A Client Side Tool for Contextual Web Search

by

Hariharan Lakshmanan

Bachelor of Technology, Civil Engineering (2001)
Indian Institute of Technology, Kharagpur

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

at the

Massachusetts Institute of Technology

June 2004

Author _____
Hariharan Lakshmanan
Department of Civil and Environmental Engineering
May 21, 2004

Certified by _____
John R. Williams
Associate Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by _____
Heidi Nepf
Chairman, Committee for Graduate Students

A Client Side Tool for Contextual Web Search

by

Hariharan Lakshmanan

Submitted to the Department of Civil and Environmental Engineering
on May 21, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis describes the design and development of an application that uses information relevant to the context of a web search for the purpose of improving the search results obtained using standard search engines. The representation of the contextual information is based on a Vector Space Model and is obtained from a set of documents that have been identified as relevant to the context of the search. Two algorithms have been developed for using this contextual representation to re-rank the search results obtained using search engines. In the first algorithm, re-ranking is done based on a comparison of every search result with *all* the contextual documents. In the second algorithm, only a *subset* of the contextual documents that relate to the search query is used to measure the relevance of the search results. This subset is identified by mapping the search query onto the Vector Space representation of the contextual documents.

A software application was developed using the .NET framework with C# as the implementation language. The software has functionality to enable users to identify contextual documents and perform searches either using a standard search engine or using the above-mentioned algorithms. The software implementation details, and preliminary results regarding the efficiency of the proposed algorithms have been presented.

Thesis Supervisor: John R Williams
Title: Associate Professor

Acknowledgements

I would first like to thank my advisor, Prof. John Williams, for providing thoughtful guidance at the appropriate time. I also thank Prof. Rory O'Connor for his support.

I would like to thank my friends who have made my stay at M.I.T pleasant and wonderful. Special thanks to Srinivas, Sivaram, Dhanush, Deepak, Phani, and Dinesh. I am indeed lucky to have friends like you all.

I would like to thank my family for supporting me in all my endeavors.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The World Wide Web (WWW) has grown significantly since its inception. There are many studies regarding the size of web and their estimates vary widely. For example, shown in Figure 1 is an estimate of the growth of the so-called public web[1] from one of the studies.

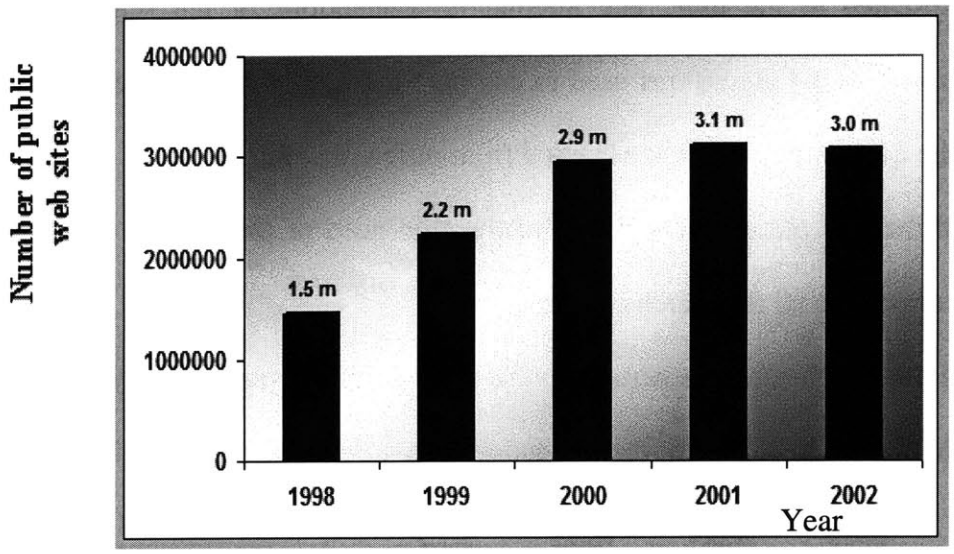

Figure 1 Growth in the number of unique public web sites (E.T. O'Neill April 2003)

The WWW now contains billions of documents. Search engines emerged in order to assist one to find documents that satisfy his/her information need from this vast pool of

[1] A public website is defined as a web site in WWW that offers unrestricted access to a significant portion of its content to the public. The collection of all unique public web sites is defined as public web. E.T. O'Neill, B. F. L., and R. Bennett, (April 2003). Trends in the evolution of the public web. D-Lib Magazine. **9**.

documents. Search engines have become an important tool for people to locate information in the web.

The search engines frequently crawl the WWW and maintain a list[2] of documents contained in the web. The users convey their information need to the search engines mainly in the form of a search query consisting of keywords. The search engines process these keywords and create an internal representation of the search query. The search engines match the representation of the search query with the representation of the documents in their list. The documents are then ranked based on this matching and the results are presented to the user. The internal representations of the search queries themselves as well as the algorithms used to compare these queries with the documents differentiate the quality of the results obtained from different search engines. However, it is to be noted that the presentation of the information need of the user in the form of keywords is common to all the search engines.

The quality of the search results generated by the search engines varies widely. For some of the search queries the search results satisfy the information need of the users. However, in many cases, the user might have to repeat the search several times with modified queries. The main reasons why the search engines do not satisfy the user information needs in most cases are as follows:

1) Given the search query, any reasonable searching algorithm might generate enormous number of documents that qualify as potential results of the search. Note that the search engine has very little information other than the few words

[2] This list is not exhaustive. Every search engine has only a partial index of the entire WWW. The combined coverage of several search engines varies by an order of magnitude S. Lawrence, C. L. G. (1998). "Searching the World Wide Web." Science **280**(5360): 98-100.

that make up the search query in order to understand the information need of the user and its context. This lack of specific information results in many documents that do not satisfy the users need to find a place at the top of the search result set.

2) There are many problems that are inherent to the nature of the medium (WWW) (M. R. Henzinger 2002). For example, there are many websites that are designed to obtain top rankings in search engine results[3]. This leads to several irrelevant documents, not connected to the user information needs, finding a top place in the search result set.

Much attention has gone in tackling the problems arising from the nature of WWW. However, improvement of the quality of search results through a better representation of the user need and the associated context has not been studied in great detail.

1.2 Thesis idea

As noted earlier, the search engine uses the *search query* to get an idea of the user information need. The *context* of the information need is, in many cases, not unambiguous to the search engine. For example, suppose a user is interested in information about "Java Islands" and he searches using the query "Java". A search with the query "Java" using the Google search engine shows that all the top results are about the Java programming language. It is to be noted that the search results do contain documents about the Java Islands though they do not figure among the top search results. The search results could have been improved if the context of the search was conveyed better. In this case, the search query "Java" did not convey the context of the search. A

[3] There are many companies that specialize in what is called "Search engine optimization" i.e., ensuring their clients get top rankings in popular search engine results.

relevant question then is whether it is possible to represent the *context* of the information need in some special format along with the *query* itself when the search request is sent to the search engine. In that case, the algorithm for handling the search requests would become more complex and would reduce the speed and efficiency of the search engines. It is to be noted that the existing functions of the client software (the web browser requesting search information) is to request the server software (search engine) for information, wait for the results, and upon receiving the results display them to the user. In many cases it is possible for the client machines to have access to the information that identifies the context. Therefore, it would be possible for the client machines to improve the search by utilizing the context information in order to refine the search results of the search engines. This assumption forms the main idea of the thesis: the aim of the thesis is to explore the utility of processing the contextual information in the client machines to improve search results.

1.3 Thesis objective

The goal of this thesis is to design and develop a proof-of-concept software application that runs on the client machine, and uses information related to the context of the search to improve the quality of search results. The steps involved can be identified as follows:

1) Develop a representation of the context information.
2) Develop algorithms for improving the search results by using the context representation.
3) Design and implement software application for performing search using the algorithms developed in step 2.

Such an application would throw light on the advantages, as well as the limitations of client side processing of context relevant information in order to improve web search results. The envisaged search process is schematically shown in Figure 2.

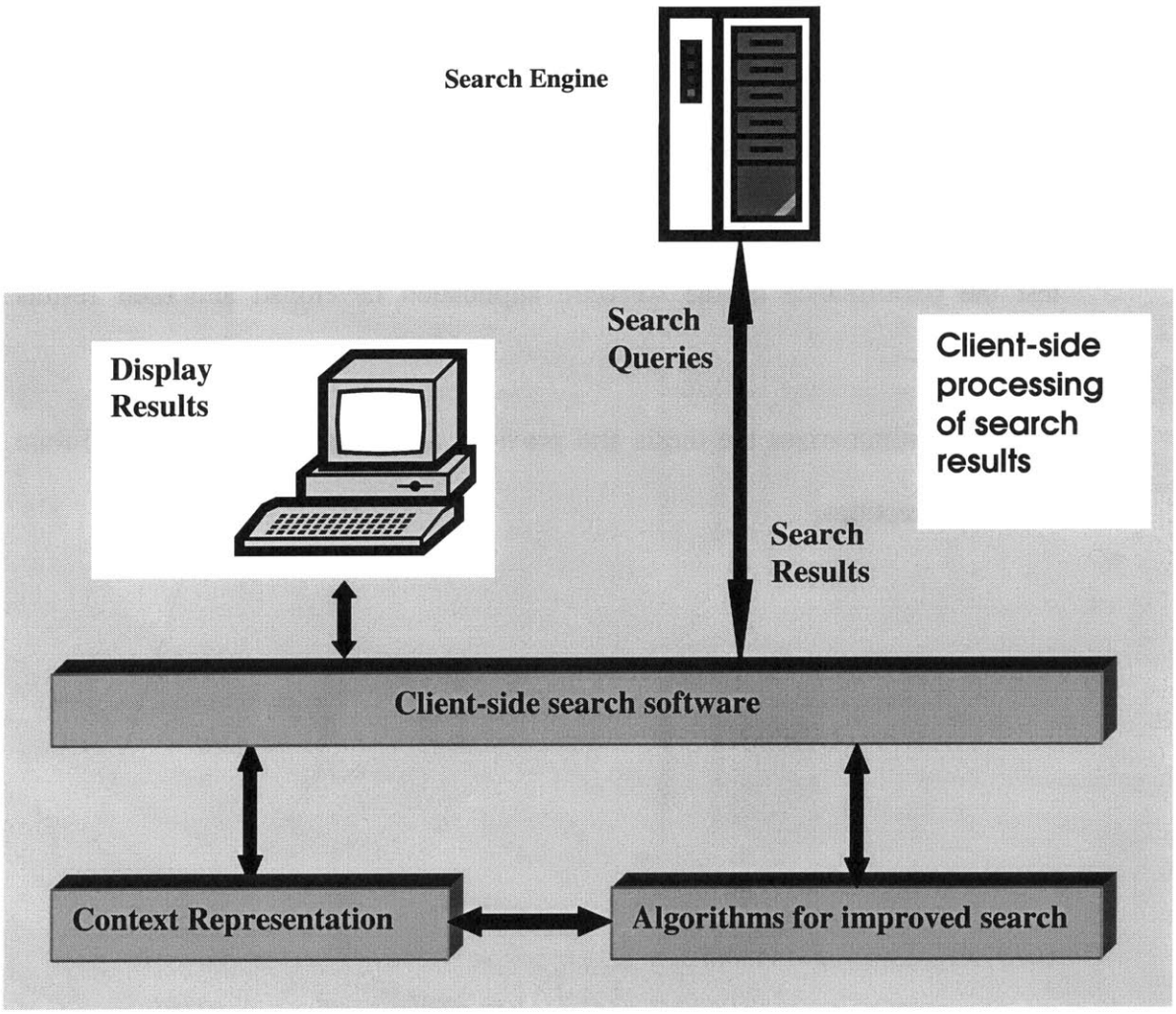

Figure 2: Schematic representation of the proposed search technique

1.4 Thesis layout

- Chapter 2 presents a general overview of issues related to web-based searching. This overview provides a perspective and also clarifies the scope of the present thesis.

- Chapter 3 discusses the proposed technique for improving the search results using information about the context of the search. The technique involves context representation, and algorithms that re-evaluate the search results using the contextual information.
- Chapter 4 discusses the design and development of a software application that implements the algorithms developed.
- Chapter 5 describes the details of some of the experiments conducted in order to test the performance of the software application developed and their results presented.
- Chapter 6 summarizes the thesis and provides a brief outline of possible future research directions.

2 Literature survey

It was mentioned in Chapter 1 that the quality of web search could be improved by using the information about the context of the search. It was also mentioned that there are several issues that affect the quality of search performed by the web search engines. A thorough understanding of these issues is important for using contextual information to enhance the quality of web search. This chapter provides an overview of some of these issues.

2.1 Emerging players in the search engine market

The search engine market is estimated to grow to $ 14.8 billion dollars by the year 2008 (http://news.com.com/2100-1024-5057653.html?tag=nl). The popular search engines today are Google (http://www.google.com), Lycos (http://www.lycos.com) and AltaVista (http://www.altavista.com). Google is, by far, the most widely used search engine. However, the search engine market is getting more competitive with the entry of new players like *Vivisimo* (http://www.vivisimo.com) and *A9* (http://a9.com). Microsoft Corporation is investing heavily in search technology, and this is expected to make the search engine market even more competitive. Besides the increasing market size, a major motivation for the new players is the fact that *search technology has not yet matured.*

2.2 Measure of search engine effectiveness

Currently, there is no accepted standard for measuring the effectiveness of the various search engines. In the field of information retrieval there are standard benchmarks for

testing the effectiveness of an algorithm (Chakrabarti 2003). A benchmark consists of a corpus of n number of pre-identified documents, and a set of queries that are pre-identified for this corpus. For each query, an exhaustive list of relevant documents (D_q) is also identified (manually). While evaluating the effectiveness of any search algorithm, a query q is submitted to the algorithm and the documents returned by the algorithm are analyzed. Let $D = (d_1, d2, \ldots, d_n)$ be the ranked list of documents returned by the algorithm in response to the search query. Let the set $R = (r_1, r_2, \ldots, r_n)$ be defined such that $r_i = 1$, if $d_i \in D_q$.

Recall and precision are two parameters that measure the performance of the search algorithms. *Recall* measures the fraction of the all the relevant documents that are present in the results, and is defined as follows:

$$\text{Recall at rank } k = \frac{1}{|D_q|} \sum_{1 \le i \le k} r_i$$

Precision measures the fraction of D that is relevant, and is defined as follows:

$$\text{Precision at rank k} = \frac{1}{k} \sum_{1 \le i \le k} r_i$$

The precision and recall values from various queries are combined in specific ways to produce a precision-recall graph for a search algorithm for that corpus. The precision-recall graph reflects the performance of the search algorithm with respect to the benchmark corpus used.

It is noteworthy that, given the nature of WWW it is very difficult to establish a standard corpus for web search engines. Nevertheless, the precision and recall parameters would be useful in understanding the performances of web search engines.

2.3 Challenges in web based search

It is reported that for 85% of the queries the users request only the first result screen (C. Silverstein 1999). Therefore, *any improvement in the search technology should aim to increase the precision of the results of the first result screen.* Some of the reasons for low precision of search results in the first screen have been identified as follows:

- The *context* of the query is not conveyed through the keywords that form the query. This might either result from the lack of specificity of the query or due to ambiguity in the words forming the query. In either case, the results of search would contain irrelevant results resulting in decreased precision.
- On an average about 550 million searches are performed daily (Roush 2004). There exists a strong interest among commercial websites to obtain top rankings in popular search engines. To this end, some web authors try to manipulate their placement in search engine rankings. This process is referred to as search engine *spamming*. This leads to some irrelevant results among the top ranked result documents leading to decreased precision.
- The presence of documents that are relevant but of *poor authorship* quality among the search results also leads to decreased precision. In traditional Information Retrieval (IR) systems, all content can be assumed to be authoritative and of high quality. However due to the democratic nature of the web, poor quality documents might find a place in the result set and thus decrease the quality of the web search.

2.4 Facing the challenges

2.4.1 Tackling poor authorship quality and spamming

The page-rank algorithm, pioneered by *Google,* has been successful in tackling the problem of poor authorship quality. The algorithm establishes a notion of authority by considering the number of incoming links to a document (L. Page 1998). The more the number of nodes with high visit rates leading to a document, the higher is its page-rank. Tackling search engine spamming is still a challenge for any search engine and is also an active area of research in the academia.

2.4.2 Understanding query context

A review of the various ways to take account of context in web search can be found here (Lawrence 2000). Understanding the context of the user request is a difficult problem for the search engines. The search engines mainly depend on the words in the query to understand the information need of the user. It is difficult for an inexperienced user to provide sufficiently detailed query to elicit the desired response from the search engines. In an attempt to understand the context of a query, the search engines can, potentially, request the user for more information regarding the context, before it compiles and ranks the search results. However, the economics of the search engine market does not allow the search engines to spend a lot of time to process every single query in an attempt to answer them to the complete satisfaction of the user. Nevertheless, the need for understanding the context of the query is increasingly felt by the players in the search engine market. Various search engines have adopted different approaches for tackling this problem. Some of these approaches are discussed in the remainder of this section.

2.4.2.1 Approaches of various search engines to understand the query context

- The *vivisimo* search engine clusters the search results under different topics so that the user can choose the cluster he/she would be interested in, and find results with more precision.
- The *mooter* (http://www.mooter.com) search engine also clusters the results in different categories like the *vivisimo*. In addition, *mooter* re-ranks its results based on the categories chosen by the user. In other words, the search engine learns the context of the information need online based on the user actions and provides results with increased precision.
- The search engine *eurekster* (http://www.eurekster.com/) tries to personalize search using the idea of social networks. The user identifies himself/herself as a part of a social group. The search algorithm takes into account the choices of the social group, and uses this information for delivering better search results for the user.
- *Altavista* uses geotracking technology to detect their visitors IP addresses and guess their geographical location, which might provide some contextual clue about the web search.
- *Microsoft* is working on a search technology for its operating system *Longhorn* that integrates and personalizes search across Microsoft software like Outlook and Office, as well as the web.

2.4.2.2 Other approaches to understand contextual information

There are web directories (for example, Yahoo directories) where web pages belonging to a specific category are listed separately (http://dir.yahoo.com/). A search can be conducted in these pre-defined categories, thus increasing the chances of finding relevant documents. However, in many cases, web pages are assigned to these categories manually, thus reducing their reach considerably.

There are category-specific search engines that specialize in indexing sites belonging to a specific category. For example Google news (http://news.google.com/) allows searches on continuously updated news items. In similar spirit, there are country specific sites, shopping specific sites, multi-media search engines, and other category-specific search engines available in the web. Note that it takes a lot of time and effort to create a category-specific search engine. Only when there is a very large community of users interested in a specific topic, there is an incentive for the creation of a category-specific search engine. McCallum *et.al.* (A. McCallum 1999) recently reported a method for automating the building and maintenance of a domain-specific search engine using reinforcement learning and text classification.

Agichtein *et.al.* (E. Agichtein 2001) proposed a method for learning query modifications for improving category specific searches. The user first selects a category for the search query and this information is used to learn category specific query modification. This method is used in the *tritus* search engine.

It should be noted that in most of the above cases, the *server* (search engine) analyzes information regarding the context to improve the search results. The *client* machine (where the web browser is hosted) possibly has more information regarding the context

of the search. In all the schemes mentioned above, the client machine does not play an active role in using the contextual information. Its role is restricted to providing information about the context (under some schemes only), and displaying the search results.

2.5 Thesis roadmap

The aim of the thesis is to explore the utility of processing the context-related information in the client machines to improve search results. The use of contextual information for improving web search results involves the following:

- Defining and representing the contextual information. (discussed in Chapter 3)
- Developing algorithms to use the contextual information to process the search results. (discussed in Chapter 3)
- Developing software application that incorporates the algorithms and that provides a user interface. (discussed in Chapter 4)
- Conducting experiments to analyze the performance of the techniques developed. (discussed in Chapter 5)

3 Context representation and re-ranking algorithms

This chapter deals with the representation of context and the use of this representation to design algorithms to improve the search process. Possible ways of using the contextual information to improve search like query modification and re-ranking the search results are discussed. Algorithms for re-ranking the search results are then presented.

3.1 Context of a search

3.1.1 Categories of search

The context of search is closely related to the intent of a search, which can be classified into the following categories:

- Search on topics of immediate relevance:

  The user is searching for extremely specific information that is of immediate relevance. For example, consider searching the web for finding the location of a particular restaurant or a shopping mall. In this case, though the information sought is extremely specific, the period over which the information is relevant to the user is quite short. Modeling the context is very difficult in this case.

- Search on specific topics of sustained interest:

  Quite often, the users have specific interests that are sustained over considerable periods of time. For example, a student attending a course will be interested in topics of the course for the duration of the semester, and a

graduate student will be interested in topics related to his/her field of research over extended periods of time. The student and the researcher often perform searches in order to obtain information relevant to their topics of sustained interest. In this category of searching, the topics of interest naturally define the context of the search.

The category of searching on specific topics of sustained interest is the focus of this thesis.

3.1.2 Context definition

Even though two users may share a same topic of interest (for example, programming) their specific interests in the topic might be different (for example, Java and C). Indeed, the context should be modeled to reflect only the specific portions of a topic that the user is interested in.

If the users are interested in specific topics for sufficiently long times, they presumably have documents on those topics. For example, in the case of a student taking a class, these documents could be the electronic class notes. These documents could then be used to define the context for the searches performed to obtain information on these topics of sustained interest of the user. Therefore, *the context is modeled as a collection of documents, one set for each topic of interest of the user.* Since the users themselves provide the representative documents that form the context (will be hereafter referred to as contextual documents), the documents are assumed to be authoritative and to be of good quality.

3.1.3 Vector Space Model for context representation

Having defined the context as a collection of documents, a way to mathematically represent the context is considered next. The Vector Space Model (VSM) (G. Salton 1975), commonly used in Information Retrieval is a simple and proven way to model collection of documents. In this thesis VSM is used in order to represent the contextual documents.

In VSM, documents are represented as vectors in a multidimensional Euclidean space. Each axis in the space corresponds to an index term, typically a word of the document. The intuition behind this approach is that documents representing similar concepts will have similar distribution of content bearing index terms[4]. The coordinate of a document along a certain axis (say representing the index term *t)* is decided by the following quantities:

3.1.3.1 Term Frequency (TF)

TF reflects the number of times the index term *t* occurs in the document in question. TF is a *normalized* quantity in order to account for differences in the length of different documents. Otherwise the longer the document the more will be the TF for the index terms, which is not semantically meaningful. One way of normalization would be to use the sum of the total number of occurrences of all the index terms in the document:

$$TF(d,t) = \frac{n_d(t)}{\sum_\sigma n_d(\sigma)}$$

[4] An index term could either be functional or content bearing. Functional words (for example, prepositions) are those that do not carry any information about the content of the document.

where $n_d(t)$ represents the number of times the index term t occurs in document d and the summation is performed over all the index terms present in d.

3.1.3.2 Inverse Document Frequency (IDF)

Parsing a document and extracting terms is going to give both content-bearing and functional index terms. Not all index terms should have the same weight and a distinction has to be made between the content-bearing words and the functional words. The measure, IDF, is based on the observation that the index terms occurring in many documents are possibly functional in nature. It seeks to assign less weight to those index terms that occur in many documents. Like TF, there are many ways of defining this measure. A sample definition is given below:

$$IDF(t) = \log(\frac{1+|D|}{|D_t|})$$

where D is the set of all documents and D_t is the set of all documents in which the index term t occurs. In the above expression, $|D|$ is the cardinality of the set D.

3.1.3.3 Combining TF and IDF

In VSM representation of a document, each axis in the vector space corresponds to an index term. The coordinate of document along a particular index axis is obtained by multiplying TF and IDF for that index term. If d_t denotes the coordinate of a document d, along the axis representing the index term t, then,

$$d_t = TF(d,t)*IDF(t)$$

It should be noted that it is not necessary to have axes corresponding to all index terms. It would be beneficial to have co-ordinate axes corresponding to only the content-bearing

index terms. Section 4.3.4 discusses specific procedures that were followed in order to screen documents and identify the content-bearing index terms.

3.1.3.4 Query representation in VSM

It is of interest to discuss the VSM representation of a search query. The Information Retrieval (IR) systems using the VSM differ in their handling of the search query. Some IR systems treat a search query as a document where as others do not. There is no established advantage for one representation over the others for general cases. The IR systems that treat the queries in the same way as documents form a representation for the search query in the same multidimensional Euclidean space as the documents using TF and IDF as described above.

3.1.3.5 Measuring similarity (*sim*) between documents

The similarity (*sim*) between two documents d and q (or between a document and a query) can be measured by computing the Euclidean distance between them. Another measure is to use the Cosine Similarity metric, which computes the cosine of the angle between the two vectors:

$$Sim(d,q) = \frac{\sum_t (d_\sigma * q_\sigma)}{\sum_t d_\sigma^2 * \sum_t q_\sigma^2}$$

where the summation is performed over all the index terms. In this thesis, the cosine similarity metric is used as a measure of similarity between documents.

The VSM based IR systems work extremely well in practice, and therefore this model has been used to represent documents in this investigation.

3.2 Using contextual documents in the search process

Section 3.1 discussed the definition of the context as a set of relevant documents on topics of interest to the user (referred to as the contextual documents) and the VSM representation of the documents. This section discusses the algorithms developed that use these contextual documents for improving the quality of web searches.

A common search process consists of the following

- Submitting a query to a search engine
- Getting the response from the search engine
- Iterating the above steps till the information need is satisfied

The context information can be used in different ways to improve the quality of the search results performed by the above conventional procedure. They include:

i. Query modification

ii. Search result re-ranking

Each of these methods is discussed in Sections 3.2.1 and 3.2.2.

3.2.1 Query modification

A technique commonly known as query modification adds additional words that reflect the context of the search to the query, resulting in increased specificity of the query. For example, suppose a student taking an introductory programming class in Java is performing a search using the query "Event model", and suppose that electronic course notes on Java programming are used as contextual documents for the search. The query can be re-formulated as "Java Event model". Naturally, this re-formulation of the query will lead to more relevant results than the case when query is just "Event model".

However, query modification method of using the context information should be implemented carefully since even a single incorrectly learned term would decrease the precision of the results drastically (M. Mitra 1998).

3.2.2 Search result re-ranking

Another method to use the context information to improve the quality of search results is to re-rank the search results obtained by submitting either the original user query or the modified user query to a search engine. As discussed previously, an improvement in the quality of the search is to increase the number of relevant results among the top search results. The aim of the re-ranking algorithms, therefore, will be to ensure that the *precision(n)* is as high as possible, where *n* is the number of search result documents that the user is interested in viewing in order to find the document that satisfies his information need. It is assumed that the query (either the original or the modified representation) is specific enough that the search engines are able to return all the relevant documents indexed by them. Therefore, the re-ranking procedure will involve looking at enough search results, re-ranking them using the contextual documents, and to present the modified results.

In this thesis, potential improvements in the quality of search results that could be achieved by using the contextual documents to re-rank the search results are investigated. For this purpose, algorithms were developed that re-rank the search results by considering the relevance of the search result documents to the context of the search (reflected by the contextual documents).

3.2.3 Algorithms for measuring the relevance of a document to the context

Re-ranking algorithms developed involve measuring the similarity of a document provided by the normal search with the contextual documents. The following sections detail two algorithms developed for computing the relevance of each search result document to the contextual documents. Re-ranking was performed based on the relative relevance of search result documents. Nomenclature relevant to the description of the algorithms is provided in Table 1.

Notation	Description
n	number of results that the user wishes to see.
$\boldsymbol{Q}$	Query (Either as entered by the user or the modified query).
R	search result document set obtained by submitting the search query (or a modified search query) to a search engine(s).
C	contextual document set.
$\boldsymbol{D}$	a document in set R

Table 1: Notation used for describing the re-ranking algorithms

3.2.3.1 Nearest documents algorithm

This algorithm identifies the set of contextual documents most closely related to a search result document, and then estimates the similarity of the search result document to this set of nearest contextual documents. The various steps involved are as follows:

Step1: Calculate the similarities (*sim*) between $\boldsymbol{d}$ and every document in C (see Section 3.1.3.5 for a discussion of the computation of *sim* between documents).

Step 2: Form a set of k (an arbitrary number) documents in C that are nearest to $\boldsymbol{d}$, using the similarity values. Denote this set of k documents by C_{kd}. (naturally, a subset of C).

Step 3: Compute the average similarity of $\boldsymbol{d}$ to these k documents of C_{kd} (denoted by S_{kd}) using the similarity values between $\boldsymbol{d}$ and each of the k documents.

Step 4: Repeat the above steps for each of the document $\boldsymbol{d}$ in R and compute the quantity S_{kd} that reflects the (average) similarity between the document $\boldsymbol{d}$ and k nearest contextual documents.

Step 5: Re-order the documents in descending order of S_{kd}.

Figure 3 schematically shows the procedure of forming the subsets C_{kd} for two documents ($\boldsymbol{d_1}$ and $\boldsymbol{d_2}$) of R, for the case of a two dimensional document space with index terms t_1 and t_2, and for a k value of 3. Circles represent the contextual documents, and the documents in R are represented by squares. The circles define the boundary for the inclusion of the contextual document in the set C_{kd}.

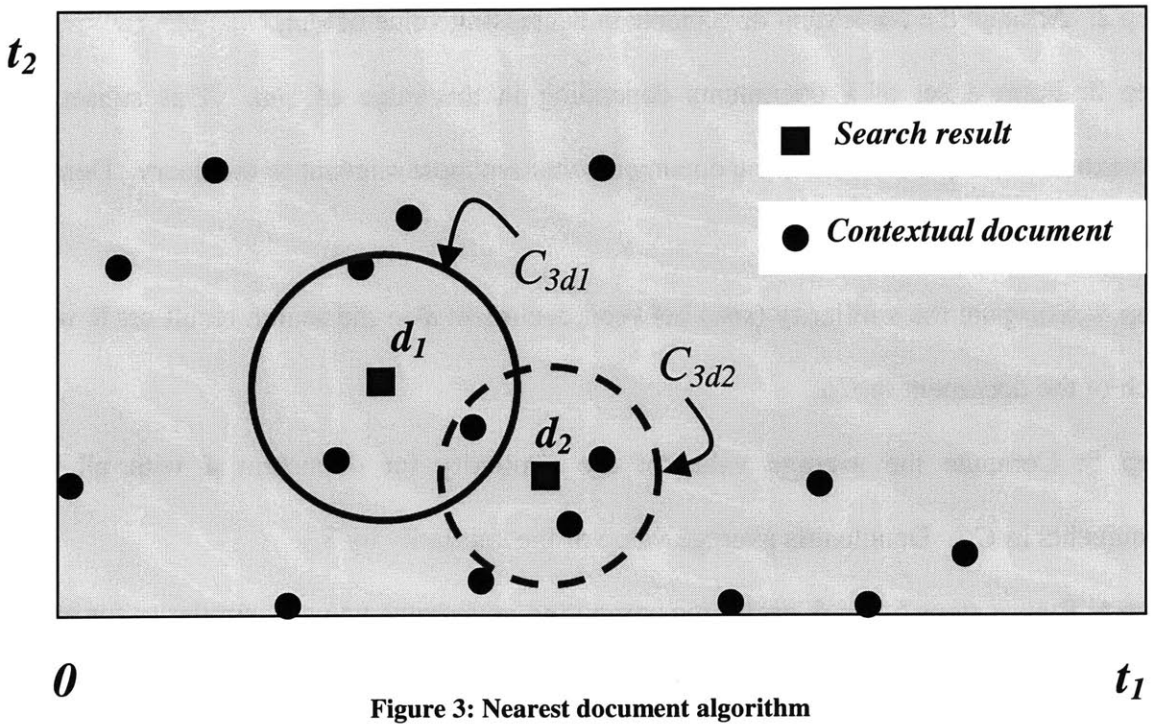

Figure 3: Nearest document algorithm

3.2.3.2 Query mapping algorithm

The second algorithm maps the search query to the vector space of the contextual documents, identifies the subset of the contextual documents that relate to the query, and then computes the similarity of a search result document to this subset of contextual documents. The important difference between this algorithm and the nearest documents algorithm is the selection of the subset of the contextual documents for performing the similarity calculation. The various steps involved with this approach are as follows:

Step 1: Compute the similarity (*sim*) between the query q with each of the contextual document in C. It is to be noted that query is considered as a document for the purpose of calculating the similarity. See Section 3.1.3.4 for a discussion of representing queries using the VSM.

Step 2: Arrange the contextual documents in decreasing value of *sim*,

Step 3: Form a set of k documents depending in the value of *sim*. This subset of contextual documents contains the documents that are most relevant to the query. Denote this subset by C_{qk}.

Step 4: Compute the similarity *(sim)* between document ***d*** in the search result set R with each of the document in C_{qk}.

Step 5: Compute the average value of the similarity for document ***d*** with all the documents in C_{qk}. Denote this average value of the similarity by S_{dk}.

Step 6: Repeat steps 5 and 6, each time computing an average value of similarity for each of the document ***d*** in R. S_{dk} reflects quantitatively the average relevance of a document ***d*** with the subset of contextual documents that is most relevant to the query q.

Step 6: Re-rank the documents ***d*** based on the values of S_{dk}.

Figure 4 shows schematically the selection of documents that forms the set C_{qk} in a two-dimensional vector space with index terms t_1 and t_2 and for a k value of 3. The query is indicated by *, the contextual documents are indicated by circles and a search result document is indicated by squares. The circle defines the boundary of the inclusion of the contextual document in C_{qk}.

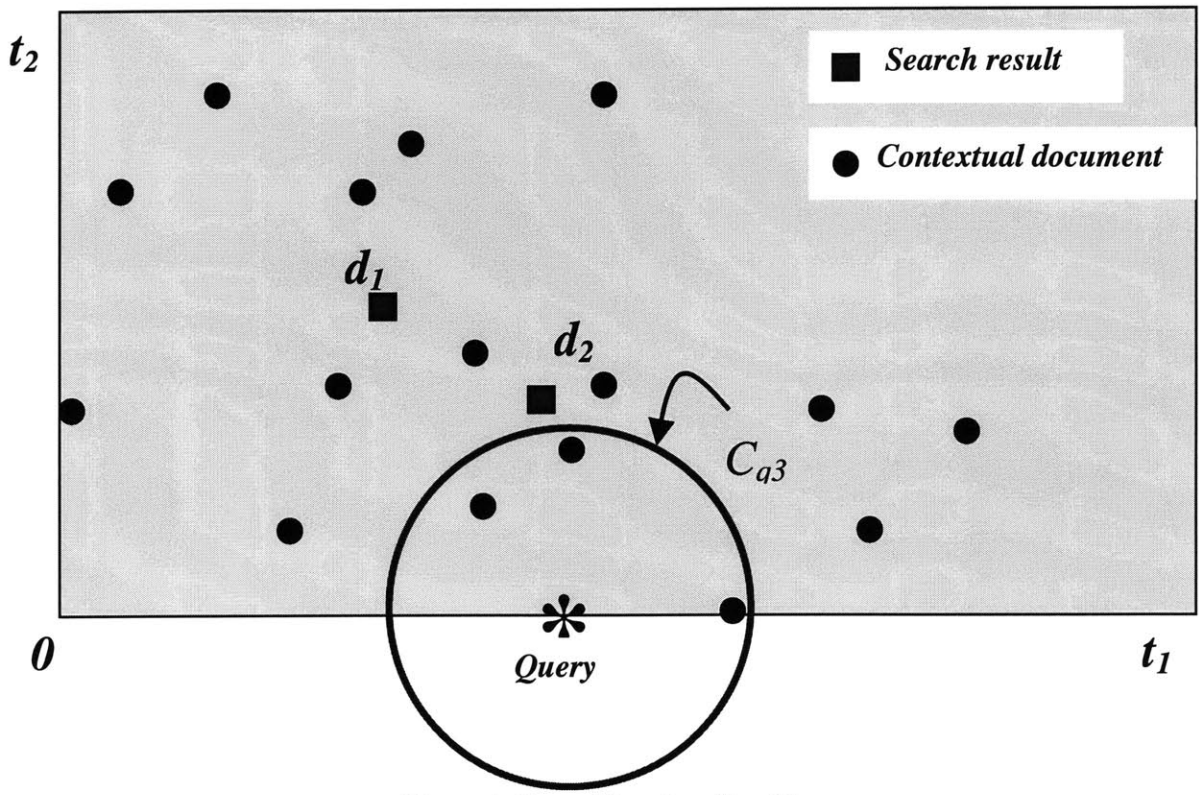

Figure 4: Query Mapping Algorithm

3.3 Chapter summary

In this chapter, the context of a search was defined to be a set of exemplary documents on a topic of sustained interest of the user. Vector space model of the information retrieval systems was used to mathematically represent documents. Two context-based re-ranking algorithms were developed in order to improve the performance of the web search by using contextual information. The two algorithms involved measuring the similarity between each document of the search result and a subset of the contextual documents in the vector space.

Chapter 4 discusses various aspects of implementation of the developed techniques in a software application.

4 Software design and implementation

This chapter details the design and implementation of a software application that implements the idea of using the context information (defined in the form of relevant documents by the user) in order to improve the quality of the web searches. In Chapter 3, algorithms to re-rank the search results by comparing the similarity of the search result documents with the contextual documents were developed. In order to quantify the similarity between documents, the use of Vector Space Model (VSM) of representation of documents was suggested. While the VSM is conceptually simple, it has to be implemented carefully to handle certain special conditions.

4.1 Software implementation

The software application that incorporates the algorithms developed was built in stages:

i. In the first stage, a software infrastructure for constructing VSM for representing the contextual documents was built and refined.

ii. In the next stage, the re-ranking algorithms developed were implemented, tested, and refined.

iii. In the final stage, a software application that incorporates the above code and provides a UI for functionality associated with the search process was created.

The details of each of the stages are provided below.

4.2 Choosing the software platform

An initial assessment of the software requirements suggested that an object oriented language like Java or C# would be most suited for the project. Not much platform support is required for the first two stages, namely, building the Vector Space Model and implementing the search algorithm. However, for the last stage a platform that provides support for creating and managing UI would be extremely useful. Considering this, Microsoft .NET was chosen as the platform of implementation. C# was the implementation language.

4.3 Building a vector space model

4.3.1 File types supported

It is to be noted that the contextual documents can be in many formats, including, PDF, text, and HTML formats. An overwhelming number of documents on the web are text files (mainly .html and .htm files). Also, a large number of educational and research documents in the web are PDF files. Considering that the search technique proposed is expected to be of great benefit to researchers and students, the application was designed to support both PDF and HTML file types.

Almost all programming languages have support to process text files. However, there is no standard support for processing PDF files. All files in PDF format were first converted to files in HTML format before being processed. A utility called "pdftohtml" that converts PDF files to HTML or XML files was used (Kruk 2003). It should be noted that the user might want to add a relevant document to the set of contextual documents anytime. Hence, it should be possible to convert PDF files to HTML format

programmatically. The .NET platform provides a way to open an external application from a .NET application using the `System.Diagnostics` Class. The following C# code snippet illustrates a way to open the "pdftohtml" utility from a .NET application and use it to convert a file in PDF format to a file in HTML format programmatically:

```
System.Diagnostics.Process.Start(@"C:\tmp\surfaceTension\pdftoHtml.exe"
,@"C:\tmp\surfaceTension\eastoe02.pdf  C:\tmp\surfaceTension\eastoe")
```

4.3.2 Processing a document

It was mentioned in Section 3.1.3 that, in the VSM representation of documents, content-bearing index terms form the axis of the vector space and the corresponding coordinates can be calculated using the IF and IDF of the index terms. This section discusses the various steps involved in processing a HTML document in order to identify the content-bearing index terms and constructing the Vector Space Model representation of the documents.

4.3.2.1 Regular expressions

For many tasks involving text processing, regular expressions are very useful. Regular expressions are useful for extracting certain pattern of characters from a text. Regular expressions originate in Unix and were fully developed by Perl scripting language. A regular expression is a pattern used to match against a string. The regular expression to be used depends on the task on hand. For example the regular expression `"\be\S*"` matches all the words beginning with letter "e". More information on regular expressions can be found in (J. Price 2002).

The `System.Text.RegularExpressions` namespace contains the classes necessary for using regular expressions in C#. Example usage will be seen in the sections below wherever appropriate.

4.3.2.2 Stripping HTML tags

Some search engines pay special attention to the HTML tags in a document as it might contain useful information about the content. However, in the present work, this was not considered. The first step in processing documents in HTML format was to remove the HTML tags. The regular expression `"<(.|\n)*?>"` was used to identify HTML tags. The following C# code snippet gives an example of the use of the above regular expression to remove HTML tags:

```
Regex.Replace(Content,pattern,patternToReplace);
```

`Regex` is a class in the `System.Text.RegularExpression` namespace. `pattern` is the variable identifying the regular expression `"<(.|\n)*?>"`. `PatternToReplace` is the string that replaces pattern wherever it is found and would be a white space in this case.

4.3.2.3 Removing "stop words"

Many languages have function words and connectives like prepositions and conjunctions that appear in large numbers in documents. They are of little use for the purpose of identifying the context, and hence should not be considered as index terms. Such words are called "stop words". Examples from English include "a", "an", "the", etc. The steps involved in removing the stop words from the un-tagged HTML documents are as follows:

i. The un-tagged HTML document was broken into words. The regular expression `"\b\S*\b"` was used for this purpose.

ii. The unique words from this document were then identified.

iii. The words in the stop word list published by the defense virtual library (http://dvl.dtic.mil) were then removed from the list of unique words. It should be noted that it might be useful to store the offset at which the "stop words" occur. This would help in the case where a phrase containing a stop word is part of the search query (example search query, "cost to go function"). However in the current case, the position of the stop words were not stored for simplification purposes. This means phrases with one or more stop words would be treated as individual words.

4.3.2.4 Word stemming

Stemming is a process by which words are reduced to their stem. For example, the words "programs" and "program" are identical for the purpose of searching and hence should be treated as the same word. Stemming methods use mainly morphological analysis to identify the variants of the same stem. A widely used stemming algorithm is the Porter's algorithm (Porter 1980). One should also note that stemming might sometimes result in decreased precision when words with different stems are mapped to the same stem. For example, the Porter's algorithm would map the words "University" and "Universal" to "univers". Abbreviations are particularly prone to being stemmed incorrectly. Hence the risks involved in stemming should be weighed carefully especially for web searches. The TF and IDF that define the coordinates of the Vector Space Model are found, not for the

index terms themselves, but for their stems. A publicly available implementation of Porter's algorithm (http://www.tartarus.org/~martin/PorterStemmer) was used.

4.3.3 TF/IDF calculation

After pre-processing a document and identifying the unique, stemmed index terms, the next step is to find out the term frequency (TF) and the inverse document frequency (IDF) of the index terms. *Most of the Information Retrieval (IR) Systems take a document-term frequency matrix and find out a term-document matrix to ensure fast look up for queries.* This is called inverted indexing (Korfhage 1997). For the current case, an inverted index was not required since a bulk of the algorithm involves comparing a search result document with a contextual document. The document term frequency matrix was constructed for every contextual document. IDF was then computed for the index terms from the TF information of the documents. Note that it is not necessary to compute the TF and the IDF for the index terms every time the application needs to perform a re-rank. Only when a contextual document is added or removed is it necessary to re-compute the TF-IDF of the index terms. The method adopted for identifying and retaining the content bearing words is described next.

4.3.4 Identifying the content-bearing index terms

The number of index terms increases with the number of contextual documents. Only a fraction of the index terms are content bearing, and hence the Vector Space of the contextual documents should only contain co-ordinate axis representing these content bearing index terms. One way to identify the content bearing index terms is to first compute the TF of the index terms after treating the entire contextual document set as a

single document. When the TF is multiplied by the IDF of the index term, a measure of the index term's relevance to the contextual documents can be obtained. .

Since the contextual documents pertain to a specific topic (context) the index terms that are relevant to the context would appear in many documents. Consider the typical weighting scheme for IDF (see Section 3.1.3.2):

$$IDF(t) = \log(\frac{1+|D|}{|D_t|})$$

Note that the IDF would penalize an index term that occurs in all the contextual documents. Since very relevant and content bearing words may also occur in all the contextual documents, IDF in the above form will not directly help in the identification of the content bearing index terms. Therefore, in order to select the content bearing index terms, the following modified formula of the weighting function was used:

$$w(t) = TF(t) + TF(t) \times \log(\frac{1+|D|}{|D_t|})$$

The relative magnitude of the *w(t)* between different index terms reflect the relative content-bearing property of the index terms. After assigning the above weight to the index terms, it was noticed (for many examples considered) that the top 10% of the index terms are typically content bearing. Accordingly, the index terms were arranged in descending order based on the weight *w(t)* and the top 10% of the index terms were retained as context bearing and the others were discarded. Note that the coordinate axes in the VSM representation of documents would represent these content bearing index terms.

The coordinate of any document *d* in this vector space is given by the following expression:

$$w(d,t) = TF(d,t) + TF(d,t) \times \log(\frac{1+|D|}{|D_t|})$$

4.4 Implementing the re-ranking algorithms

The above section dealt with the implementation of the VSM for representing the documents. This section describes the implementation of the re-ranking algorithms developed.

4.4.1 Getting the search result set

The search algorithms first obtain a set of possibly relevant results by querying popular search engines with the search query entered by the user. In this investigation, the Google search engine was used for this purpose. Google provides an API (Google) that provides programmatic access to a Google web service. The web service enables search leveraging the Google search mechanism and provides results that are identical to searching through the web-based interface. The API also provides all the information about a search result like its URL and title. This API was used to get information regarding the URLs of the top results of the search query using Google. The number of search results that would be processed by the context-based re-ranking algorithm can be varied. Web requests were then programmatically sent to the URLs of the search results and individual search result documents were downloaded. The `WebRequest` class in the `System.Net` `namespace` was used in order to communicate a web request and receive a text stream. The Google API also exposes functionality for getting the cached versions of the search results from Google database. This service was also used at times as an alternative to directly downloading the search result.

4.4.2 Preprocessing the search result

Once a search result document was obtained using one of the methods described above, it was pre-processed to remove HTML tags and stop words (see Sections 4.3.2.2 and 4.3.2.3). The preprocessed search result document was then represented in the Vector Space of the contextual documents.

4.4.3 Vector class

A `Vector Class` was written to represent a document in the Vector Space. This Class has two main methods:

1. An integer representing the number of dimensions of the Vector Space
2. An array of double precision numbers representing the weight of the Vector along each of the dimensions

There are also `get` and `set` methods for retrieving and setting the weights of the Vector along each of the co-ordinate axes.

The code for the `vector` class is given below

```
public class Vector
      {
            int numDimension;
            double[] weights;
            public Vector( int numDimension)
            {

                  this.numDimension = numDimension;
                  weights = new double[numDimension];
            }

            public void setWeight(int dimension, double weight)
            {
                  weights[dimension] = weight;
            }

            public int getNumDimension()
            {
                  return numDimension;
            }
```

```
        public double[] getVector()
        {
              return weights;
        }

        public void setWeight(double[] weights)
        {
              for (int i = 0; i < weights.Length; i++)
              {
                    this.weights[i] = weights[i];
              }
        }
        public double getWeight(int dimension)
        {
              return weights[dimension];
        }

}
```

4.4.4 Representing a document as a Vector

A *method* was written to return a Vector Object representing a document in the Vector Space of the contextual documents. The method signature is given below:

```
public static Vector calcDocVect(String currentDocument , bool useIDF)
```

The variable `currentDocument` represents the pre-processed search result document. Individual index terms are extracted and the document term frequency is calculated for the content bearing index terms retained. The term frequencies were computed by normalizing the raw term frequencies by the length of pre-processed search result document. A Vector object is created after suitably weighting the TF with IDF.

4.4.5 Calculating the similarity between a document and the context

Since both the algorithms developed involves calculating similarity between the search result document and the contextual documents, a method was written to compute the

similarity of a search result document and the contextual documents. The method signature is given below

```
calculateContextSimilarity(curDocumentVect);
```

Here `curDocumentVect` represents the `Vector` object representing the search result document in the vector space of the contextual documents.

The first step in computing the similarity between a search result document vector and the contextual documents was to create vector objects to represent all the contextual documents. At this point, implementing the two search algorithms developed (see Sections 3.2.3) was straight forward as it reduces to repeatedly computing the similarity using the Cosine Similarity Metric between a context document vector and a search document vector (see Section 3.1.3.5). The search result documents were ranked based on the decreasing order of this similarity measure.

The documents for the search results were initially downloaded sequentially. It was noticed that this process comprised a major portion of the time taken by the re-ranking algorithm. Note that it is not necessary for *all* the result documents to be downloaded for the context-based re-ranking algorithm to operate on the search result documents. The application was written such that each document is downloaded, processed, and compared to the contextual documents in different threads in parallel. After all the search result documents were measured for the similarity with the contextual documents, re-ranking was performed. This procedure was used to multi-thread the software application.

4.5 *Integrated software application*

A software application incorporating the developed algorithms and providing a UI for associated functionality was built. The software is a Windows Forms application built using Visual C#. This section explains the design and implementation of the software.

4.5.1 Features of the software application

Some of the features that were incorporated in the software application developed are listed below:

- Add documents to the set of contextual documents
- Delete documents from the set of contextual documents
- Browse the web
- Search using a standard search engine – Google was chosen for this task
- Search using the context-based re-ranking algorithms

Browsing almost always accompanies a search process and hence it is important for the software to have the ability to browse. Building software to allow browsing would take a lot of time and effort. Interestingly, there is a web browser control that can embed in a Windows form application. The control exposes functionality that allows one to programmatically perform functions like navigating to a URL, refreshing the currently displayed web page, stopping a request for a web page, and all the standard functions that one expects in a web browser. The following functionality were implemented for browsing applications:

1. Specify a URL to navigate
2. Navigate to the home page

3. Go back to the last visited page
4. Go Forward
5. Stop a current request for a web page
6. Refresh the currently displayed web page

4.5.2 Class design and implementation

The terminology of the Model-View-Controller pattern (Sun) would be used to explain the design of the classes.

The web browser control was enough for providing a view for the application. Hence no separate class was necessary for modeling the view.

A toolbar was provided along with toolbar buttons to allow the user to perform each of the functions listed above. This forms the controller.

A class called `Threader` provides the model for the software. Each `threader` object models a certain number of search results. This class is a multi-threaded version of the search algorithm implementation discussed previously. In addition to the data members needed for the search algorithm, there are other variables needed for multi-threading. For example, there are variables for, tracking the search result documents assigned to the thread, storing their URLs, the similarity comparison algorithm to be used, storing the similarity of the search result documents to the context and so on. There are appropriate `get` and `set` methods for these variables. For each search result document that it tracks, the `threader` object downloads, processes and computes the similarity of the search result document with the contextual document set and stores the same. The `threader` class uses the `Vector` class for computing the similarity.

The following figure shows the relationship between the main classes

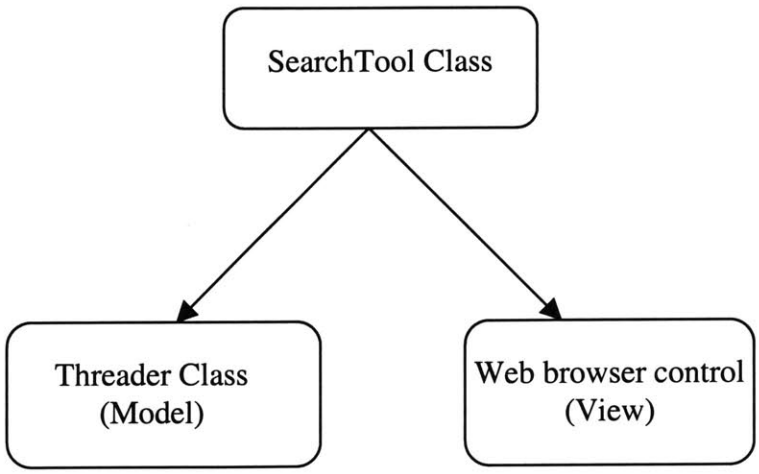

Figure 5: Relationship between the main classes

Figure 6: Screenshot of the UI

The event handler for the "search button" generates the appropriate number of threader objects and launches them in separate threads. It then waits for all the threader objects to finish their work. Once all the threader objects finish computing the similarity of the search results that they track, the event handler re-ranks the search result document set. The following code snippet, from the event handler for the "search button" shows how similarity is computed for the search result documents using the *"query mapping"* method.

```
// Search Query
String searchString = textBoxSearch.Text;
// Gets the vector object representing the search query
Vector queryVector = Threader.calcDocVect(searchString,false);
// number of documents to be displayed
int topNumDocs = 10;
// Calculate the contextual documents relevant to the search query
int[] docIndex =
Threader.calculateDocumentSimilarity(queryVector,topNumDocs);
// Set the values for variables of threader class
Threader.setR(r);
Threader.setTitle(title);
Threader.setURL(URL);
Threader.setNumDocsToProcess(threadSize);
Threader.setSimilarity(similarity);
Threader.setSummary(summary);
Threader.setRankingAlgorithm("QueryProximity");
Threader.setTopDocsIndex(docIndex);
//Create threader objects and compute similarity
for (int i = 0; i < numOfDocsConsidered/threadSize; i++)
```

```
{
workerThreaders[i] = new Threader(timeBefore, i*threadSize);
testThreads[i] = new Thread(new
ThreadStart(workerThreaders[i].computeSimilarity));
testThreads[i].Start();
}
// Wait for all threader objects to complete
for (int i = 0; i < numOfDocsConsidered/threadSize; i++)
{
testThreads[i].Join();
}
```

Once the search result documents are re-ranked, the top results are displayed using a HTML page generated on the fly that displays the title of the search result and the URL. The HTML page is generated on the fly using a template HTML page that has the layout correctly specified and with appropriate blank spots for the URL and title of the results.

The event handler for the refresh, stop, go, back, forward and home buttons use the appropriate methods of the web browser control.

The event handler for the "`try Google`" button parses the search query entered by the user and generates appropriate web request and displays the response in the web browser control.

The event handler for the "`add to context`" button creates an instance of a class called "`addToContextDialog`". This class is derived from the "Form" class in the `System.Windows.Form` namespace. Hence this object appears as a separate window as soon as it is created. The user can either choose to add a file from the local machine or can download a resource from a URL. The resource identified by the URL should either

be in the "pdf" format or in "HTML" format. If the resource is in "pdf" format it is converted to "HTML" format as soon as it is downloaded, by launching an external process as discussed in the section on context representation. Screenshots of this process are shown in Figure 7.

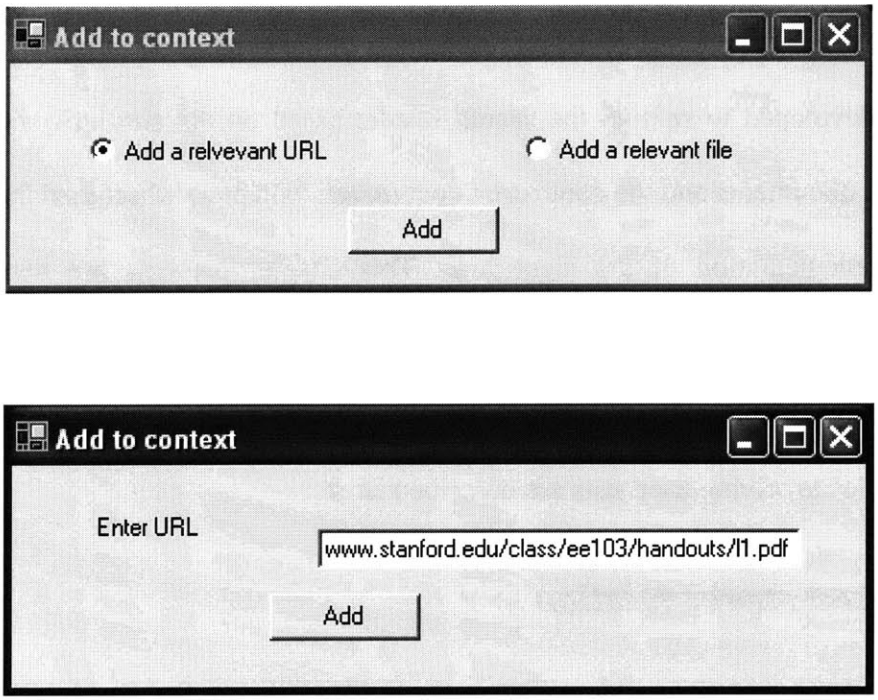

Figure 7: Adding a document to the context document set

The event-handler for the `"Del doc from context"` button throws a `"file chooser"` that helps the user to select a file to be deleted from the context document set.

The context representation changes as soon as a document is added to it or as soon as a document is deleted. Hence the context representation is re-learnt every time a document is added or deleted from the context.

Chapter 5 describes some of the test searches that were performed using the software application and the performance of the context-based re-ranking algorithms developed in improving the quality of the searches.

5 Results and discussion

In the previous chapters, the thesis idea of using the contextual information in the client machines in order to improve the quality of web searches was discussed in detail. Chapter 3 discussed the VSM based representation of the contextual information, and the algorithms developed to re-rank the search results based on the similarity between the search result documents and the contextual documents. Chapter 4 discussed the details of software implementation of the algorithms. This chapter presents the results of the experiments performed in order to test the effectiveness of the algorithms for providing better search results. Results related to the quality of context representation are presented first. Examples involving user tests are described later.

5.1 Context representation

The results of the context representation analysis (described in Section 4.3.3) performed for a set of contextual documents are presented in this section. Consider an example where the documents from course notes of an M.I.T class (Introduction to computers and engineering problem solving) are selected to identify the context. This class uses the Java programming language to introduce object oriented programming concepts. The course notes were downloaded from the OpenCourseWare[5] site for the course.

Index terms and the content-bearing terms for the contextual documents were identified after performing a screening for stop-words, and word stemming. Some index terms[6] are listed in Table and are arranged in the descending order of term frequency:

[5] An MIT initiative to provide free and open course materials (http://ocw.mit.edu)
[6] Note that the index terms are stemmed to their roots

Index term	Term frequency
Public	711
Method	478
New	439
Int	429
Doubl	389
Void	313
Java	257
Object	236
Privat	228
Static	213
Quot	195
String	186
Thi	183
Implement	178
Valu	160
Call	152
Class	147

Table 2: Term frequency of index terms

In order to identify the content bearing terms, weights *w(t)* for these index terms were calculated (see Section 4.3.4) using the TF and IDF of the index terms. The index terms are arranged in descending order of the weights *w(t)* in Table 2:

Index term	Weight
Public	1492
Method	1003

Index term	Weight
Doubl	928
New	921
Int	900
Privat	544
Java	539
Void	529
Object	495
Quot	465
Static	447
Implement	424
List	390
String	390
Thi	384

Table 3: Content bearing index terms for the selected document set

Naturally, the index terms with high weights as compared to other index terms with lower weights (not listed in the table) form the content bearing terms, and therefore, the axes of the Vector Space representation of the documents.

Some insights gathered in the process of identifying the content bearing index words are worth a mention: It was noticed that many one-letter characters (like A, D, Z etc.) and numbers found a place as index terms. This was mainly a result of inconsistent conversion from the "pdf" to the HTML formats. Consequently, a method was written to filter numbers and all index terms of length less than 3 characters. It was observed that the quality of the index terms improved dramatically with this additional filtering.

5.2 Parameter values for the re-ranking algorithms

The nearest document algorithm has a single parameter to be set, k. Recall that k is the number of the contextual documents closest to a search result document to be considered for computing the similarity of a search result document to the contextual document set.

It was observed that the value of 1 for k provided good performance. In general, the closest document in the contextual document set to a search result document was seen to provide a good estimate of the relevance of the search result document.

In the case of the query mapping approach, it was observed that performance improved with higher k. This is however very query specific and no satisfactory rule could be derived. It should be noted that in some cases not a single contextual document matched the search query and in these cases the algorithm defaults to the nearest document approach.

5.3 Example search results from user tests

There is no standard benchmark for comparing web search results. Currently user tests are the only way to measure the relevance of the search results. To form definite conclusions about the utility of the software, extensive user tests are required on a significant number of users for a considerable period of time. This is beyond the scope of the present work. However, limited user tests were conducted to obtain feedback on the utility of the software. It is expected that the software will be useful in an academic and a research environment. Hence, the software was given to a few researchers at M.I.T to perform web search relating to their research. The results from the user tests are encouraging and serve to validate the key ideas of the thesis. A few example results from

the user tests are given below. The results obtained using the software are compared with the results obtained using the Google search engine. The nearest document approach was used for testing in the examples reported below and the parameter k associated with the nearest document approach was set to a value 1.

5.3.1 Example 1

For this case, the search pertained to a field called dynamic surface tension. The contextual document set consisted of five review journal papers that discuss the various theoretical and experimental advances in the field of 'dynamic surface tension' written by eminent researchers. The review papers were used as contextual documents as they provided a broad overview of the field and covered the concepts that were of interest to the user.

Search Results:

Case 1: Search query: "Ovalbumin Adsorption"

User expectation: Ovalbumin is a protein. The user is interested in the *dynamics* aspect of the ovalbumin adsorption on *water/air interface.*

Comments on the results:

- Google search: Many documents that deal with the adsorption of ovalbumin on *solid surfaces* were present among the top search results. Only a few of the top ten search results were relevant (See Table 4). Moreover, only one among the top ten search result documents addressed the *dynamics* aspect of the adsorption of ovalbumin.

- Context-based re-ranking algorithm: About four documents were found in the top ten results that addressed the *dynamics* of the adsorption of ovalbumin on water/air interfaces. It is to be noted that the re-ranking algorithm retained all the relevant documents that were present in the top ten of Google search result.

Summary of results for this query:

The following table summarizes the results for this query. Here, the top ranked results from both Google and the developed software are ranked for relevance to the search query. *"I"* indicates an irrelevant result and *"R"* indicates a relevant result. Further, the number of stars indicates the degree of relevance. The stronger the relevance of the result the more stars that it gets.

Ranking of search result	Google	Context-based re-ranking
1	I	I
2	I	R*
3	I	R**
4	R*	R***
5	I	R**
6	R***	I
7	I	I
8	R**	R***
9	I	I
10	I	R

Table 4: Comparison of search results for search query "Ovalbumin Adsorption"

Case 2: Search query: "Adsorption of ionic surfactants"

User expectation: There exists a lot of literature on the *non-dynamics* aspect of the adsorption of ionic surfactants. The user was specifically interested in looking for information on the *dynamics* aspect of adsorption of ionic surfactants on water/air interface. Note that, although the cases 1 and 2 described above may appear to be of similar nature, they belong to different set of literatures altogether.

Comments on the results:

- Google Search: A majority of the top ten search results of Google dealt with the *non-dynamics* aspect of the adsorption of ionic surfactants (See Table 5).
- Context-based re-ranking algorithm: The algorithm performed a satisfactory job in bringing out the search results that dealt with the *dynamics* aspect of the adsorption of ionic surfactants on water/air interface, in the top ten results.

Summary of results for this query:

Ranking of search result	Google	Context-based re-ranking
1	I	R
2	I	I
3	I	R**
4	I	I
5	I	I
6	R*	I
7	I	I
8	I	I
9	R**	R**

10	I	R**

Table 5: Comparison of search results for search query "Adsoption of ionic surfactants"

Case 3: Search query: "Space technology applications of dynamic surface tension"

User expectation: The search query is very specific and the user expectation is clearly reflected by the query itself.

Comments:

- Google search: Google seemed to give a lot of weight to the 'space technology' part of the query. The importance associated with the concept of 'dynamic surface tension' and its potential applications in space technology seemed to be less significant in the results that Google provided (see Table 6).
- Context-based re-ranking algorithm: With the context clearly defined by the review articles in the field of dynamic surface tension, the context-based re-ranking algorithm picked up extremely relevant results from the sample chosen for re-ranking. For example, some of the results were as specific as, dealing with the fields of impact of the dynamics of adsorption of surfactants in space technology.

Summary of results for this query:

Ranking of search result	Google	Context-based re-ranking
1	R*	R***
2	R	R*
3	I	R***

Ranking of search result	Google	Context-based re-ranking
4	I	R
5	I	I
6	I	R***
7	I	I
8	I	R*
9	I	R
10	I	I

Table 6: Comparison of search results for search query "Space technology applications of dynamic surface tension"

5.3.2 Example 2

For this case, the search pertained to *"strained Si technology"*, specifically relating to the processing and the physics of device. The contextual document set consisted of few review journal papers and research articles from relevant web sites (http://www.eetimes.com, http://www.ibm.com). These documents discussed the relevant theoretical and experimental advances in the field of processing technology and the physics of transistors built on *"strained Si platform"*.

Example search results

Case 1: Search query: "Device Physics Lectures"

User expectation: Interested in documents describing the device physics of the transistors used in microelectronics.

Comments on the results:

- Google search: Some of the results that appeared in the Google search were irrelevant since they did not address the physics of the

transistors. Many of the results led to the course web sites in different universities and were not useful. However, some of them were useful as they had links to other websites that contained helpful material.

- Context-based re-ranking algorithm: Most of the retrieved documents addressed the pertinent topic of device physics in transistors. However, the documents that had links leading to relevant links were not chosen among the top search results. The top search results included many relevant but short documents.

Summary of results for this query:

Ranking of search result	Context-based re-ranking	Google
1	R**	I
2	I	R*
3	R**	I
4	R**	R***
5	R**	R
6	R**	R***
7	R**	I
8	R**	R*
9	R**	R*
10	R**	I

Table 7: Comparison of search results for search query "Device Physics Lectures"

Case 2: Search query: "MOSFET device fabrication strained si"

User expectation: Get documents related to various processing steps for fabrication of MOSFET on *"strained Si platform"*. This query is very specific and a document can be precisely classified as relevant or irrelevant.

Comments:

- Google search: Google gave very relevant results. Many of the top ranked documents rated relevant were either very relevant themselves or had links to very relevant documents
- Context-based re-ranking algorithm: Most of the top search results were very relevant. Some of the relevant documents appearing in the top results of Google search were missing. However the overall the quality of the results were very good.

Summary of results for this query:

Ranking of search result	Context-based re-ranking	Google
1	R***	R**
2	R**	R**
3	R**	I
4	R**	R**
5	R***	I
6	R*	R**
7	R*	R***
8	I	I
9	R**	R***
10	R*	I

Table 8: Comparison of search results for search query "MOSFET device fabrication strained Si"

5.4 Observations on the performance of the software

The above results provide typical examples of the performance of the software. On the basis of user feedback a summary of the performance of the software is given below:

- The search results seem to be more relevant in general as compared to standard search engines like Google.
- The more the number of contextual documents the better seems to be the quality of the search results. This is because more the number of contextual documents, better is the context representation and hence the better quality of the search results.
- The query mapping method works better for well-defined specific queries. The nearest documents approach works better for queries that are general and not very well defined.
- The quality of the search results depends on the quality of the search query. This is understandable, as only a fixed number of search engine result set documents are examined and if the query is not specific enough this set does not contain relevant documents.
- The re-ranking algorithm favors short documents. The similarity measure normalizes TF-IDF based on document length. Some short documents contain a high proportion of contextual index terms. However, their document length is too short to make them relevant. The re-ranking algorithms do not filter such documents.

- No special consideration is given to documents from authoritative sources. The re-ranking algorithm does not use hyperlink structure analysis and this led to some authoritative documents to be ranked low.
- The time taken for a search using the developed software is much more than the time taken for a search using a standard search engine. This is because the context representation for search result documents is done on the fly. For each query, the search results are first downloaded and then processed to form a context representation and re-ranking is then done using the algorithms developed.

Initial user tests with the software application were reported in this chapter and the user feedback validates the main ideas of the thesis. Chapter 6 summarizes the project and provides directions for future research.

6 Conclusions and future work

6.1 Project summary

The motivation behind the current work is to develop algorithms to improve the search process using contextual information. It is assumed that the client machine (hosting the client software and making the web search request) has possible access to information identifying the context. Hence, the client machine can play an active part in processing the search results to filter and re-rank them based on the context of the search.

A methodology for identifying the context in the form of example documents provided by the user was proposed. A scheme for representing the contextual documents using a VSM was proposed and implemented. Re-ranking algorithms were developed that would process a set of search results for a search query from various search engines and re-rank them using the contextual information. These algorithms were implemented and tested on a document set comprising course notes from an MIT class and the results were found encouraging.

A software application was developed to provide a User Interface to the search algorithm implementation. The software provides also capability for browsing and identifying documents related to the context. The initial user tests with the software were encouraging and strengthen the assumptions of this thesis. However, a lot more needs to be done in order to realize the full potential of context specific searching. Some of them are detailed in the next section.

6.2 Future work

6.2.1 Context representation

Currently the index terms are words occurring in the context document set. However, it is not necessary that words alone be the index terms. Commercial search engines and Information Retrieval systems store phrases as index terms as well. Schemes for detecting and including phrases, as index terms should be provided.

The position of the index terms as well as the stop words in the document is currently not stored. Storing this information would be useful to identify related index terms.

The number of content bearing index terms retained is currently cut off at 10% of the total number of index terms. This measure was fixed based on observations over different document sets. However, there is no statistical generative model of the context documents that justifies this cut off. A statistical model that takes into account the weight of the index terms as given by TF-IDF and determines the cut-off percentage as a function of the weight would be more rational.

6.2.2 Re-ranking algorithms

The query-mapping algorithm does not work when there is no overlap between the query and the context document set. This does not mean that there are no contextual documents that are relevant to the search query. Rather, this algorithm is unable to find the relevant documents. It may be possible to build a statistical model for finding related index terms using the term frequencies, document frequencies and the positional information of index terms in a document. Using this model, one can map both the search query as well as the

related index terms onto the context document set and find a better approximation to the set of contextual documents relevant to the search query.

6.2.3 Query expansion

Since a set of documents identifying the context is available, it might be possible to automatically expand the user search query by adding suitable index terms. This technique goes by the name "Query expansion" and is well studied(S. Chakrabarti 1999). A good query expansion would give a more specific and focused query and hence would be able to get a search result set where the top ranked results are more relevant. However, a bad expansion would completely bring down the quality of results and hence this technique has to be used carefully.

6.2.4 Incorporating feedback

At present the software has only the documents identified by the user to represent the context. However, there are other clues that might indicate the context. For example, the user might be able to provide some explicit feedback on the relevance of the search results. This feedback can be used to refine the context representation. There are well known algorithms for using the user feedback. One such method is called Rocchio's method and it changes the vector representation of the query by incorporating the feedback information. This modified query representation can then be used in the search algorithms.

6.2.5 Building a context specific crawler

The region of the web that is relevant to the context of the user interest is a very small portion of the web. It might be possible to identify these regions in the web by starting

with the context document set and crawling the web intelligently to find documents relevant to the context. An approach called focused crawling (S. Chakrabarti 1999) investigated the building of topic specific crawlers based on a set of exemplary documents. It might be possible to build a similar focused crawler for a specific context. An efficient representation of the web pages found relevant by the crawler could be stored. For a search query entered by the user, these documents can also be considered in addition to the search result documents returned by a search engine. Further, the user feedback on these search results would also be useful in refining the assessment of the relevance of these web pages as well as identifying new regions of interest in the web.

6.2.6 User interface

Presentation of the search results to the user is important for a successful search. The search engine Vivisimo clusters the documents by topics to enable the users to easily browse through the results. A similar idea can be used to present search results to the user. The topics can be identified from the context document set by clustering techniques and the search results can be categorized into these topics. This would provide an intuitive way for the user to browse through the search results. The implicit feedback obtained by observing the category chosen by the user for browsing can be used to refine the future search results.

6.2.7 Computational speed

A more efficient implementation of the proposed search algorithms will reduce the processing time significantly. However a significant portion of the processing time is spent on downloading documents from the web for each search. The context specific

crawling suggested before can reduce this time to some extent. Some of the search results returned might belong to the portion of the web indexed by the crawler and hence a representation of the same would be available. In such a case it would not be necessary to download the document.

6.3 Conclusion

This thesis has presented initial work regarding representing and utilizing contextual information for enhancing web search. Further implementation of the above mentioned ideas would go a long way towards realizing the exciting possibility of personalized web search.

References

A. McCallum, K. N., J. Rennie, K. Seymore (1999). Building Domain-Specific Search Engines with Machine Learning Techniques. AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace.

C. Silverstein, M. R. H., J. Marais and M. Moricz (1999). "Analysis of a very large AltaVista query log." SIGIR Forum **33**.

Chakrabarti, S. (2003). Mining the web, Morgan Kaufmann.

E. Agichtein, S. L., L. Gravano (2001). Learning Search Engine Specific Query Transformations for Question Answering. Tenth International World Wide Web Conference WWW10.

E.T. O'Neill, B. F. L., and R. Bennett, (April 2003). Trends in the evolution of the public web. D-Lib Magazine. **9**.

G. Salton, A. W., C. S. Yang (1975). "A Vector space model for automatic indexing." Communications of the ACM **18**(11).

Google Google web APIs, http://www.google.com/apis/.

J. Price, M. G. (2002). Mastering Visual C#, Sybex.

Korfhage, R. R. (1997). Information storage and retrieval, John Wiley & Sons, Inc.

Kruk, M. (2003). PDFTOHTML, http://pdftohtml.sourceforge.net.

L. Page, S. B., R. Motwani, T. Winograd (1998). The PageRank Citation Ranking: Bringing Order to the Web, Stanford Digital Library Technologies Project.

Lawrence, S. (2000). "Context in Web Search." IEEE Data Engineering Bulletin **23**(3): 25-32.

M. Mitra, A. S., C. Buckley (1998). Improving automatic query expansion. Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval.

M. R. Henzinger, R. M. C. S. (2002). "Challenges in web search engines." ACM SIGIR Forum **36**(2): 11-22.

Porter, M. F. (1980). "An algorithm for suffix stripping,." Program **14**(3): 130-137.

Roush, W. (2004). Search Beyond Google. Technology Review.

S. Chakrabarti, M. V. d. B., B. Dom (1999). Focused crawling. 8th International World Wide Web Conference.

S. Lawrence, C. L. G. (1998). "Searching the World Wide Web." Science **280**(5360): 98-100.

Sun Model-View-Controller documentation from Sun website, http://java.sun.com/blueprints/patterns/MVC.html.