, C.W., Mills, J. and Keen, E.M. In *Factors Determining the Performance of Indexing Systems, Vol 1 Design, Vol 2 Test Results*. Aslib Cranfield Reasearch Project, Cranfield England, 1966.

[Crouch88] Crouch, C.J. A Cluster Based Approach to Thesaurus Construction. In *Proceedings of ACM SIGIR International Conference on Research and Development in Information Retrieval,* 1988.

[CX95] Croft, W.B., Xu, J. Corpus-Specific Stemming Using Word Form Co-occurence. In *4th Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*. Las Vegas, NV: University of Nevada, pp 485- 502, 1988.

[Croft95] Croft, W.B. What Do People Want from Information Retrieval? In *D-Lib Magazine*. pp 485- 502, December 1995.

[Dai90] Dairymple, P. Retrieval by Reformulation in Two Library Catalogues: Toward a Cognitive Model of Searching Behaviour. In *Journal of the American Association for Information Science*, 41(4):272-281, 1990.

[EB94] Efthimidadis, E., Biron, P.V. UCLA-Okapi at Trec-2: Query Expansion Experiments, In *The Second Text REtrieval Conference (TREC-2)*. National Institute of Standards and Technology, U.S. Department of Commerce, March 1994. NIST Special Publication 500-215.

[Eft93] Efthimiadis, E. A User-Centered Evaluation of Ranking Algorithms for Interactive Query Expansion. In *Proceedings of the 16th ACM/SIGIR Conference* pp. 36-47, 1993.

[ELK91] Egan, D., Lesk, M., Ketchum, R.D. et al. Hypertext for the Electronic Library? CORE Sample Results, In *Third ACM Conference on Hypertext Proceedings*. San Antonio, Texas, pp. 299-312,1991.

[Field75] Field, B.J., Semi-automatic development of thesaurii using free-language vocabulary analysis, British Library Research. Development. Department,1975. Report 5260.

[Fox92] Fox, C. Lexical Analysis and Stoplists. In Frakes, W.B. & Baeza-Yates, R. (Eds.), *Information Retrieval, Data Structures and Algorithms*. New Jersey: Prentice-Hall, pp. 102-130, 1992.

[Frakes92a] Frakes, W.B. Stemming Algorithms. In Frakes, W.B. & Baeza-Yates, R. (Eds.), *Information Retrieval, Data Structures and Algorithms*. New Jersey: Prentice-Hall, pp. 131-160, 1992.

[Frakes92b] Frakes, W.B. Introduction to Information Storage and Retrieval Systems. In Frakes, W.B. & Baeza-Yates, R. (Eds.), *Information Retrieval, Data Structures and Algorithms*. New Jersey: Prentice-Hall., pp. 131-160,1992.

[FB92] Frakes, W.B. & Baeza-Yates, R. (Eds.). *Information Retrieval, Data Structures and Algorithms.* New Jersey: Prentice-Hall, 1992.

[Gre92] Grefenstette, G. Use of Syntactic Context to Produce Term Association Lists for Text Retrieval. In *Proceedings of ACM SIGIR International Conference on Research and Development in Information Retrieval,* 1992.

[HD80] Hall, P.A.V., Dowling, G.R., Approximate String Matching. In *Computing Surveys*, 12(4):381-402, 1980.

[Har88] Harman, D. Towards Interactive Query Expansion. In *Proceedings of ACM SIGIR International Conference on Research and Development in Information Retrieval,* 1988.

[Har91] Harman, D. How Effective is Suffixing? In *Journal of the American Society for Information Science* 42(1):7-15, 1991.

[Har93] Harman, D. Overview of the First Text REtrieval Conference. In *Proceedings of the 16th ACM/SIGIR Conference*. New York: Association for Computing Machinery. pp. 36-47,1993.

[Har95] Harman, D. (Ed.). Overview of the Third Text REtrieval Conference (TREC-3). In *Proceedings of The Third Text REtrieval Conference (TREC-3).* National Institute of Standards and Technology, U.S. Department of Commerce, 1995. NIST Special Publication 500-225.
http://potomac.ncsl.gov/TREC/t3_proceedings.html

[Har96] Harman, D. Overview of The Fourth Text REtreival Conference (TREC-4), Overhead slides. In *Proceedings of The Fourth Text REtrieval Conference (TREC-4).* National Institute of Standards and Technology, U.S Department of Commerce, 1996.
http://potomac.ncsl.gov/TREC/t4_proceedings.html

[HWW86] Hendry, I.G., Willet, P., Wood, F.E. INSTRUCT: A Teaching Package for Experimental Methods in Information Retrieval. Part I The User's View. In *Program* 20(3):245-263, 1986.

[Hull96] Hull, D. Stemming Algorithms: A Case Study for Detailed Evaluation. In *Journal of the American Society for Information Science*. 47(1):70-84, 1996.

[JC94] Jing, Y., Croft, B.W., An Association Thesaurus for Information Retrieval. In *Proceedings of RIAO*, 1994.

[KB96] Koenemann, J., Belkin, N., A Case for Interaction : A Study of Interactive Information Retrieval Behaviour and Effectiveness. In *Proceedings of Computer Human Interaction*. ACM Press, New York pp.205-212, 1996.

[KC92] Krovetz, R., Croft, W.B., Lexical Ambiguity in Information Retrieval. In *ACM Transactions on Information Systems*, 10(2):115-141, April 1992.

[Krov93] Krovetz, R. (1993). Viewing Morphology as an Inference Process. In *Proceedings of the 16th ACM/SIGIR Conference* (pp. 191-202). New York: Association for Computing Machinery.

[Kuh91] Kuhithau, C. Inside the Search Process: Information Seeking from the User's Perspective. In *Journal of the American Society for Information Science*, 42(5):361-371, 1991.

[LPTW81] Lennon, J., Pierce, D., Tarry, B., Willet, P. An Evaluation of Some Conflation Algorithms for Information Retrieval. In *Journal of Information Science*. 3: 177-183, 1981.

[Lov68] Lovins, J. Development of a Stemming Algorithm. In *Mechanical Translation and Computational Linguistics*, 11: 22-31,1968.

[LRK73] Lowe, T.C., Roberts, D.C., Kurtz, P. Additional Text Processing For On-line Retrieval (The RADCOL System), Technical Report TADC-TR-73-337, 1973.

[LK95] Lu, A.X., Keefer, R.B. Query Expansion/Reduction and its Impact on Retrieval Effectiveness. In *Proceedings of The Third Text REtrieval Conference (TREC-3)*. National Institute of Standards and Technology, U.S. Department of Commerce, 1995. NIST Special Publication 500-225.

[Mal91] Malmkjaer, K. *The Linguistics Encyclopedia*. New York: Rouledge, 1991.

[MH91] Morris, J., Hirst, G. Lexical Cohesion computed by thesaural relations as ain indicator of the structure of text. In *Computational Linguistics* 17(1):21-49, 1991.

[Pai94] Paice, C.D., An Evaluation Method for Stemming Algorithms. In *Proceedings of the 17th ACM/SIGIR conference, pp.42-49,* 1994.

[PP78] Pacak, M.B., Pratt, A.W., Identification and Transformations of Terminal Morphemes in Medical English. Part II, In *Methods of Information In Medicine*, 17:95-100,1978.

[Port80] Porter, M.F. An Algorithm for Suffix Stripping. *Program*, 14(3):130-137, (1980).

[PW91] Peat, H.J., Willet, P. The Limitations of Term Co-Occurrence Data for Query Expansion in Document Retrieval Systems. In *Journal of the American Society for Information Science*, 42(5):378-383.

[QF93] Qiu, Y., Frei, H.P. Concept Based Query Expansion, In *Proceedings of the 16th ACM/SIGIR Conference,* pp. 160-169. New York: Association for Computing Machinery.

[Ras92] Rassmeussen, E. Clustering Algorithms In Frakes, W.B. & Baeza-Yates, R. (Eds.), *Information Retrieval, Data Structures and Algorithms*. New Jersey: Prentice-Hall, pp. 419-442, 1992.

[Sal68] Salton, G.. *Automatic Information Organization and Retrieval*. New York: McGraw-Hill, 1968.

[Sal83] Salton, G.. *Introduction to Modern Information Retrieval*. New York: McGraw-Hill, 1983.

[Sal86] Salton, G. Another Look at Automatic Text-Retrieval Systems. In *Communications of the ACM*, 29(7):648-658,1986.

[San94] Sanderson, M. Word Sense Disambiguation and Information Retrieval. In *Proceedings of the 17th ACM/SIGIR conference, pp.61-69,* 1994.

[Sav93] Savoy, J. Stemming of French Words Based on Grammatical Categories. In *Journal of the American Society for Information Science*. 44(1):1-9, 1993.

[Seu60] Seuss, Dr., *Green Eggs and Ham*. Beginner Books Inc. New York.

[Shaks] Shakespeare, Wm. *The Tragedy of Romeo and Julliet*, Signet Classic, New York, 1964.

[Ste96] Steinberg, S.G., Seek and Ye Shall Find (Maybe). In *Wired Magazine 4.05*, pp108, 1996.

[Sto95] St. Onge, D. Detecting and Correcting Malapropisms with Lexical Chains. Technical Report CSRI-319, University of Toronto, 1995.

[UD83] Ulmschneider, J., Doszkocs, T., A Practical Stemming Algorithm For Online Search Assistance. In *Online Review*, 7:301-315, 1983.

[Van79] vanRijsbergen, C.J. *Information Retrieval*, Butterworths. London, 1979. http://dcs.glasgow.ac.uk/keith

[Van84] vanRijsbergen, C.J. Research and Development in Information Retrieval. In *Prodeedings of the Third Joint BCS and ACM Symposium.* July, 1984.

[Voo93] Voorhees, E.M.. Using WordNet to Disambiguate Word Senses for Text Retrieval. In *Proceedings of the 17th ACM/SIGIR conference.* pp.61-69, 1993.

[Voo94] Voorhees, E.M. Query Expansion Using Lexical-Semantic Relations. In *Proceedings of the 17th ACM/SIGIR Conference.* pp.61-69, 1994.

[Wal88] Walker, S.. Improving Subject Access Painlessly: Recent Work on the Okapi Online Catalogue Projects. In *Program*, 22(1):21-31, 1988.

[WMB94] Witten, I.H., Moffat, A., Bell, T.C., *Managing Gigabytes : Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York, 1994.

[Xiao94] Xiao, H. Using a Thesaurus for Query Refinement in Full-Text Retrieval. *UW Center for the New Oxford English Dictionary and Text Research*. University of Waterloo Technical Report OED-94-01, 1994.

[Zipf45] Zipf, G.K. The meaning-frequency relationship of words. In Journal of General Psychology, 3:251-256, 1945.

Appendix A

The Porter Stemming Algorithm[Port80]

In order to explain the Porter stemming algorithm in detail, some terms and notation need to be defined.

1. *Consonant* is a letter other than A,E,I,O or U, and other than Y preceded by a consonant.
2. *Vowel* is any letter that is not a consonant.
3. *C* represents a sequence of consonants of length greater than 0.
4. *V* represents a sequence of vowels of length greater than 0.
5. All words can be written in the form $[C](VC)^m[V]$. Where the contents of the square brackets are optional. *m* is called the *measure* of the word or word part that it is attached to (see Figure 51 for examples).
6. *<X> indicates that the stem ends with a the letter X.
7. *v* indicates the stem contains a vowel.
8. *d indicates the stem ends in a double consonant.
9. *o indicates the stem ends with a consonant-vowel-consonant sequence. The final consonant must not be w,x,or y.
10. Rules are given in the form (*condition*)S1→S2. Meaning that if a word terminates with the suffix S1, and the stem preceding S1 satisfies the given condition, S1 is replaced with S2.

Measure	*Examples*
m=0	TR, EE, TREE, Y, BY
m=1	TROUBLE, OATS, TREES, IVY
m=2	TROUBLES,PRIVATE, ORRERY

Figure 51: Examples of the *measure* of a word

For each set of rules, only the one with the longest matching S1 for the given word is applied.

Step 1 deals with plurals and past participles.

Step 1a

Conditions	*Suffix*	*Replacement*	*Examples*
NULL	sses	ss	caresses→caress
NULL	ies	i	ponies→poni
			ties→ti
NULL	ss	ss	caress→caress
NULL	s	NULL	cats→cat

Step 1b

Conditions	*Suffix*	*Replacement*	*Examples*
(m>0)	eed	ee	agreed→agree
			feed→feed
(*v*)	ed	NULL	plastered→plaster
			bled→bled
(*v*)	ing	NULL	motoring→motor
			sing→sing

Step 1b1 (to be performed if the second or third rules of Step 1b is successful)

Conditions	*Suffix*	*Replacement*	*Examples*
NULL	at	ate	conflat(ed)→conflate
NULL	bl	ble	troubl(ing)→trouble

Conditions	*Suffix*	*Replacement*	*Examples*
NULL	iz	ize	siz(ed)→size
(*d and not(*<L>or *<S> or *<Z>))	NULL	single letter	hopp(ing)→hop tann(ed)→tan fall(ing)→fall hiss(ing) →hiss fizz(ed)→fizz
(m=1 and *o)	NULL	e	fail(ing)→fail fil(ing)→file

Step 1c

Conditions	*Suffix*	*Replacement*	*Examples*
(*v*)	y	i	happy→happi sky→sky

Step 2

Conditions	*Suffix*	*Replacement*	*Examples*
(m>0)	ational	ate	relational→relate
(m>0)	tional	tion	conditional→condition rational→rational
(m>0)	enci	ence	valenci→valence
(m>0)	anci	ance	hesitanci→hesitance
(m>0)	izer	ize	digitizer→digitize
(m>0)	abli	able	conformabli→ conformable
(m>0)	alli	al	radicallie→radical
(m>0)	entli	ent	differentli→different
(m>0)	eli	e	vileli→vile
(m>0)	ousli	ous	analogousli→analagous
(m>0)	ization	ize	vietnamization→ vietnamize
(m>0)	ation	ate	predication→predicate
(m>0)	ator	ate	operator→operate

Conditions	*Suffix*	*Replacement*	*Examples*
(m>0)	alism	al	feudalism→feudal
(m>0)	iveness	ive	decisiveness→decisive
(m>0)	fulness	ful	hopefulness→hopeful
(m>0)	ousness	ous	callousness→callous
(m>0)	aliti	al	formaliti→formal
(m>0)	iviti	ive	sensitiviti→sensitive
(m>0)	biliti	ble	sensibilit→sensible

Step 3

Conditions	*Suffix*	*Replacement*	*Examples*
(m>0)	icate	ic	triplicate→triplic
(m>0)	ative	NULL	formative→form
(m>0)	alize	al	formalize→formal
(m>0)	iciti	ic	electriciti→electric
(m>0)	ical	ic	electrical→electrc
(m>0)	ful	NULL	hopeful→hope
(m>0)	ness	NULL	goodness→good

Step 4

Conditions	*Suffix*	*Replacement*	*Examples*
(m>1)	al	NULL	revival→reviv
(m>1)	ance	NULL	allowance→allow
(m>1)	ence	NULL	inference→infer
(m>1)	er	NULL	airliner→airlin
(m>1)	ic	NULL	gyroscopic→gyroscop
(m>1)	able	NULL	adjustable→adjust
(m>1)	ible	NULL	defensible→defens
(m>1)	ant	NULL	irritant→irrit
(m>1)	ement	NULL	replacement→replac
(m>1)	ment	NULL	adjustment→adjust
(m>1)	ent	NULL	dependent→depend
(m>1)	(*S) ION	NULL	expansion→expans
(m>1)	(*T)) ION	NULL	adoption→adopt
(m>1)	ou	NULL	homologou→homolog
(m>1)	ism	NULL	communism→commun
(m>1)	ate	NULL	activate→activ
(m>1)	iti	NULL	angularity→angular
(m>1)	ous	NULL	homologous→homolog
(m>1)	ive	NULL	effective→effect
(m>1)	ize	NULL	bowdlerize→bowdler

Step 5a

Conditions	*Suffix*	*Replacement*	*Examples*
(m>1)	e	NULL	probate→probat
			rate→rate
(m=1)	not *oE	NULL	cease→ceas

Step 5b

Conditions	*Suffix*	*Replacement*	*Examples*
(m>1)	*d and *<L>	single letter	controll→control
			roll→roll

Appendix B

Description	*Number of occurrences*
Contains Multiple Apostrophes	175
multiple as a result of erroneous trailing apostrophe	104
multiple due to occurrence of 'n' (i.e. mix’n’match, rock’n’roll)	20
Suffixes	
's	1,721,065
n't	371,990
' includes s'	202,756
've	31,047
're	30,228
'll	24,253
'd	20,501
in'	2,926
'n	230
'a most commonly a typographical error for 's	77
'em	43
'ing	
Prefixes	
o'	973
d'	795
l'	136
j'	86
n'	67
e'	39
a'	35

Description	*Number of occurrences*
Other	51,735
two words concatenated where the first contains a suffix which includes an apostrophe	2,730
Grand Total	2,459,173

Figure 52: Ananalysis of apostrophe data.

Appendix C

A sample equivalence class that was formed using the text collection from TREC-4.

object object objected objecter objecters objectic objecting objection objectional objections objective objectively objectiveness objectives objectivity objectize objectized objectizing objects object' object's objective' objectives' objectivity' objectivity's objectize' objects' objects's a-match-the-objects-behind-the-doors action-object action-object-actor alice-as-object almost-object application-as-object base-object c-object-oriented categories-objectives class-and-object class-and-objects class-object classand-object classes-as-objects code-attached-to-objects collection-of-named-objects copy-object corp-object cross-object data-object data-objects' document-object elements-objects embedded-object essentials-tools-objects euro-objections faint-object for-object foreign-object-ingestion full-object gateway-between-object generic-object graphical-object graphics-object has-object-orientation hate-object hope-object hyper-object hyper-objects input-object inter-object knowledge-objects large-object less-than-objective macro-object management-by-objective match-the-objects meta-object meta-objects mini-object mixed-object model-object molded-object money-is-no-object money-no-object more-object-oriented more-objective multi-object multi-objective multiple-object-type multiple-objective near-object-oriented nexpert-object no-objection non-object non-object-oriented non-objecting non-objection non-objective not-so-object-oriented not-so-objective noun-verb-object object-a object-access object-action object-analysis object-approach object-attribute object-attribute-value object-aware object-background object-base object-based object-binding object-blend object-brothers object-browsing object-by object-by-object object-capable object-center object-centered object-centric object-changing object-checking object-class object-classes object-code object-code-compatible object-code-level object-coded object-communication object-compatible object-coordinate object-creation object-cutting object-data object-database object-definition object-description object-development object-document object-drawing object-driven object-duplication object-editable object-editing object-embedded object-embedding object-engaging object-engine object-enhanced object-ese object-extended object-file object-flow object-format object-friendly object-graphic object-graphics object-handling object-identifier object-identifiers object-in object-including object-independent object-instance object-intelligent object-interchange object-interface object-is object-kit object-laden object-leaning

object-level object-library object-light object-like object-linkable object-linked object-linking object-linking-and-embedding object-locating object-locking object-mad object-magement object-making object-management object-manipulation object-mapping object-masking object-message object-message-object object-message-object object-messaging object-metamorphosis object-method object-model object-modeling object-module object-morphing object-motion object-naming object-navigator object-ness object-neutral object-number object-obsessed object-ofiented object-oiriented object-on-screen object-only object-oreinted object-orented object-organized object-oricnted object-orient object-orientable object-orientated object-orientation object-oriente object-oriented object-oriented' object-oriented-applications object-oriented-development object-oriented-language object-oriented-like object-oriented-ness object-oriented-programming object-oriented-something object-orientede object-orientedness object-orientedness' object-orientedoffspring object-orienteering object-orienteers object-orineted object-orinted object-oti object-pascal object-pixel object-placement object-polygon-laser object-position object-pretty object-printed object-processing object-programming object-property-role object-property-role-relationship object-queue object-recognition object-refinement object-reflected object-related object-relationship object-rendering object-request-broker object-saving object-scaling object-sciences object-script object-selection object-sensitive object-sharing object-snap object-something-or-other object-space object-speak object-specific object-star object-storage object-store object-stores object-structured object-style object-such object-supplied object-supporting object-swapping object-technology object-the object-throwing object-to object-to-object object-to-relational object-to-screen object-tracking object-transferring object-transmission object-type object-value object-valued object-verb-subject object-view object-views object-which object-windows object-within-window object-works object-wringing object-z objected-oriented objected-provides objective-c objective-conflict objective-order objective-oriented objective-setting objective-side objective-to objectivity-based objects-all-the-way objects-by objects-documents objects-encapsulated objects-for-users objects-images objects-including objects-it objects-modular objects-oriented objects-radio objects-resources objects-shell objects-to objects-tools objects-when part-object per-object persistent-object price-is-no-object print-object pro-object pseudo-object-oriented pseudo-objects quasi-object-oriented ray-object rectangular-object-of-interest reference-object reference-objects remote-object-accessing result-objective revenue-objectives rule-and-object-based same-object says-object-oriented scope-and-objectives semi-object-oriented shared-object single-object soft-object solid-object standard-object sub-objects subject-object-matrix subject-verb-object task-object test-objective text-as-object text-object

Appendix D

List of suffixes used in the English language (from the *Oxford English Dictionary*).

-a
-ac
-acy
-ad
-adelphia
-ado
-aemia
-al
-amy
-an
-ance
-ancy
-and
-ane
-ant
-ard
-arious
-arium
-ary
-at
-ate
-atic
-atile
-ator
-by
-cade
-cide
-dione
-dom
-dyne
-ean
-ed
-ee
-een
-eer
-el
-els
-en
-ence
-ene
-enic
-enous
-ent
-eous
-er
-ern
-ery
-escence
-escent
-ese
-esque
-ess
-est
-et
-etin
-ette
-etum
-fic
-fication
-fold
-fue
-ful
-fy
-head
-hood
-i
-ia
-ial
-ian
-iana
-iasis
-ice
-id
-idene
-idin
-idine
-il
-ile
-in
-ina
-ine
-ion
-ismus
-ist
-ite
-ition
-itis
-itol
-ium
-ive
-kin
-kins
-le
-lecithal
-less
-let
-lin
-lock
-lon
-ly
-mo
-more
-most
-mycin
-ness
-o
-oan
-ode
-oid
-ol
-ola
-ole
-oloy
-on
-one
-onic
-onium
-orama
-oside
-ot
-ote
-ior
-ism
-our
-ous
-parous
-paus
-phaous
-plastic
-rihts
-ry
-s
-ship
-sis
-some
-speak
-sphere
-ster
-sterol
-style
-teria
-th
-thon
-trice
-tron
-trophic
-tropic
-tude
-ty
-type
-ual
-ular
-ule
-ulent
-ulose
-uncle
-up
-ville
-vorous
-wards
-wick
-y
-yer
-ylidene
-zoiteto

Appendix E

Appendix F

Brief introduction to GCL syntax

Each statement is terminated with a newline

```
statement := definition | Q
```

A definition associates the query with the symbol. The symbol may be used in place of *Q* in future queries.

```
Definition := symbol = Q
Q:= Q1 < Q2       Q1 contained in Q2
    | Q1/< Q2     Q1 not contained in Q2
    | Q1 > Q2     Q1 containing Q2
    |Q1 /> Q2     Q2 not contained in Q2
    | Q1 <> Q2    Q1 followed by Q2
    |Q1 ^ Q2      both of Q1 and Q2
    | Q1 + Q2     one of Q1 or Q2
    | term
    | fixed_size_interval
```

A term is a string of lowercase letters delimited by double quotes (e.g. “stargazer”).

A fixed size interval is an integer in square brackets (e.g. [3]).

For example, the query to find the phrase “boldly go” near “five year mission” is :

```
 ((“boldly” <> ”go”)<[2]) ^ ((“five” <> “year” <> “mission”)<[3])
```

or

```
P1 = (“boldly” <> ”go”)<[2])
P2 = (“five” <> ”year” <> “mission”)<[3])
P1 ^ P2
```

Appendix G

20 Largest Equivalence Classes

Rank	*Stem*	*Size*
1.	non	5699
2.	to	4843
3.	and	3913
4.	base	3855
5.	a	3235
6.	on	2878
7.	the	2744
8.	like	2467
9.	n	2452
10.	in	2429
11.	type	2239
12.	anti	1952
13.	of	1899
14.	re	1843
15.	pre	1779
16.	relat	1625
17.	style	1473
18.	self	1419
19.	size	1322
20.	high	1319

Appendix G

Top 49 Words by Frequency of Occurrence

Rank	***Word***	***Frequency***	***Rank***	***Word***	***Frequency***
1.	the	16,762,164	26.	o	825,017
2.	of	8,051,232	27.	have	811,152
3.	to	6,885,009	28.	not	764,468
4.	a	6,587,970	29.	but	744,537
5.	and	6,562,235	30.	its	669,230
6.	in	5,138,292	31.	i	577,317
7.	for	2,914,554	32.	new	570,721
8.	is	2,811,991	33.	one	552,985
9.	that	2,308,968	34.	can	532,109
10.	m	2,302,719	35.	they	530,685
11.	p	2,023,210	36.	s	518,055
12.	said	1,808,111	37.	his	506,748
13.	on	1,795,191	38.	more	502,757
14.	by	1,598,246	39.	about	484,860
15.	as	1,558,829	40.	other	477,403
16.	be	1,493,659	41.	no	472,415
17.	at	1,339,175	42.	also	456,265
18.	at	1,339,175	43.	their	449,157
19.	from	1,257,247	44	system	435,975
20.	are	1,240,892	45.	data	435,603
21.	an	1,192,092	46.	been	434,242
22.	or	1,148,526	47.	than	424,732
23.	this	896,522	48.	u	414,168
24.	he	860,764	49.	if	413,042
25.	has	854693			

Appendix H

Information Need Statement for Query 202

Status of nuclear proliferation treaties -- violations and monitoring.

Baseline Query

```
nukes0 = (("atom" + "atomic" + "hydrogen") <> ("bomb" + "bombs"))
           < [2]
nukes1 = "thermonuclear" + (("thermo" <> "nuclear") < [2])
nukes2a = "weapon" + "weaponry" + "missile" + "missiles"
nukes2b = "submarine" + "submarines" + "sub" + "subs"
nukes2c = "reactor" + "reactors"
nukes2 = (("nuclear" + "atomic") <> (nukes2a + nukes2b +
            nukes2c)) < [2]
nukes3 = "plutonium" + "calutron"
nukes = nukes0 + nukes1 + nukes2 + nukes3
treaty0 = "SALT" + "START"
treaty1 = "treaty" + "treaties" + "agreement" + "agreements"
treaty2 = ("test" <> "ban") < [2]
treaty = treaty0 + treaty1 + treaty2
monitor0 = "enforcement" + "enforce" + "limitation" +
            "proliferation"
monitor1 = "supervision"  + "monitor" + "monitors" + "monitoring"
monitor2 = "violation" + "violated" + "violating" + "violates"
monitor3 = "inspect" + "inspects" + "inspecting" + "inspected"
monitor4 = "observation" + "observer" + "observers"
monitor = monitor0 + monitor1 + monitor2 + monitor3 + monitor4
q0 = treaty^nukes^monitor
q1 = treaty^("nuclear" + "atomic")^monitor
q2 = (treaty^nukes) + (nukes^monitor)
@rank 202 q0 q1 q2
```

Filter 1

```
nukes0 = (("atom" + "atome" + "atomic" + "atomically" + "atomics"
      + "atoms" + "hydrogen") <> ("bomb" + "bombe" + "bombed" +
      "bombing" + "bombings" + "bombs")) < [2]
nukes1 = "thermonuclear" + (("thermo" <> "nuclear") < [2])
nukes2a = "weapon" + “weapons” + "weaponry" + "missil" +
      "missile" + "missiles"
nukes2b = "submarine" + "submarines" +  "sub" + "subs"
nukes2c = "reactor" + "reactors"
nukes2 = (("nuclear" + "atomic") <> (nukes2a + nukes2b +
      nukes2c)) < [2]
nukes3 = "plutonium" + "calutron"
nukes = nukes0 + nukes1 + nukes2 + nukes3
treaty0 = "SALT" + "START"
treaty1 = "treaties" + "treaty" + "agreemeents" + "agreement" +
      "agreements" + "agreemment"
treaty2 = (("test" + "tested" +  "testing" + "testings" + "tests"
      + "testting")<> ("ban" + "banned" + "banning" + "bannings"
      + "bans")) < [2]
treaty = treaty0 + treaty1 + treaty2
monitor0 = "enforcable" + "enforce" + "enforceability" +
      "enforceable" + "enforceably" + "enforced" + "enforceent" +
      "enforcement" + "enforcements" + "enforces" + "enforcible"
      + "enforcing" + "enforcment" +  "limit" + "limitation" +
      "limitations" + "limite" + "limited" + "limiteds" +
      "limites" + "limiting" + "limitive" + "limits" +
      "proliferate" + "proliferated" + "proliferates" +
      "proliferating" + "proliferation" + "proliferations"
monitor1 = "supervise" + "supervised"  + "supervises" +
      "supervising" + "supervision" + "supervisions" + "monitor"
      + "monitorable" + "monitored" + "monitoring" +
      "monitorings" + "monitors"
monitor2 = "violatated" + "violate" + "violated" + "violater" +
      "violaters" + "violates" + "violating" + "violatings" +
      "violation" + "violations" + "violator"+ "violators"
monitor3 = "inspect" + "inspected" + "inspecting" + "inspection"
      + "inspectional" + "inspections" + "inspects"
monitor4 = "observance" +  "observation" + "observations" +
      "observe" + "observed" + "observer" + "observers" +
      "observes" + "observing"
monitor = monitor0 + monitor1 + monitor2 + monitor3 + monitor4
q0 = treaty^nukes^monitor
q1 = treaty^("nuclear" + "nucleare" + "nuclearization" +
      "nuclearized" + "atom"+ "atome" + "atomic" + "atomically" +
      "atomics" + "atoms")^monitor
```

```
q2 = (treaty^nukes) + (nukes^monitor)
@rank 202 q0 q1 q2
```

Filter 2

```
nukes0 = (("atom" + "atomic"  + "hydrogen") <> ("bomb" +
      "bombing" + "bombings" + "bombs")) < [2]
nukes1 = "thermonuclear" + (("thermo" <> "nuclear") < [2])
nukes2a = "weapon" + “weapons” + "weaponry" + "missile" +
      "missiles"
nukes2b = "submarine" + "submarines" +  "sub" +"subs"
nukes2c = "reactor" + "reactors"
nukes2 = (("nuclear" + "atomic") <> (nukes2a + nukes2b +
      nukes2c)) < [2]
nukes3 = "plutonium" + "calutron"
nukes = nukes0 + nukes1 + nukes2 + nukes3
treaty0 = "SALT" + "START"
treaty1 = "treaties" + "treaty" + "agreement" + "agreements"
treaty2 = (("test" +  "testing" +  "tests" ) <> ("ban" + "banned"
      + "banning" +"bannings" + "bans")) < [2]
treaty = treaty0 + treaty1 + treaty2
monitor0 = "enforce"  + "enforced" + "enforcement"  + "enforces"
      +  "enforcing"+ "enforcment" +  "limit" + "limitation" +
      "limitations"  + "limited" + "limiteds" +  "limites" +
      "limiting" + "limits" + "proliferate" + "proliferated" +
      "proliferates" +  "proliferation"
monitor1 = "supervise"  + "supervises" + "supervising" +
      "supervision"   + "monitor"  + "monitoring" " + "monitors"
monitor2 = "violatated" + "violate" + "violated" + "violater" +
      "violaters" + "violates" + "violating" + "violatings" +
      "violation" + "violations" + "violator"+ "violators"
monitor3 = "inspect" + "inspected" + "inspecting" + "inspection"
      + "inspections" + "inspects"
monitor4 =  "observation" + "observations" + "observe" +
      "observed" + "observer" + "observers" + "observes" +
      "observing"
monitor = monitor0 + monitor1 + monitor2 + monitor3 + monitor4
q0 = treaty^nukes^monitor
q1 = treaty^("nuclear"  + "atom" +  "atomic")^monitor
q2 = (treaty^nukes) + (nukes^monitor)
@rank 202 q0 q1 q2
```

Appendix I

Average Precision By Query

Query #	Baseline	Manually Expanded	Fully Expanded	Filter 1	Filter 2	Approx. Man. Expand	Max
202	0.3370	0.3212	0.1247	0.1249	0.3212	0.1925	0.3370
203	0.0403	0.1720	0.1219	0.1134	0.1720	0.1819	0.1819
204	0.1202	0.1132	0.1986	0.1509	0.1132	0.1102	0.1986
205	0.1619	0.2178	0.0782	0.2143	0.2178	0.2178	0.2178
206	0.0049	0.0049	0.0049	0.0049	0.0049	0.0049	0.0049
207	0.3938	0.5130	0.4101	0.5444	0.5130	0.5161	0.5444
208	0.0060	0.0227	0.0221	0.0232	0.0221	0.0247	0.0247
209	0.3134	0.3134	0.3072	0.3076	0.3150	0.3134	0.3150
210	0.4914	0.5512	0.2504	0.2589	0.5512	0.5512	0.5512
211	0.2585	0.2593	0.2544	0.2544	0.2613	0.2693	0.2693
212	0.1143	0.1426	0.1472	0.1472	0.1416	0.1426	0.1472
213	0.2572	0.2052	0.2424	0.2460	0.2052	0.2001	0.2572
214	0.0423	0.2142	0.2103	0.2104	0.2142	0.2144	0.2144
215	0.5683	0.5799	0.5797	0.5797	0.5799	0.5799	0.5799
216	0.4713	0.4723	0.5455	0.5455	0.4723	0.4723	0.5455
217	0.2091	0.1943	0.1983	0.2033	0.1943	0.1943	0.2091
218	0.0259	0.0244	0.0421	0.0244	0.0244	0.0244	0.0421
219	0.0586	0.0627	0.0534	0.0565	0.0699	0.0697	0.0699
220	0.4328	0.4308	0.3333	0.4407	0.4308	0.2917	0.4407
221	0.2049	0.2265	0.1867	0.1875	0.2262	0.2265	0.2265
222	0.1687	0.4394	0.1662	0.1869	0.4377	0.1893	0.4394
223	0.0258	0.0258	0.0249	0.0258	0.0260	0.0258	0.0260
224	0.4535	0.4756	0.4979	0.4999	0.4752	0.4756	0.4999
225	0.6835	0.6833	0.6833	0.6833	0.6831	0.6834	0.6835
226	0.1639	0.2041	0.1595	0.1603	0.2042	0.2041	0.2042
227	0.4054	0.4775	0.4409	0.4409	0.4775	0.4567	0.4775
228	0.0083	0.0109	0.0127	0.0124	0.0109	0.0109	0.0127
229	0.5418	0.5766	0.5859	0.5859	0.5766	0.5766	0.5859
230	0.3952	0.6277	0.5554	0.5555	0.6280	0.6277	0.6280
231	0.1077	0.1025	0.1091	0.1091	0.1025	0.1025	0.1091
232	0.0151	0.0111	0.0133	0.0140	0.0111	0.0111	0.0151
233	0.6163	0.6163	0.5652	0.5652	0.6108	0.6163	0.6163
234	0.5590	0.7029	0.7029	0.7029	0.7029	0.7029	0.7029
235	0.3035	0.6865	0.4362	0.4393	0.6865	0.6558	0.6865
236	0.0050	0.0116	0.0114	0.0115	0.0116	0.0097	0.0116
237	0.3913	0.5610	0.5802	0.4780	0.5610	0.5598	0.5802
238	0.0413	0.1853	0.4430	0.4431	0.1853	0.1734	0.4431

Appendix I

Query #	Baseline	Manually Expanded	Fully Expanded	Filter 1	Filter 2	Approx. Man. Expand	Max
239	0.2248	0.2094	0.2369	0.2410	0.2094	0.2230	0.2410
240	0.2748	0.2892	0.2039	0.2110	0.2892	0.2927	0.2927
241	0.0187	0.0221	0.0367	0.0336	0.0221	0.0221	0.0367
242	0.5585	0.5221	0.5221	0.5221	0.5232	0.5221	0.5585
243	0.0083	0.0076	0.0065	0.0085	0.0077	0.0078	0.0085
244	0.5063	0.5150	0.5064	0.5064	0.5150	0.5063	0.5150
245	0.0231	0.1835	0.1846	0.1848	0.1837	0.1845	0.1848
246	0.1443	0.3310	0.1534	0.2552	0.3110	0.2940	0.3310
247	0.3886	0.3896	0.3936	0.3935	0.3896	0.3894	0.3936
248	0.5371	0.5583	0.5222	0.5350	0.5581	0.5374	0.5583
249	0.0279	0.0295	0.0297	0.0297	0.0295	0.0295	0.0297
250	0.1550	0.1889	0.2105	0.2108	0.1889	0.1889	0.2108
overall	0.2503	0.2997	0.2715	0.2793	0.2994	0.2873	0.3155

Appendix J

Ranked Performance of Query Method on a query by query basis

Query #	Baseline	Manually Expanded	Fully Expanded	Filter 1	Filter 2	Approx. Manual Expansion
202	1	2	6	5	3	4
203	5	2	3	4	2	1
204	3	4	1	2	4	5
205	3	1	4	2	1	1
206	1	1	1	1	1	1
207	5	3	4	1	3	2
208	5	3	4	2	4	1
209	2	2	4	3	1	2
210	2	1	4	3	1	1
211	4	3	5	5	2	1
212	4	2	1	1	3	2
213	1	4	3	2	4	5
214	5	2	4	3	2	1
215	3	1	2	2	1	1
216	3	2	1	1	2	2
217	1	4	3	2	4	4
218	2	3	1	3	3	3
219	4	3	6	5	1	2
220	2	3	4	1	3	5
221	3	1	4	4	2	1
222	5	1	4	3	2	3
223	2	2	3	2	1	2
224	5	3	2	1	4	3
225	1	3	3	3	4	2
226	3	2	5	4	1	2
227	4	1	3	3	1	2
228	4	3	1	2	3	3
229	3	2	1	1	2	2
230	5	2	4	3	1	2
231	2	3	1	1	3	3
232	1	4	3	2	4	4
233	1	1	3	3	2	1
234	2	1	1	1	1	1
235	5	1	4	3	1	2
236	5	1	3	2	1	4
237	5	2	1	4	2	3
238	3	4	2	1	4	5
239	2	5	1	3	5	4
240	3	2	5	4	2	1

Appendix J

Query #	Baseline	Manually Expanded	Fully Expanded	Filter 1	Filter 2	Approx. Manual Expansion
241	4	3	1	2	3	3
242	1	3	3	3	2	3
243	2	5	6	1	4	3
244	3	1	2	2	1	3
245	6	5	2	1	4	3
246	6	1	5	4	2	3
247	5	3	1	2	3	4
248	4	1	6	5	2	3
249	3	2	1	1	2	2
250	4	3	2	1	3	3
Average	3.22	2.39	2.94	2.45	2.39	2.53

Appendix K

Performance Compared to baseline

Query #	*Full Expansion*	*Filter 1*	*Filter 2*	*Approx. Man Expansion.*	*Manual Expansion*
202	-	-	-	-	-
203	+	+	+	+	+
204	+	+	-	-	-
205	-	+	+	+	+
206	0	0	0	0	0
207	+	+	+	+	+
208	+	+	+	+	+
209	-	-	+	0	0
210	-	-	+	+	+
211	-	-	+	+	+
212	+	+	+	+	+
213	-	-	-	-	-
214	+	+	+	+	+
215	+	+	+	+	+
216	+	+	+	+	+
217	-	-	-	-	-
218	+	-	-	-	-
219	-	-	+	+	+
220	-	+	-	-	-
221	-	-	+	+	+
222	-	+	+	+	+
223	-	0	0	0	0
224	+	+	+	+	+
225	0	0	-	0	0
226	-	-	+	+	+
227	+	+	+	+	+
228	+	+	+	+	+
229	+	+	+	+	+
230	+	+	+	+	+
231	+	+	-	-	-
232	-	-	-	-	-
233	-	-	-	0	0
234	+	+	+	+	+
235	+	+	+	+	+
236	+	+	+	+	+
237	+	+	+	+	+
238	+	+	+	+	+
239	+	+	-	-	-
240	-	-	+	+	+

Query #	*Full Expansion*	*Filter 1*	*Filter 2*	*Approx. Man Expansion.*	*Manual Expansion*
241	+	+	+	+	+
242	-	-	-	-	-
243	-	+	-	-	-
244	0	0	+	0	0
245	+	+	+	+	+
246	+	+	+	+	+
247	+	+	+	+	+
248	-	-	+	+	+
249	+	+	+	+	+
250	+	+	+	+	+
+	27	30	34	32	32
-	19	15	13	11	11

Legend

- \- more than .002 below the baseline
- *0* within ± .002 of baseline
- \+ more than .002 above baseline

Appendix L

Interpolated Precision-Recall Information

	0.00	0.10	0.20	0.30	0.40	0.50
Baseline	0.6518	0.5179	0.4296	0.3503	0.2943	0.2365
Full Expansion	0.7241	0.5339	0.4616	0.3676	0.3127	0.2706
Filter 1	0.7137	0.5445	0.4791	0.3799	0.3208	0.2737
Filter 2	0.7004	0.4955	0.4372	0.3666	0.3126	0.2635
Manual Expansion	0.7235	0.5725	0.5081	0.414	0.3497	0.2973
Approx. Manual Expansion	0.7139	0.5606	0.4928	0.3867	0.332	0.2862

	0.60	0.70	0.80	0.90	1.00
Baseline	0.1797	0.1269	0.0818	0.0448	0.0105
Full Expansion	0.2228	0.1701	0.107	0.0752	0.0095
Filter 1	0.2255	0.1698	0.1082	0.0716	0.0111
Filter 2	0.225	0.1729	0.1005	0.0721	0.0111
Manual Expansion	0.2361	0.1813	0.1177	0.0705	0.0109
Approx. Manual Expansion	0.2345	0.1822	0.1109	0.0709	0.011