SeRIF: Refining WWW searches using a modified relevance feedback method

Rudy Darmawan
The University of Montana

Let us know how access to this document benefits you.
Follow this and additional works at: https://scholarworks.umt.edu/etd

Recommended Citation
https://scholarworks.umt.edu/etd/5072
Permission is granted by the author to reproduce this material in its entirety, provided that this material is used for scholarly purposes and is properly cited in published works and reports.

** Please check "Yes" or "No" and provide signature **

Yes, I grant permission ✓
No, I do not grant permission 

Author's Signature [Signature]

Date 8/1/96

Any copying for commercial purposes or financial gain may be undertaken only with the author's explicit consent.
SeRIF: Refining WWW Searches Using A Modified Relevance Feedback Method

by

Rudy Darmawan

B.S. The University of Montana, 1994

presented in partial fulfillment of the requirements
for the degree of
Master of Science

The University of Montana
July 1996

Approved by:

[Signatures]

Chairperson

Dean, Graduate School

8/5/96

Date
SeRIF: Refining WWW Searches Using A Modified Relevance Feedback Method (88 pp.)

Director: Nicholas P. Wilde

Systems currently available for searching the World Wide Web (WWW) tend to produce low precision results. This is often due to the search system’s failure to identify the correct semantic identity of the words in the query, and the failure of the user to supply a specific and well defined query. Both failures can be corrected if the users of the search system has a way to communicate their desires, based on the information found in the initially retrieved documents, back to the search system. The relevancy feedback method, from the field of information retrieval, can be used to obtain users’ feedback and automatically build refined queries that will, hopefully, give higher precision results. A system called SeRIF (Search Refinement Incorporating relevancy Feedback), based on existing WWW search system technology and the relevancy feedback method, was built. SeRIF was tested using a variety of queries to determine how well the proposed solution works. The test results showed that SeRIF was able to help in identifying the correct semantic identity of the words used in the initial query. However, SeRIF was not very successful in finding documents on some specific topics based on documents describing more general topics.
Contents

1 Introduction .............................................. 1
   1.1 Problem Description .............................. 1
   1.2 Proposed Solution ................................. 5

2 Understanding What The Search System Does .... 8
   2.1 Components Of A WWW Search System ....... 8
   2.2 Two Modes Of Operation ......................... 10
   2.3 Stemming Algorithm ............................... 15
   2.4 Term Weighting Process ......................... 21
   2.5 Similarity Measurement ........................... 24

3 Finding What The User Actually Wants ........... 25
   3.1 Problem Formulation: Finding What The User Actually Wants .... 25
   3.2 Relevance Feedback And Query Reformulation .... 30

4 Proposed Solution And Implementation .......... 35
   4.1 Proposed Solution ................................. 35
   4.2 System Implementation ......................... 37
       4.2.1 SeRIF User Interface ....................... 41
       4.2.2 SeRIF Indexing System .................... 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.3</td>
<td>User’s Query Database</td>
<td>44</td>
</tr>
<tr>
<td>4.2.4</td>
<td>SeRIF Query Reformulation System</td>
<td>45</td>
</tr>
<tr>
<td>4.3</td>
<td>An Example</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>Testing: Set Up and Result</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Tests Set Up</td>
<td>55</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Test Cases</td>
<td>57</td>
</tr>
<tr>
<td>5.2</td>
<td>Test Result</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>Discussion</td>
<td>72</td>
</tr>
<tr>
<td>6.1</td>
<td>Test Cases Evaluation</td>
<td>72</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Set I</td>
<td>72</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Set II</td>
<td>75</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Set III</td>
<td>76</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Set IV</td>
<td>78</td>
</tr>
<tr>
<td>6.2</td>
<td>General Discussion</td>
<td>79</td>
</tr>
<tr>
<td>6.3</td>
<td>Suggestions For Future Work</td>
<td>80</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions</td>
<td>83</td>
</tr>
<tr>
<td>7.1</td>
<td>Conclusion</td>
<td>83</td>
</tr>
<tr>
<td>7.2</td>
<td>Summary</td>
<td>85</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Modules of A WWW Search System ...................................................... 11

4.1 Components of SeRIF (Search Refinement Incorporating relevancy Feedback System) ................................................................. 38

4.2 Flowchart representation of SeRIF Query Reformulation System ...... 47

4.3 SeRIF Homepage ................................................................................... 49

4.4 Dynamic HTML page showing result of the user defined query .......... 50

4.5 Extra browser window for document viewing .................................... 52

4.6 Result of the automatically formed query ......................................... 53
List of Tables

2.1 Non-Content Words, page 1 ................................................................. 12
2.2 Non-Content Words, page 2 ................................................................. 13
2.3 List of Endings for the Stemming Process, page 1 .............................. 18
2.4 List of Endings for the Stemming Process, page 2 .............................. 19
2.5 Context Sensitive Rules Associated with Endings ............................ 20
2.6 Transformation Rules Used In Recording Process ............................ 22
Chapter 1

Introduction

1.1 Problem Description

In the recent years, we have seen a drastic increase in the number of people using the internet[19]. One of the main factors that has contributed to this increase is the emergence of the World Wide Web (WWW). The WWW is a system that links information from different internet sites. An internet user uses a Web browser to go from one site to another to find information. The success of the WWW can be attributed mostly to the existence of user-friendly browsers that make “surfing the net” easy and fun.

However, the rapid increase of information available through the WWW may cause a problem when a user wants to find some specific information[22]. One solution often proposed are good internet agents. There is no exact definition of an agent. A broad definition given, by F.C. Cheong[6] is:

“Simply put, agents can be considered personal software assistants with authority delegated from their users.”

An internet agent is basically an agent that operates over the internet. The job of
this kind of agent is often to assist its human user in finding information in fast and effective ways. These needs have inspired many research projects which in turn have given birth to many different Web robots/agents[6].

WWW search systems are one type of well-known internet agent. There are many agents that can be classified as WWW search systems, such as the WebCrawler (University of Washington) and Lycos (Carnegie Mellon University). These agents, although typically called “search engines” for short, are actually made up of several different cooperating software modules. Modules of a typical WWW search system include:

1. A Web agent module (find and fetch web pages).
2. An indexing module (index web sites and pages).
3. A query server module (resolve user’s given queries).
4. A search-engine module (coordinate activities of other modules).

Throughout this thesis, I will use the term “WWW Search System” or “Search System,” for short, to refer to the entire collection of modules. I will refer to the individual modules by their module names.

One disadvantage of this type of WWW Search System agent is that they waste a lot of network resources because they work independently from one another, while they are building very similar things[6]. One proposed solution for this problem is a new kind of indexing agent. Research on these agents concentrates on how to efficiently gather information to build larger and more complete indexes. The current state of the art in research for these kinds of agents seems to suggest a distributed agent. That is, the agent is actually a collaboration of several agents running on different machines and communicating their findings with each other.
The two most mentioned agents of this kind are Harvest (University of Colorado) and WebAnts (Carnegie Mellon University).

Harvest consists mainly of two components, a *gatherer*, which collects indexing information and a *broker*, which provides an incrementally indexed query interface to the gathered information. The gatherer is designed with the ability to run on the provider site, thus saving a great deal of server load and network traffic. The information gathered by gatherers can be summarized and interpolated upward to many brokers. Brokers can also share information between themselves[11]. This tree-like approach allow the sharing of information so gatherers will not index the same document twice. WebAnts tries to solve the problem with a cooperative Web explorer (ant). Ants are designed to share information so no duplicate efforts are made in the information discovery and indexing process[23].

Although these newer indexing agents may solve to some degree the network bottleneck problem, they do not provide a solution to another major problem of the WWW search system. Search systems often do not return what the user is actually looking for, or rather, the real problem is that they often return so many links, consisting of both related and unrelated information, that the relevant information is lost in the noise.

In the field of information retrieval, the quality of the result of an index search can be measured in two different forms: *recall* and *precision* [9]. Recall ($R$) is the proportion of relevant material retrieved from all the relevant material available in the index:

$$R = \frac{\text{number of items retrieved and relevant}}{\text{total relevant items in collection}}$$

Precision ($P$) is the proportion of relevant material over all the retrieved material:

$$P = \frac{\text{number of items retrieved and relevant}}{\text{total retrieved}}$$
According to Pinkerton[2], the creator of WebCrawler (a search engine), the main weakness of the WebCrawler (and other WWW search systems) are their precision. WebCrawler is usually able to retrieve enough relevant information from the available collection (i.e. it has high recall). However, in the process, it usually also retrieves a great deal of non-relevant information (i.e. it has low precision). This is caused mainly by the following problems:

1. Novice users have problems writing well-formed queries. Problematic areas include syntactic errors, incorrect specification of Boolean operations, and misunderstanding of the indexing system and underlying system model [20]. The problem is caused by the lack of training that most users have in query formulation.

2. There is tremendous diversity in the words people may use to describe an object or a concept. This places limits on keyword-based systems' performance[20]. There is no one-to-one matching between word choice and meaning. So the problem is caused by the search system relaying on the textual instead of the semantic use of words. Even if the user gives a “good” query, the WWW search system will still have problem identifying what the user really wants.

3. Often, the user doesn’t know exactly what he/she is searching for in the first place. This leads to the use of more generic terms in the query formulation. For example, an out-of-state traveller might type in “Montana National Park.” However, what they are really looking for is information about Glacier National Park. For one reason or another, they may not remember the name of the park, so they go with the more general term. This search will result in many links to all kinds of park information in Montana.
Several solutions for the above problems have been proposed and worked on, but none have fully solved the problem yet. Softbot is one internet agent system that is under development at the University of Washington [18]. The ultimate purpose of Softbot is to take away from the user many of the difficulties of finding information on the internet by delegating the task to an agent. The scope of Softbot is much broader than just the WWW. Softbot attempts to help internet users in finding a person (similar to Netfind[17]), information (including surfing the web and downloading information via ftp), a particular publication, etc. The drawback with Softbot is that the user must know precisely what he/she is looking for. For example, Softbot is useful when a user needs to find a specific publication by a specific author. Users must articulate their query very precisely before submitting the request to Softbot. Often, users will have only a very general idea about what they want to find. A good analogy to this problem is a shopper in a large mall. When shoppers go to the mall, they might be looking for a general item, say a television. However, they might not know the exact store that they need to go to, nor the exact brand to buy. If the shopper were to know the exact store, brand, and model, he/she could have asked a friend (or agent) to get it. At this point, Softbot's knowledge base is still too limited, to be adapted for this kind of general internet searching. The other problem with a private robot, like Softbot is the heavy load that it puts on the network[2].

1.2 Proposed Solution

Truly intelligent internet agents, that find the correct information for the user with the least effort on the user's part, and that need only small amounts of network bandwidth, would be ideal. In the future, a system like Softbot might be able to achieve this ideal. At present, the WWW search system agent approach seems to be
the most practical one. However, the main problem with search systems is their lack of precision. What this means is that the information that the user wants is there, but it is often intermixed with a lot of non-useful information. The causes of this low precision are described in the previous section. These causes are all related to the user's given query and the search system's misinterpretation of the query. The question that we have, then, is whether it is possible to form a new kind of query that when used with the WWW search system will return a higher precision result.

From the research in the field of information retrieval, it has been shown that it is possible to automatically form a query that will result in a higher precision retrieval. This kind of query reformulation is based on the relevancy feedback method[7]. The technique of automatic query reformulation can be used if three components are available:

1. An initial, user-defined query.

2. A small subset of documents from the big list of documents retrieved using the initial query.

3. User-given feedback on the relevancy of each of the document in that small set of documents.

I believe that it is possible to use this relevancy feedback method to build a new variety of WWW search system. This type of search system, when given the components described above, will be able formulate a new query. This new query, when used with an existing WWW search system in the retrieval process, will give a better precision result. This thesis presents the implementation and testing of such a WWW search system, called SeRIF (Search Refinement Incorporating relevancy Feedback system)
There are two main parts to this new variety of search system. The first part is an existing WWW search system that will provide an initial documents collection based on user's query. Since it is essential for us to understand how the search system works, I have devoted Chapter 2 of this thesis to a discussion of WWW search systems. The second part of our solution is a system that will get a user's feedback and using that feedback, form a new, hopefully better query. The understanding of user feedback is important to this second part. Thus Chapter 3 describes how a search system can be told what a user really wants. In chapter 4, the proposed solution is described in more detail as well as the implementation of my system, SeRIF. Chapter 5 contains the description of tests performed using this system as well as the results obtained. In chapter 6, the results of the testing are presented in more detail. Finally, chapter 7 will provide the conclusion and summarize lessons learned from this thesis.
Chapter 2

Understanding What The Search System Does

2.1 Components Of A WWW Search System

An understanding of the components of a typical search system is critical for this project. As the objective is to refine the result returned by the WWW search system, it is necessary to know how the search system produces those results in the first place. This understanding becomes even more necessary when we try to apply some of these same techniques in our refinement process.

A WWW search system usually consists of four modules[6]:

1. The search-engine module.
   This is the "brain" of the whole WWW search system. Its main function is to direct and coordinates the activities of the rest of the system. It also determines the set of sites that contain documents to be retrieved and initiates the retrieval process by sending requests to the Web agents module. The search-engine module generally has two modes of operation. Refer to Section 2.2 for descriptions
of these modes.

2. The agent module.

This module manages the work of internet agents. These internet agents are responsible for actually retrieving the documents from the different URL sites. One important aspect of these agents/robot is that they should abide to a set of conventions for web robots. These conventions are not standard, but rather are guidelines based on a consensus of the various WWW newsgroups and mailing lists. One of the most important guidelines is that the web robot should not bog down a server by rapidly firing requests to that server. For this reason, most document retrieval agents use some sort of scheduling methods for choosing the server to access. One of those scheduling methods is based on the modified breath-first algorithm. The WebCrawler system uses this method[2]. Given a set of sites to visit, agents based on this method will fetch the first level documents from those sites. After all the first level documents are retrieved, the agents retrieves the second level documents by following links from the first level documents. The other scheduling method is based on a probabilistic scheme like the one used by Lycos[6]. Since the agent is rotating between different sites, it will not slow down the server of a certain site. More information about these guidelines[16] and ways to exclude certain robots from a web server[15], is available at http://info.webcrawler.com/mak/projects/robots/robots.html.

3. The indexing module.

This module indexes and stores those documents retrieved by the agents in a database. The database does not contain the raw documents, rather it contains the document metadata (i.e. its location, size, etc.), links between documents, and the full-text index. Full-text indexing means using all terms found in the
document as indexes for that document. *Terms* are defined as non-trivial words that are have been reduced to their morphological roots. Full-text indexing usually reflects the content of a document better than title or abstract indexing. In the early days of “traditional” information retrieval field, full-text indexing was not very feasible, but as the price of computer based information storage continues to fall, it has become a very common practice in the field.

4. The query server module.

This module consists of two parts: the front end and the back end. The front end is the Web document that the user sees and uses to enter his/her queries. The back end is an engine that processes user queries to find related documents to be returned to the user. The process of finding the documents is often based on the similarity between the document with the query. See Section 2.5 for explanation of the *similarity measurement* process.

Figure 2.1 shows a pictoral representation of these four modules.

### 2.2 Two Modes Of Operation

As mentioned earlier, a typical search system has two modes of operation. The first one is the *indexing mode*. The goal of this mode is to build indexes of as many sites of the Web as possible, without placing too large a load on any specific Web server. The search-engine module of the system will build a list of sites to be visited and send the requests to the internet agents of the agent module. The agents fetch documents using some sort of scheduling method, as described in the agent module of Section 2.1. Each fetched document is then indexed. One of the more common indexing methods is based on the vector space model[1].
The vector space model works as follows:

1. Non-content words are stripped from the document.

   Non-content words are words that do not carry specific meaning in the document. They appear as “connecting” words, “stop” words or “fluff” words. Typically these words are high frequency words and can comprise up to 40-50 percent of the text in a document[9]. H.P. Luhn, one of the pioneers in the field of information retrieval, specified a cut-off threshold for excluding non-significant words[13]. The cut-off is used to eliminate words that occur so often in English documents, that they do not aid significantly in identifying the contents of a document. Using this cut-off, a list of non-content words in English has been identified. The lists of those words, taken from Rijsbergen’s book[5], are shown in Table 2.1 and Table 2.2.
<table>
<thead>
<tr>
<th>A</th>
<th>ABOUT</th>
<th>ABOVE</th>
<th>ACROSS</th>
<th>AFTER</th>
<th>AFTERWARDS</th>
<th>AGAIN</th>
<th>AGAINST</th>
<th>ALL</th>
<th>ALMOST</th>
<th>ALONE</th>
<th>ALONG</th>
<th>ALREADY</th>
<th>ALSO</th>
<th>ALTHOUGH</th>
<th>ALWAYS</th>
<th>AMONG</th>
<th>AMONGST</th>
<th>AN</th>
<th>AND</th>
<th>ANOTHER</th>
<th>ANY</th>
<th>ANYHOW</th>
<th>ANYONE</th>
<th>ANYTHING</th>
<th>ANYWHERE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANNOT</td>
<td>CO</td>
<td>COULD</td>
<td>DOWN</td>
<td>DURING</td>
<td>EACH</td>
<td>EG</td>
<td>EITHER</td>
<td>ELSE</td>
<td>ELSEWHERE</td>
<td>ENOUGH</td>
<td>ETC</td>
<td>EVEN</td>
<td>EVER</td>
<td>EVERY</td>
<td>EVERYONE</td>
<td>EVERYTHING</td>
<td>EVERYWHERE</td>
<td>EXCEPT</td>
<td>FEW</td>
<td>FIRST</td>
<td>FOR</td>
<td>FORMER</td>
<td>FORMERLY</td>
<td>FROM</td>
<td>FURTHER</td>
</tr>
<tr>
<td>IS</td>
<td>IT</td>
<td>ITS</td>
<td>ITSELF</td>
<td>LAST</td>
<td>LATTER</td>
<td>LATTERLY</td>
<td>LEAST</td>
<td>LESS</td>
<td>LTD</td>
<td>MANY</td>
<td>MAY</td>
<td>ME</td>
<td>MEANWHILE</td>
<td>MIGHT</td>
<td>MORE</td>
<td>MOREOVER</td>
<td>MOST</td>
<td>MOSTLY</td>
<td>MUCH</td>
<td>MUST</td>
<td>MY</td>
<td>MYSELF</td>
<td>NAMELY</td>
<td>NEITHER</td>
<td>OUR</td>
</tr>
</tbody>
</table>

Table 2.1: Non-Content Words, page 1
| ARE       | HAD       | NEVER     | STILL     | WHEREIN  |
| AROUND    | HAS       | NEVERTHELESS | SUCH     | WHEREUPON |
| AS        | HAVE      | NEXT      | THAN      | WHEREVER  |
| AT        | HE        | NO        | THAT      | WHETHER   |
| BE        | HENCE     | NOBODY    | THE       | WHITHER   |
| BECAME    | HER       | NONE      | THEIR     | WHICH     |
| BECAUSE   | HERE      | NOONE     | THEM      | WHILE     |
| BECOME    | HEREAFTER | NOR       | THEMSELVES | WHO       |
| BECOMES   | HEREBY    | NOT       | THEN      | WHOEVER   |
| BECOMING  | HEREIN    | NOTHING   | THENCE    | WHOLE     |
| BEEN      | HEREUPON  | NOW       | THERE     | WHOM      |
| BEFORE    | HERS      | NOWHERE   | THEREAFTER | WHOSE     |
| BEFOREHAND| HERSELF   | OF        | THEREBY   | WHY       |
| BEHIND    | HIM       | OFF       | THEREFORE  | WILL      |
| BEING     | HIMSELF   | OFTEN     | THEREIN   | WITH      |
| BELOW     | HIS       | ON        | THEREUPON | WITHIN    |
| BESIDE    | HOW       | ONCE      | THESE     | WITHOUT   |
| BESIDES   | HOWEVER   | ONE       | THEY      | WOULD     |
| BETWEEN   | I         | ONLY      | THIS      | YET       |
| BEYOND    | IE        | ONTO      | THOSE     | YOU       |
| BOTH      | IF        | OR        | THOUGH    | YOUR      |
| BUT       | IN        | OTHER     | THROUGH   | YOURS     |
| BY        | INC       | OTHERS    | THROUGHOUT | YOURSELF  |
| CAN       | INDEED    | OTHERWISE | THRU      | YOURSELVES |

Table 2.2: Non-Content Words, page 2
2. Words in the document are reduced to their morphological roots. These words that have been reduced to their morphological roots are called terms. The reason for this step is because the information that is significant to the user is contained in the root of the word. Word suffixes function only as a mechanism for expressing the root in a grammatical form[14]. For example, the words “runs” and “running” come from the same base word “run.” The essential word is run, the prefixes “s” and “ing” being used because of their grammatical significance. Because morphological roots are also known as stems, this reduction process is known as the stemming process. The stemming process is based on the stemming algorithm described in Section 2.3.

3. Terms in the document have differing importance to the document as a whole. The degree of importance of a term is reflected in its weight. Term weight is affected by the number of occurrences of that term inside that particular document, the number of occurrences of that term across all documents in the collection and the size (number of terms) of that particular document where the term is found. The term weighting process is described in Section 2.4.

The second mode of a WWW search system is the real mode. The goal of this mode is to find documents that are most similar to the user’s query[2]. When the WWW search system receives a user’s query, it will try to find documents on its database that are similar to the query. The similar documents are found by measuring the similarity between the user’s query with the documents already collected in the indexing mode. The similarity measurement process is explained in Section 2.5. The query-documents similarity measurement will give an initial documents list. In the real mode of operation, the WWW search system will follow links from these documents to find more documents. The WWW search system will check its database
to see if documents referred to by the initial documents are already indexed. If the documents are already indexed, the links from those documents will be followed, otherwise, the documents will be indexed and then the links from these documents are also followed.

The reason for following links from previously retrieved relevant documents is based on the intuition that we are more likely to find relevant documents from other relevant documents. Unfortunately, this is not a correct assumption if the original documents themselves are not relevant documents. This does not necessary mean that the similarity measurement process is defective. It is often true that some words have more than one meaning. So while the document might contain the first meaning of the word, the user may want the other one. The problem here is that the search system is depending totally on its own (rather rudimentary) knowledge.

When new documents are retrieved the indexing process is run over these documents to include them in the documents collection. The similarity calculation process is then rerun to find new list of relevant documents. The process will then be repeated until enough relevant documents are found or a timeout occurs. The result of this process is what a typical search system user might see when they submit a query to the query server.

2.3 Stemming Algorithm

A stemming algorithm is a computational procedure that reduces all words with the same root/stem to a common form[14]. This procedure usually involves stripping the derivational and inflectional suffixes of those words. The algorithm was originally built in the 1960’s for applications in the area of computational linguistics and information retrieval. The original purpose of this algorithm is to maximize the usefulness of
the subject terms in document classification[14]. However, it has been shown that stemming words in a document is also useful in the indexing environment to reduce a variety of different forms of the same root word[9]. For example the words "analysis," "analyzing," "analyzed," and "analysing" come from the same stem "analy." When a stem word such as "analy" is used to identify documents as well as in queries, it is easier to find relevant matches between documents and queries.

A typical stemming algorithm uses two principles: iteration and longest-match. Iteration is used because of the fact that suffixes are attached to stems in a certain order. Suffixes are thus separated into different order-classes. For example to stem the word "relatedness," we need to go through two iterations. The first time through, we remove "ness" and the second time, we remove "ed," leaving the stem "relat." The longest-matching principle states that, if there are two possible suffixes that can be removed from a word at the same time, the longest-matching suffix should be removed. For example when stemming the word "proclamation," we can take out the suffix "ion" or the suffix "ation." In this case, we should take out the longer suffix ("ation").

These two principals are not enough for a good stemming algorithm. The reason is that the stems generated often contain problems due to spelling exceptions. It turns out that stems can be spelled in more than one way. In English these problems often are due to the Latinate derivations[14]. For example "producer" and "production" should reduce to the same stem. However removing the suffix "er" from "producer" gives the stem "produc" whereas removing the suffix "ion" from "production" results in the stem "product." Some sort of post stemming procedures are needed to handle these exceptions. Two of those are recording and partial matching[8]. The idea behind recording is that most of the spelling changes that occur can be adequately covered
by a small set of context sensitive transformation rules. The partial matching method is based on the assumption that spelling changes in English are restricted to certain types which may occur, but do not always occur, and the assumption that these changes involve no more than two letters at the end of a stem.

There are many varieties of stemming algorithms. For the purpose of this thesis, I have chosen to use an algorithm based on the method built by G. Salton as part of the SMART project\cite{8} and a paper by J.B. Lovins\cite{14}. The algorithm consists of two parts. The first one is the ending/suffix removal process, the second one is the recording process.

This is how the removal process works:

1. Determine the minimum length of the final stem.

2. Given a word and an ending list, find the longest ending from the ending list that matches the suffix of the word. The ending lists are shown in Table 2.3 and Table 2.4. Note: The capital letters are not part of the ending, rather it is a condition code. The meaning of those rules are shown in Table 2.5. If such an ending exists continue to the next step, otherwise that word is the stem so go to the recording process.

3. Check to see if the context sensitive rule of that particular ending is satisfied. The context sensitive rules is shown in Table 2.5.

4. If the context sensitive rule is satisfied, remove the ending.

5. Regardless of what happened in step 4, choose the next longest ending that matches the remaining suffix of the word. If such an ending exists, go to back to step 3, otherwise we have found the stem of the word so we go to the recording process.
<table>
<thead>
<tr>
<th>Ending Length</th>
<th>Ending Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>alistically B   arizability A   izationally B</td>
</tr>
<tr>
<td>10</td>
<td>antialness A   arisations A   arizations A   entialness A</td>
</tr>
<tr>
<td>9</td>
<td>allically C    antaneous A   antiality A   arisation A   arization A</td>
</tr>
<tr>
<td></td>
<td>ationally A    ativeness A   eableness E   entations A   entiality A</td>
</tr>
<tr>
<td></td>
<td>entialize A    entiation A   ionalness A   istically A   itousness A</td>
</tr>
<tr>
<td></td>
<td>ization A      izational A</td>
</tr>
<tr>
<td>8</td>
<td>ableness A    arizable A   entation A   entially A   eousness A</td>
</tr>
<tr>
<td></td>
<td>ibleness A    icalness A   ionalism A   ionality A   ionalize A</td>
</tr>
<tr>
<td></td>
<td>ousness A    izations A   lessness A</td>
</tr>
<tr>
<td>7</td>
<td>ability A    aically A   alistic B   alities A   ariness E</td>
</tr>
<tr>
<td></td>
<td>aristic A    arizing A   ateness A   atingly A   ational B</td>
</tr>
<tr>
<td></td>
<td>atively A    ativism A   elihood E   encible A   entally A</td>
</tr>
<tr>
<td></td>
<td>entials A    entiate A   entness A   fulness A   ibility A</td>
</tr>
<tr>
<td></td>
<td>icalism A    icalist A   icality A   icalize A   ication G</td>
</tr>
<tr>
<td></td>
<td>icianry A    ination A   ingness A   ionally A   isation A</td>
</tr>
<tr>
<td></td>
<td>ishness A    istical A   iteness A   iveness A   ivistic A</td>
</tr>
<tr>
<td></td>
<td>ivities A    ization F   izement A   idially A   ousness A</td>
</tr>
<tr>
<td>6</td>
<td>aceous A    acious B   action G   alness A   ancial A   ancies A</td>
</tr>
<tr>
<td></td>
<td>ancing B    ariser A   arized A   arizer A   atable A   ations B</td>
</tr>
<tr>
<td></td>
<td>atives A    eature Z   efully A   encies A   encing A   ential A</td>
</tr>
<tr>
<td></td>
<td>enting C    entist A   eously A   ialist A   iality A   ialize A</td>
</tr>
<tr>
<td></td>
<td>ically A    icance A   icians A   icists A   ifully A   ionals A</td>
</tr>
<tr>
<td></td>
<td>ionate D    ioning A   ionist A   ously A   istics A   izable E</td>
</tr>
<tr>
<td></td>
<td>lessly A    nesses A   oidism A</td>
</tr>
</tbody>
</table>

Table 2.3: List of Endings for the Stemming Process, page 1
Table 2.4: List of Endings for the Stemming Process, page 2
A -> No restriction on stem
B -> Minimum stem length = 3
C -> Minimum stem length = 4
D -> Minimum stem length = 5
E -> Do not remove ending after 'e'
F -> Minimum stem length = 3 and do not remove ending after 'e'
G -> Minimum stem length = 3 and remove ending only after 'f'
H -> Remove stem ending only after 't' or 'll'
I -> Do not remove ending after 'o' or 'e'
J -> Do not remove ending after 'a' or 'e'
K -> Minimum stem length = 3 and remove ending only after 'l', 'i',
or 'u'x'e' (where x stands for any letter).
L -> Do not remove ending after 'u', 'x', or 's', unless 's'
follows 'o'
M -> Do not remove ending after 'a', 'c', 'e', or 'm'
N -> Minimum stem length = 4 after 's'xx
(where x stands for any letter), elsewhere = 3
O -> Remove ending only after 'l' or 'i'
P -> Do not remove ending after 'c'
Q -> Minimum stem length = 3 and do not remove ending after 'l',
or 'n'
R -> Remove ending only after 'n' or 'r'
S -> Remove ending only after 'dr' or 't', unless 't' follows 't'
T -> Remove ending only after 's' or 't', unless 't' follows 'o'
U -> Remove ending only after 'l', 'm', 'n', or 'r'
V -> Remove ending only after 'c'
W -> Do not remove ending after 's' or 'u'
X -> Remove ending only after 'l', 'i', or 'u'x'e'
(where x stands for any letter)
Y -> Remove ending only after 'in'
Z -> Do not remove ending after 'f'
AA -> Remove ending only after 'd', 'f', 'ph', 'th', 'l',
'er', 'or', 'es', or 't'
BB -> Minimum stem length = 3 and do not remove ending after 'met' or
'ryst'
CC -> Remove ending only after 'l'

Table 2.5: Context Sensitive Rules Associated with Endings
The recording process:

1. Given the stem generated by the ending removal process, undouble the final consonant of the stem, if applicable.

2. The remaining stem is checked against the transformation rules. If there is an applicable rule, use that rule to record the stem. If no rule is applicable, that stem is the final stem. The transformation rules are given in Table 2.6

3. Record the final stem.

The stems thus generated by the stemming process are also called terms. It is, these terms that are then weighted using the term weighting process.

2.4 Term Weighting Process

A term’s weight is an indicator of the importance of that term in a particular document[1]. More important terms are assigned higher weights.

A term’s weight is defined as follows:

1. Let $i$ be a term in the document where $i = 1, 2, ..., n$.

2. Let $tf_i$ (term frequency) be the number of occurrences of term $i$ in the document.

3. Let $idf_i$ (inverse document frequency) be the inverse of the number of occurrence of term $i$ across all documents in the collection.

4. Let $E$ be the Euclidean length of the document. Where

$$E = \sqrt{\sum_{i=1}^{n} (tf_i \times idf_i)}$$
1. Remove one of double "b", "d", "g", "l", "m", "n", "p", "r", "s", "t"
2. "iev" -> "ief"
3. "uct" -> "uc"
4. "umpt" -> "um"
5. "rpt" -> "rb"
6. "urs" -> "ur"
7. "istr" -> "ister"
8. "metr" -> "meter"
9. "olv" -> "olut"
10. "ul" -> "l" except following "a", "i", "o"
11. "bex" -> "bic"
12. "dex" -> "dic"
13. "pex" -> "pic"
14. "tex" -> "tic"
15. "ax" -> "ac"
16. "ex" -> "ec"
17. "ix" -> "ic"
18. "lux" -> "luc"
19. "uad" -> "uas"
20. "vad" -> "vas"
21. "cid" -> "cis"
22. "lid" -> "lis"
23. "erid" -> "eris"
24. "pand" -> "pans"
25. "end" -> "ens" except following "s"
26. "ond" -> "ons"
27. "lud" -> "lus"
28. "rud" -> "rus"
29. "her" -> "hes" except following "p", "t"
30. "mit" -> "mis"
31. "end" -> "ens" except following "m"
32. "ert" -> "ers"
33. "et" -> "es" except following "n"
34. "yt" -> "ys"
35. "yz" -> "ys"

Table 2.6: Transformation Rules Used In Recording Process
then the weight of term $i$ in the document is

$$Weight_i = \frac{tf_i \times idf_i}{E}$$

There are three factors that affect the term weight: the term frequency ($tf_i$), the inverse document frequency ($idf_i$), and the Euclidean length of the document ($E$). A term that occurs more frequently in a document (high $tf_i$), will have a higher weight. The logic behind this is that a document writer often repeats terms that are important to what he/she is writing about. It is important to note here that these high frequency terms come from content words, since the non-content words were stripped in the earlier processing. There are, however, some terms that are common to many documents in a certain topic. For example, the word PC (Personal Computer) may not be a very discriminating word in the document collection when we are searching about “Different types of PC”. However, the same word (PC) is a discriminating word when we are searching about “Different types of computers”. This consideration is factored into the equation by taking an inverse of the occurrence of the term ($idf_i$) across all documents in the collection. The more often a term occurs across the collection, the less important it becomes to the user. There is also a known misleading factor that will increase the weight of a term - the length of the document. A term generally occurs more often in a longer document than in a shorter one. In order to correct this, the Euclidean length is used to neutralize the difference in document lengths.

The terms, along with their weights, are stored in the database by the indexing module of the WWW search system. A document’s weighted terms are the product of the indexing mode of the system. They are a very important part when matching a user’s query with documents in the collection, through the similarity measurement approach.
2.5 Similarity Measurement

Similarity measurement is a method for assessing the relevance of documents in a collection, based on the weights of the terms found in the user’s query and the documents. In the previous sections, the process by which terms in a document are weighted has been described. The same process is applied to the words in a user supplied query.

Once the weighting of the terms in the query is done, we have term weights for both the documents and the query. In order to find a set of matching documents, similarities between the query and documents are calculated based on the weights of their respective terms, as follows:

1. Let \( j \) be a document in the document collection, where \( j = 1, 2, \ldots, m \).

2. Let \( i \) be a common term in both the query and document \( j \), where \( i = 1, 2, \ldots, n \).

3. Let \( d_{ij} \) be the weight of term \( i \) in document \( j \) and \( q_i \) be the weight of term \( i \) in the query.

then the similarity between the query \( (Q) \) and the document \( j \) \((D_j)\) is

\[
Sim(Q, D_j) = \sum_{i=1}^{n} (q_i \times d_{ij})
\]

Documents with higher similarity to the query, get a higher ranking. Performing this similarity test against all available (i.e. indexed documents) provides the WWW search system an initial list of matching documents.
Chapter 3

Finding What The User Actually Wants

3.1 Problem Formulation: Finding What The User Actually Wants

Chapter 2, discussed the workings of a typical WWW search system. Here, I analyze the reasons for its deficiencies. The primary deficiency of concern is its lack of precision. As defined before, precision is simply the proportion of all the retrieved documents that are relevant. Low precision means that the retrieved documents consist of many documents which are not relevant to the intended search from the user’s point of view. The problem here is the difference between what the user wants and what has been communicated to the WWW search system. This problem is caused by the following factors:

1. The difficulties of writing a well formed query, especially for novice users[20]. Users often write queries that are interpreted differently by the search system than intended. For example, the use of boolean operation symbol is often
confusing to a novice user. The user might intend to “AND” two words in the query, but uses the “OR” symbol because he/she doesn’t know the difference. The problem is caused by the lack of training that most users have in query formulation.

2. There is no one-to-one matching between the **textual** identity of a word and the **semantic** identity of that word. Textual identity is based on the alphabetical characters that form the word. The textual identity of the word “java” is a word that is formed by the letters “j,” “a,” “v,” and “a,” in that order. The semantic identity is based on the definition of the word as found in a dictionary. A word can only have one textual identity, but it can have many semantic identities depending on the context that it is used in. The word “java” can mean the Java\(^{tm}\) programming language developed by Sun Microsystems, Inc., or the Java island located in Indonesia, or the slang name for coffee. A WWW search system only uses the textual identity of the word, not its semantic identity. When a user gives a query to the system, it will match the query with the document index based on the textual identity of the words found in the query. Even if the user gives a well-formed query, the search system might still have problem identifying what the user really wants, if the terms in the query have multiple semantic identities.

3. Many times, when users begin a search, they don’t know the exact thing that they are looking for. They might have a general idea about the search topic, but not the exact piece of information. This leads to the use of more generic terms in the query formulation. For example, an out-of-state traveller might type in “Montana National Park.” However, what they are really after is information about Glacier National Park. For one reason or another, they might not
remember the name of the park, so they go with the more general term. This search will result in many links to many different kinds of park information in Montana. Since the query is very general, the result returned by the search system is also very general.

Most WWW search systems try to solve the first problem by giving a list of boolean operators that the user can choose from in a clear way (i.e. using the actual word “AND” and “OR” instead of the symbolic operator). By default, most search systems assume that the user wants to “AND” all the words in the query. This is done with the assumption that a true novice (one who does not even know about the existence of boolean operators) will be searching on a single subject at a time. A typical search system will give help pages to assist users in forming more complex queries using the system. Although this problem still exists even after applying these solutions, I will not deal with this problem in this thesis. For this thesis, I will concentrate only on the second and third problems.

WWW search systems often return so many documents that it is impossible for the users to read them all in a reasonable time. A typical search system has an option for the user to set the maximum number of results returned. The users can thus set a small enough number that they can read all of those documents. However since the precision is low, not all of this small number of documents will contain the information that the users are looking for. Both the second and third problems described earlier contribute to this low precision problem. However, even though the precision of the initial search result may be low, it provides a good place to start the searching process if the search system can somehow incorporate the “missing information” needed to do more searching. What is meant by the missing information here is information not available to the users when they formed the initial query. Since the information is
not available to the users, it can not have been expressed in the initial query. There are two types of missing information depending on whether it is caused by the second or the third problem described above.

The second problem may cause the search system to return documents which contain the right words, but in a different context. The users might not know how well their intended semantic identity of the word matches with what is available in the index of the WWW search system. Since the search system only uses the textual identity of a word, users have no way of knowing whether the search system is retrieving documents that match the intended semantic identity, until they receive the results of their query back from the system. The missing information, in this case, is the documents that best convey the semantic identity of the words as the user understands them. Once users receive the initial search result, they can look at those documents and determine which ones carry the semantic identity they had in mind when they formed the query. The users may want the WWW search system to fetch more documents similar to those documents that convey the right semantic identity of the query words from the users point of view.

The third problem may cause the WWW search system to return documents that are only somewhat related to what the user is actually searching for. The users might not know the specific item they are looking for when forming the initial query. It might be that they do not know if the information that they are looking for actually exists, or they simply do not remember the specific name of the particular piece of information. The missing information, in this case, might be words in those returned documents that more specifically indicate the information searched for. Once the users get back the result of the initial search, they can look at those documents and find some documents that have the specific information that they are looking for.
After the users have read the small set of documents returned by the initial search process, this missing information may become available to them. However, with current search systems, the users cannot communicate their findings to the WWW search system directly. They cannot ask the search system to use their findings to find more similar documents. The users can certainly reform and resubmit the query, but they would first have to figure out what terms to add to the query. The users would need to find those words that make some documents more relevant than the others. We need to keep in mind that those words are not the ones that the users initially thought of as good distinguishing words. If that were the case, the users would have gotten all right documents in the first place. The process of finding the right words to add to the query is certainly not trivial, and is considered by others to be a hard intellectual task[7].

What is needed is a new kind of system that will communicate the users’ finding to the WWW search system automatically. Essentially, we need a system that can provide the solution to the following problem:

*Given a small set of documents retrieved by a WWW search system, is it possible to develop a new variety of search system that will get the users’ opinion of relevancy to the goals of the search for each document in the set. Can the system then put a discrete value on the users’ opinion and use that value to find a new set of documents that has better precision than the first set of documents.*

So, there is a need to somehow reformulate the initial query to express the newly given information. In the field of information retrieval, there is known technique to do a *Query Reformulation* based on the *Relevance Feedback* theory[7]. By applying this theory, it is possible for us to build the system mentioned above.
3.2 Relevance Feedback And Query Reformulation

Relevance Feedback is a technique by which either a human user or an intelligent computer program (agent) gives feedback on how well a certain document matches what is being searched for. The technique was originally developed in the mid-1960s. The assumption behind this method is that the initial retrieval process usually does not yield a good result[7]. The reasons for the failure are caused by the problems described in Section 3.1.

The initial retrieval, therefore, is conducted more as a trial run to get an initial set of documents that can be used to form a new and more relevant query. Given the results of the initial search, the relevance feedback process will choose important terms attached to the documents identified as important to the users. In the same token, the process will discard terms attached to the documents that are identified as non-relevant to the user. Essentially, the process aims to find the most distinguishing terms that identify the relevant documents, then adds those terms into the query for next round retrievals. The relevance feedback method will move the query closer to the relevant items and away from the non-relevant items. The relevancy feedback method also takes away the details of the query formulation process from the users and constructs useful search statements automatically[7].

Several different forms of query reformulation using relevancy feedback have been proposed and used. Please refer to papers by Salton et. all [9], [7], [21] for more information about the different types of relevance feedback methods available. Most variations are caused by the differences in the document collection under which the relevancy feedback method is applied. The differences are the size of documents in the collection and the diversity of documents in the collections. Since most of exper-
mentation related to the relevancy feedback method was done before the emergence of the WWW, there are no known experimental results that show how well the relevancy feedback method applies to the document collection found on the WWW. I believe that the WWW’s document collection is somewhat unique, and different from the document collection studied in the traditional information retrieval field. The document collection on the WWW can be characterized as follows:

1. Documents are in the form of hypertext. However, a typical WWW search system only does the text indexing. What this means is that even the full text indexing system cannot capture all the information presented in the document.

2. The definition of “a document” in the WWW is often different from the traditional definition of “a document.” In the traditional sense a document means a whole entity (i.e. a book, a paper, or an article). In the WWW, a document refers to one Web page. The page might contain a whole entity, or it may contain part of that whole entity with hyperlinks to other parts of that entity located on other Web pages. Of course, a writer can always put the whole document on one Web page, but if the document is huge, it will take a long time to transfer the document to the reader’s site. This certainly defeats the purpose of hypertext.

3. Because the term document is used to refer to a Web page, a typical WWW documents’ size is fairly small. From our experience, a typical WWW document contains less than 20KB of text. However, this is not a guarantee, since it is possible to put a whole book into one WWW page.

4. The topics covered by WWW documents are very diverse - anything from cooking to music to science. However, I find that there are more computer related
materials than other topics.

For the purposes of this thesis, I have attempted to modify the relevancy feedback method to reflect what might be most suited for the WWW document collection. This is how a new query is formed in our modified relevancy feedback method:

1. Let $D_i$ be a document returned by the initial retrieval, $i = 1, 2, ..., N$. The initial retrieval is done using the user defined query. Individual documents thus retrieved can be relevant or non-relevant.

2. Let $W_i$ be the weight given by the user to the document $D_i$, based on the relevance of document $D_i$ to what the user is looking for. There are various methods for quantifying the weight: boolean, linear, and discrete. The boolean method allows the user to give the value 1 for relevant document and 0 for non-relevant document. The linear method allows the user to give any value in a certain range (i.e. 0 to 100, -50 to 50, etc.). The discrete method allows the user to choose from a discrete number of value choices (i.e. -50/totally non-relevant, -25/somewhat non-relevant, 25/somewhat relevant, 50/highly relevant). I chose to use the discrete method in SeRIF. Primarily, this is because with the binary method, the user can only identify a document as relevant or non-relevant. However, the result of searches usually return relevant documents with different degrees of relevancy. Some documents are very relevant but some are only somewhat relevant. The binary method does not allow these differences to be expressed. The linear method, on the other hand, is too specific. The user will most likely have difficulty determining the exact relevancy value for each document. Although it is fairly easy for the user to determine if one document is more relevant than another, it is often hard to know the exact degree of
relevancy difference. This makes it hard for the user to assign a specific value, if the linear method is used.

3. Let $t_j$ be a term in document $D_i$, where $j = 1, 2, \ldots, n$.

4. Let $w_{ij}$ be the weight of term $t_j$ in document $D_i$. $w_{ij}$ is calculated using the 
   - Term Weighting Process described in Section 2.4.

5. Let $R_j$ be the relevancy indicator of term $t_j$ across all documents retrieved. $R_j$ 
   is calculated as:

$$
R_j = \sum_{i=1}^{N} (W_i \times w_{ij})
$$

We need to understand what $R_j$ really measures here, since $R_j$ determines how 
well the newly formed query works when it is used in the next round retrieval 
process. There are two components that affect the value of $R_j$: $W_i$ and $w_{ij}$. 
$W_i$ indicates how relevant the document $D_i$ is as a whole to what the user is 
searching for. A high or positive value $W_i$ means that document $D_i$ is a relevant 
document. A low or negative value means the document is not relevant. $w_{ij}$ 
measures the degree of importance of term $t_j$ in document $D_i$. High $w_{ij}$ tells us 
that term $t_j$ is an important term inside document $D_i$. This also means that 
term $t_j$ "carries" the idea/thesis of document $D_i$, therefore, term $t_j$ is a good 
distinguishing term for document $D_i$. Naturally, if $W_i$ is high/positive and $w_{ij}$ 
is also high, we should probably add the term $t_j$ into the new query. However, 
it is possible for the term $t_j$ to have a heavy weight in a document(s) other than 
document $D_i$. Another document(s), $D_{i+1}$, might have a low/negative $W_{i+1}$ 
value. Thus, we do not want to add terms that are significant to both relevant 
and non-relevant documents to the new query, as these terms might caused the
retrieval of more non-relevant documents. Thus, we take the sum of $W_i \times w_{ij}$ over all the documents returned from the initial search. This method helps us to find the best term(s) that distinguishes the relevant documents without also distinguishing the non-relevant documents.

6. Suppose we want to add $m$ of the best distinguishing terms to the original query to form a new query. $m$ is usually a number smaller than 10, since adding too many terms might cause recall to become too low to be useful. We sort, in a descending order, every term $t_j$ based on their corresponding $R_j$ measurement, and take the $m$ highest values from the list.

7. Given the list of $t_1, t_2, ..., t_m$ and the initial query $Q_0 = q_1, q_2, ..., q_s$. Where
   - $q_1, q_2, ..., q_s$ are terms in the initial query and $s$ is the number of terms in the initial query. Then the new query $Q_1$ is:

\[
Q_1 = Q_0 + \sum_{k=1}^{m} t_k
\]

It is important to note that the relevancy feedback is an iterative method. That means the query resulted from this first iteration $Q_1$ can be used to generate a new query $Q_2$, $Q_2$ can be used to generate $Q_3$ and so on.

I believe that by applying this kind of relevancy feedback method we can get new queries that will lead to retrieval of more relevant documents. In the following chapters, I will describe my implementation of this method and present the results from using the program for real retrieval processes.
Chapter 4

Proposed Solution And Implementation

4.1 Proposed Solution

In Chapter 3, I described the relevancy feedback method that can be used to incorporate user-given information into a query. The goal of the relevancy feedback method is to generate a new query that, when used to retrieve documents from the WWW collection of documents, will return a higher precision result.

In the traditional information retrieval field, the relevancy feedback method has been proven to improve the precision of high-recall searches by up to 50 percent, and the precision of low-recall searches by up to 20 percent[9]. As we know from the paper by Brian Pinkerton[2], most WWW based search gives a high-recall result. This means that the relevance feedback method is a well-suited method for increasing the precision of the WWW search result.

The aim of this thesis is to build a system that will allow the integration of the relevancy feedback method into a WWW search system. WWW search systems are
a particularly good fit to the relevancy feedback method since they allow users to describe an initial query. WWW search systems also return a set of documents as the result of those queries. The set of initial documents and the initial query are exactly the two ingredients needed in order to apply the relevancy feedback method.

My system works as follows:

1. The user submits a query to a WWW search system.

2. The WWW search system retrieves a list of documents based on the given query. This list contains both relevant and non-relevant documents.

3. The first \( x \) documents listed in the list of documents are retrieved from their URL sites, where \( x \) is an arbitrary number decided ahead of time. For this thesis, I used 10 as the value of \( x \).

4. Every retrieved document is indexed using the indexing process described in Section 2.2.

5. The list of the \( x \) retrieved documents is then presented to the user.

6. The user visits every document on the list and determine how relevant that particular document to what he/she is looking for. The user gives his/her relevancy judgement by choosing from available list of values. The values are highly relevant, somewhat relevant, somewhat non-relevant, and totally non-relevant. The numerical numbers assigned to those four values are 50, 25, -25, and -50, respectively.

7. Based on the value of relevancy given by the user to each document, the precision is calculated. If the document’s relevancy value is highly relevant or
somewhat relevant, the document is considered a relevant document, otherwise it is considered non-relevant. The precision is:

\[
\text{Precision} = \frac{\text{number of relevant documents}}{10}
\]

8. A new query is formed using the method described in Section 3.2. The relevancy feedback method is applicable because we have the documents' relevancy values as well as the documents' terms index.

9. The new query is submitted to the WWW search system and the process goes back to step 2.

Steps 2 - 9 described above can be repeated until the precision becomes one (all documents in the subset list are relevant), the user terminates the process, or the list of returned documents becomes empty.

4.2 System Implementation

I have implemented the solution proposed in Section 4.1 into a system called SeRIF (Search Refinement Incorporating relevancy Feedback system). A graphical representation of SeRIF's components is shown in Figure 4.1.

There are seven components shown in the graph. The first component of SeRIF is the human user. The user is considered as one component of SeRIF because he/she is actively participating in the process of information finding. The user does not just provide a query, then sit back and wait until SeRIF returns a final result. The user is involved, instead, in the process by giving his/her feedback on every (sometimes intermediate) result returned by SeRIF. It is this feedback, given by the user, that makes it possible to do the refined query formulation in the first place.
Figure 4.1: Components of SeRIF (Search Refinement Incorporating relevancy Feedback System)
The Internet Workspace component of SeRIF is the WWW itself, i.e. that “big database” containing all Web pages. The Internet Workspace is where the information that the user wants to find resides.

The third component of SeRIF is the Infoseek\textsuperscript{[tm]} Search System\cite{12}. Infoseek is one of the search systems available on the WWW. The decision to use Infoseek is primarily based on personal preference, but the techniques incorporated in SeRIF could be used with any WWW searching system. We use Infoseek with both the initial query and the automatically formed query to find a list of documents relevant to the query. The user of SeRIF might not know the actual underlying WWW search system that is used, because the user only interacts with SeRIF’s user interface. Additional information about Infoseek can be found at http://guide-p.infoseek.com/. It may be possible, in the future, to allow the user to pick one of several popular WWW search systems that he/she wants to use.

The previous three components of SeRIF are considered indirect components. They are not implemented as code inside SeRIF. The remaining components, however, are implemented using the Perl programming language and the HyperText Markup Language (HTML). The Document Viewing page, SeRIF Initial Query page, and SeRIF User’s Feedback page, are members of the same component, that is SeRIF User Interface component. The User’s Query Database is a database that contains the terms along with their corresponding weights from every document that SeRIF uses for the query reformulation process. These documents are those that the user looks at and assigns relevancy scores. SeRIF User Interface component, the User’s Query Database as well as the other two components will be described in more detail in section 4.2.1 - section 4.2.4.

Meanwhile, I would like to describe the two possible protocols for implementing
SeRIF, namely the Common Client Interface (CCI) protocol and the Common Gateway Interface (CGI) protocol. CCI is an extension protocol to NCSA Mosaic that allows programmers to create client site software that interacts with NCSA Mosaic[3]. The client site software can be written in either the C or Perl programming language. The graphical user interface for this kind of software can be created using X-Motif or Tk. The web browser functions as the connection between the client program and the rest of the WWW, and as a way to view the retrieved Web documents. The advantage of this approach is that a single user can run the client program at his/her site at any time. The user does not have to depend on a particular server site to be up, since the client program is run locally. Of course, the CCI implementation of SeRIF still needs to use Infoseek, or some other WWW search system, and requires that system’s server to be up and taking the request. There are two major disadvantages of the CCI approach. The first is that it requires different binary files for different client platforms (for a C implementation), or it requires that the client system has a Perl interpreter. The second disadvantage is that this kind of implementation will always need to build the user’s query database. This process consumes a lot of time (and is currently a bottleneck in SeRIF).

The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers[4]. With the CGI implementation of SeRIF, the program is located on a particular server site. The user of a CGI-based implementation must connect to the server site, via a URL link, to use SeRIF. The program gets user input from the Web browser and displays output to the Web browser using the specified CGI protocol. The disadvantage of the CGI approach is the time of availability to the user. When the SeRIF server site is down or busy, the user cannot use SeRIF. There are three advantages of using CGI. Since
the program is running on the server site, only the server need to have the compiler or the interpreter. The second advantage of CGI is that the SeRIF program can be written in any programming language (not only C and Perl) that supports standard input and standard output, and CGI is supported by many more web browsers (not just NCSA Mosaic). The third advantage of CGI is that it would be possible to incorporate SeRIF into an existing WWW search system (like Infoseek). By using an existing search system, SeRIF does not need to build the user's query database since the documents' term indices are already built and stored in the search system database.

As we can see, the strength of CCI is the weakness of CGI and vice versa. However, I chose to use the CGI protocol for the following three reasons:

1. As mentioned earlier, the process of building the user's query database is a bottleneck for SeRIF. Since it may be possible to eliminate this bottleneck in the future using the CGI implementation, the use of CGI is preferred.

2. Since the WWW is built to be a client/server system, it may be better to use the same overall architecture (i.e. client/server based) for SeRIF.

3. The progress on CCI has slowed down a lot since the beginning of 1996. The reason is unclear; possibly many people have lost interest in the CCI approach to running client programs in favor of the Java applet approach. Note: It may be possible, in the future, to build a SeRIF-like system as a Java applet. However, Java was too immature to consider at the time this work was done.

4.2.1 SeRIF User Interface

The query server module of a typical WWW search system is the interface between the users of that search system with the database of that search system. As mentioned
before, the query server module consists of two parts: the front end and the back end. The front end is the user interface of the whole system. The user interface is the homepage of the WWW search system, where the user will enter the query and view the result of the query. The user interface of the Infoseek system is what the users see when they visit the http://guide-p.infoseek.com/ [12] site, as well as the page(s) that contain all the links returned as the search result.

SeRIF needs to pre-process both the information given by the user (query and relevance ratings) and the information returned as the search result (result filtering). For these reasons, SeRIF needs its own user interface so SeRIF can process what is given by the user and filter what is given by the WWW search system.

The user interface of SeRIF is basically a set of web pages, written in HTML. The SeRIF user interface consists of three types of pages, as shown in Figure 4.1. The first is SeRIF initial query page. When users visit the SeRIF homepage, they are presented with this page through their web browser. This page allows the users to enter an initial query. Once the initial query is submitted to SeRIF, SeRIF will launch a request to Infoseek to get a list of document URL’s based on the initial query. A specified number of top entries from the list are selected, and SeRIF generates a dynamic HTML file that displays those URL’s. This page is the SeRIF user feedback page. On the SeRIF user feedback page, alongside every entry, there is relevancy rating for that entry. If the user clicks on one of the links shown on the SeRIF user feedback page, a new web browser opens, the document corresponding to the link is fetched and the document’s content is shown in the newly opened window. This is the document viewing page.

There are two browser windows used to display the three types of user interface. The first browser window is used to display the SeRIF initial query page and the
SeRIF user feedback page. The second browser window is used to display the document viewing page. The reason for using two windows is for side-by-side visibility. Using SeRIF, the user can view a source document and give a score to the document at the same time without digging through the history list.

4.2.2 SeRIF Indexing System

SeRIF does the indexing for the ten documents presented to the user for their relevancy rating. After submitting a query (either the initial or the automatically generated query) to the Infoseek, SeRIF receives back a list of documents (in terms of URL information). SeRIF takes the first ten of the URL’s in the list since they reflect the ten most relevant documents according to the Infoseek. SeRIF then accesses the Internet Workspace and fetches each of those ten documents. The indexing is then done to each document, using the vector space method similar to the indexing method of a typical WWW search system, as described in Chapter 2.

For every document, this is the process that SeRIF goes through to build the document’s terms index:

1. The raw HTML document is fetched from its URL site. Only the text/html part of the document is fetched. All other parts such as images or sound in the document are ignored. The fetched document contains all the HTML tags.

2. The HTML tagged document is cleaned up by removing all the HTML tags. This step creates a regular ASCII document.

3. The ASCII document is parsed and the non-content words are stripped from the document. The lists of non-content words used by SeRIF are shown in Table 2.1 and Table 2.2.
4. Every remaining word in the document is stripped down to its morphological root using the stemming algorithm described in Section 2.3. The result of the stemming algorithm is the stem of the word. The collection of word stems from the document are the collection of terms contained in that document. SeRIF keeps track of every word and its corresponding term in a separate database. This original word database is used later on by SeRIF query reformulation system to find the original word to be added to the new query.

5. The term weighting process, as described in Section 2.4 is applied to the terms generated in the previous step.

6. Every term in the document along with its weight is recorded in the document term database. This database is used by the query reformulation system to find the best term(s) to be used to generate a new query.

4.2.3 User's Query Database

The user's query database is a combination of two databases: The original word database and the document term database. These two databases are products of SeRIF indexing system. The user's query database is not a fixed database. The database is destroyed after it is used to formulate a new query and a new list of documents is retrieved from the Infoseek result. A new user's query database is built using the documents found in that new list, so the user's query database always contains the information about the specified documents that the user is working with (i.e. assigning relevancy measures).

The original word database contains every word found in the ten documents along with the word's corresponding stem/term. The document term database is actually a set of databases, one for each document. Each of these contains every term found
in that particular document along with the term's weight.

The user's query database is used in SeRIF query reformulation system to find the best term(s) to form a new query.

4.2.4 SeRIF Query Reformulation System

The SeRIF Query Reformulation System is the "central brain" of the whole SeRIF. It coordinates the works of all other components of SeRIF. It works as follows:

1. SeRIF query reformulation system gets the user's given query.

2. Infoseek is contacted to get the list of documents relevant (according to the Infoseek) to the initial query.

3. A specified number of documents on the list are selected. The number of documents selected at a time by SeRIF is ten.

4. SeRIF query reformulation system builds a dynamic HTML page that contains the list of the documents selected in the previous step. This dynamic HTML file also contains the relevancy measurement choices for each entry in the list. SeRIF uses a discrete method for measuring relevancy. There are four values that are possible for each document. The values are highly relevant (i.e. 50), somewhat relevant (i.e. 25), somewhat irrelevant (i.e. -25), and totally irrelevant (i.e. -50).

5. The dynamically built HTML page is displayed in the Web browser window. At the same time, the SeRIF query reformulation system forks a new process. This new process performs the indexing as described in section 4.2.2.
6. The user gives his/her feedback for each document listed in the dynamically built HTML page. The user also gives the number of words to add to the newly formed query. The page is then submitted back to SeRIF.

7. Upon receiving the page, the SeRIF query reformulation system extracts out the user’s given relevancy feedback value for each document.

8. The SeRIF query reformulation system then calculates the precision of this retrieval process based upon the feedback values. If the value of a document is highly relevant or somewhat relevant, the document is considered a relevant document, otherwise it is considered as a non-relevant document.

9. Using the documents’ relevancy feedback values along with the user’s query database, the SeRIF query reformulation system can run the relevancy feedback method described in Section 3.2.

10. The previous step results in a list of word(s). These words are added to the initial query to form a new query.

11. This new query is used to retrieve a new list of relevant documents using the Infoseek.

12. This whole process can be repeated until the precision becomes one (i.e. all ten documents are relevant), the user chooses to stop the process, or the recall becomes 0 (no documents are returned).

A flowchart representation of SeRIF query reformulation system is shown in Figure 4.2.
Start

Get Initial Query from the user

Activate Infoseek using the given query

Take the highest ranked URL's from Infoseek result

Create a user feedback page using the highest ranked URL's

Display page on the browser page

Get the relevancy measurement value from the page submitted by user

Calculate the Precision

Precision = 1 or Recall = 0

No

Yes

Add new terms to query

User choose to continue

Apply the Relevancy Feedback Method

Stop

Figure 4.2: Flowchart representation of SeRIF Query Reformulation System
4.3 An Example

The easiest way to see how SeRIF works is with an example. Here, I will present a typical query result refinement session using SeRIF. A Web server site is set up to act as a SeRIF server. (This SeRIF server site is only run when I am running tests, and is not currently open for public access.) The first step needed in order to use SeRIF is to connect to the SeRIF server site. This is done by opening an URL connection to the SeRIF server. Opening SeRIF’s URL will take us to the SeRIF homepage, as shown in Figure 4.3.

Assume, for this example, that a user is looking for information about the Java island (an island located in the Indonesian archipelago). The user will type “java island” into the text entry field available on the Figure 4.3, and hits the Enter key to submit. SeRIF will take in the page through CGI, and extract out the user query. SeRIF will then connect to Infoseek and ask Infoseek to do a query retrieval using the user defined query.

Infoseek returns a list that contains URL’s to documents that match the “Java island” query. SeRIF takes this list and extracts out the ten highest ranked URL’s and uses them to create the user feedback page. This dynamic HTML page, as shown in Figure 4.4, contains the following:

1. The ten highest ranked URL’s extracted from the original list returned by Infoseek.

2. The choices of relevancy value for each document. The default value is the “Somewhat relevant” choice.

3. The field to enter the number of words to add to the original query.
Figure 4.3: SeRIF Homepage
Figure 4.4: Dynamic HTML page showing result of the user defined query
The user can click on any of the URL's shown in the dynamic HTML page. This will take him/her to the URL's site and show him/her the document. In order to make document viewing easier, SeRIF use an additional browser window to display the document. Using an extra window, we can view the document as well as the user feedback page side by side, as shown in Figure 4.5.

The user can view each of the selected documents and assign a relevancy value to each. In this case, it turns out that only two of the ten documents are relevant to the "Java island" that the user is looking for. The other eight documents talk about the Java programming language (programming language developed by Sun Microsystem, Inc.) or the Java company (a tourism company located in Hawaii). The user wants the system to form a new query for us by adding one word into the original query. Once the user is done assigning relevancy values and setting the number of words to add, the user submits the dynamic HTML page back to SeRIF.

SeRIF takes our response and calculates the precision on the set of ten documents retrieved using the user's defined query. Since only two documents are relevant (one Very relevant and the other Somewhat relevant), the precision is 0.2 (2 out of 10). Since the precision is not one and we want to continue the process, SeRIF figures out the best word to add to the query. It turns out that the best word being added to the query is the word "Indonesian." SeRIF adds this word to the query and uses the list "java island indonesian" as the new query to send to Infoseek. Infoseek returns a new list containing URL's of documents that are relevant to the newly formed query. SeRIF extracts out the ten highest ranked URL's from this new list and dynamically creates another user feedback page. This new dynamic HTML page is shown in Figure 4.6.

This newly formed user feedback page contains:
Java is the political, geographic and economic centre of the Indonesian archipelago. It's a relatively small island, (only half the size of England) but has a population of 112 million, accounting for 55% of the country's total population. The island is long and narrow in shape, with a string of volcanic mountains punctuating its spine. It was on Java that the Hindu-Buddhist empires reached their zenith, producing architectural wonders such as Borobudur and Prambanan. When Islam came to the island in the 15th century, it absorbed rather than erased local cultures, leaving Java with a mish-mash of historic influences and religions. A strong consciousness of ancient religious and mystical thought carries over into present-day Java, providing a bulwark against wholesale modernisation.

Much of the young republic's history was hacked out of Javanese soil - including the major independence battles and the emergence of the two strongest political parties - and today the island plays an extraordinarily dominant role in Indonesia. It has been said that Soeharto is much more a Javanese king than an 'elected' president, and that Indonesia is much more a Javanese kingdom than a republic. To a large extent, the rebellions of the Sumatrans, Minahassans and Ambonese in the 1950s and 1960s were rebellions against Javanese domination of the archipelago.

The island is certainly the most developed in the Indonesian archipelago, but despite its political and economic primacy it is still struggling with the twin demons of overpopulation and poverty. The visitor is confronted by a society in transition - one which is keen to embrace the benefits of modernity but determined not to lose its heritage in the process. Thus fast-food joints, shopping malls, satellite TV and the other material accoutrements of the West live cheek by jowl with a vibrant traditional culture.
Thank you for using SeRIF.

You've identified 2 documents as relevant. The precision is 0.2.

The refined query is java island indonesian

The added words number is 1

*Give your score of relevance for these links:

Lonely Planet — Destination: Java

—— http://www.lonelyplanet.com.au/dest/java.htm (Score 56, Size 18K)
Your given relevance score:

✓ Very relevant ✗ Somewhat relevant ✓ Somewhat irrelevant □ Totally irrelevant

Lonely Planet — Destination: Indonesia

—— http://www.lonelyplanet.com.au/dest/indo.htm (Score 56, Size 33K)
Your given relevance score:

✓ Very relevant ✗ Somewhat relevant ✓ Somewhat irrelevant □ Totally irrelevant

THE HISTORY OF INDONESIA

—— http://www.pacs.org/history.html (Score 55, Size 66K)
Your given relevance score:

✓ Very relevant ✗ Somewhat relevant ✓ Somewhat irrelevant □ Totally irrelevant

Travel Information of Indonesia

—— http://www.travel.indonesia.net/autointro/html (Score 55, Size 52K)
Your given relevance score:

✓ Very relevant ✗ Somewhat relevant ✓ Somewhat irrelevant □ Totally irrelevant

Indonesia Net — Discover Indonesia (index)

—— http://www.go-indonesia.com/indonesia/index.HTML (Score 55, Size 5K)
Your given relevance score:

✓ Very relevant ✗ Somewhat relevant ✓ Somewhat irrelevant □ Totally irrelevant

INDONESIA: Sightseeing

—— http://www.go-indonesia.com/sightseeing/interestright.html (Score 55, Size 4K)
Your given relevance score:

✓ Very relevant ✗ Somewhat relevant ✓ Somewhat irrelevant □ Totally irrelevant

Figure 4.6: Result of the automatically formed query
1. The precision of the documents in the previous list.

2. The newly formed query.

3. The top URL's extracted from the list returned by Infoseek.

4. The choices of relevancy value for each document. The default value is the “Somewhat relevant” choice.

5. The field to enter the number of words to be added to the newly formed query.

In this case, it turns out that nine out of the ten documents retrieved contain some information about the island of Java in Indonesia. The one document that does not contain information about the island of Java is a blank document (the document has moved to another unknown Web site). However, judging from the title of the Web document, the original document indexed by Infoseek should contain information about the island of Java in the right context.

The user can continue the process by assigning a relevancy value for each document and also the number of words to add to the query. What the user will get back is another dynamic HTML page like Figure 4.6, but with different values for the precision, the query, and the top URL's. The user can continue this process of refining the search as many times as he/she would like to, until the precision reaches one, the user chooses to quit, or the query becomes so specific that no documents are found.
Chapter 5

Testing: Set Up and Results

5.1 Test Set Up

With SeRIF, I am trying to increase the precision of the search result of a typical WWW search system. As described in Section 3.1, there are three problems that can cause the search system to return low precision results. SeRIF is intended to correct the second and the third problems. Although both of these two problems result in low precision retrievals, the nature of the non-relevant documents retrieved are very different. The first problem results in the retrieval of documents that carry the words specified in the query (correct textual identity), but these words appear in a wrong context (wrong semantic identity). The second problem results in the retrieval of documents that carry the correct textual and semantic identities of the query words. However, the documents returned often do not carry the specific information that the users really want. This is due to the users’ inability to supply specific words that described what they are really looking for. Essentially, we can think of the problems as types of miscommunication between the user and the search system. In first of the two, the search system fails to distinguish the semantics that the user is trying
to communicate. In the second problem, the user fails to give an adequately specific request to the search system.

SeRIF is built as a layer between the user and the WWW search system. SeRIF’s aim is to “correct” these miscommunications. Since the nature of the miscommunications is different, we need to set up two different sets of test cases. The first set of test cases contain queries with words that have two or more semantic identities. For each test case, I chose one of the semantic identities and ran SeRIF using the query, to see if SeRIF can distinguish the semantics in mind. For the second set of test cases, I set up queries that represented some general information about a topic. For each test case, I also wrote down the specific information that I really had in mind. I ran SeRIF with the general information query and then used SeRIF’s relevancy feedback process attempt to guide me to the specific information wanted. The evaluation on how well SeRIF performs in both set of test cases is measured using the precision improvement that returned from every round of relevancy feedback.

For each of those two test sets, I also wanted to test whether the number of words added to the query in each iteration will affect the amount of improvement in precision. I wanted to see whether adding one word per iteration caused a different improvement in precision as compared to adding three words per iteration.

Therefore, there are four sets of test cases:

1. Set I: Finding correct semantics by adding one word per iteration.
2. Set II: Finding correct semantics by adding three words per iteration.
3. Set III: Finding specific information by adding one word per iteration.
4. Set IV: Finding specific information by adding three words per iteration.
For each query, SeRIF is used to run the refinement process until the earliest of these five events happened:

1. The precision is equal to 1.
2. The precision is equal to 0.
3. The recall is equal to 0 (i.e. the query does not result in any documents).
4. No word can be added to the query. It is possible that the relevant documents are short documents that only contain very few significant words. After a few iterations, it might be possible that no word can be added.
5. The relevancy feedback process has been iterated five times. Theoretically, SeRIF’s query reformulation process can be run as many times as the user wants to. However, if the user needs to run the process more than five times, it defeats SeRIF’s purpose of providing fast and precise results.

5.1.1 Test Cases

There were four test sets as described earlier. Each set contains five queries. Some of these queries were contributed by other computer science students at the University of Montana. These are the queries, grouped by sets:

Set I:

1. Query: ‘‘java island.’’
   The correct semantic identity is the island of Java located in the Indonesian archipelago.

2. Query: ‘‘ada organization.’’
   The correct semantic identity is the American Disability Act related organization.

3. Query: ‘‘nick wilde.’’
The correct semantic identity is information related to Dr. Nick Wilde, a professor at the University of Montana.

4. Query: "'roosevelt.'"
The correct semantic identity is president Theodore Roosevelt.

5. Query: "'jordan.'"
The correct semantic identity is the country of Jordan located in the Middle East.

Set II:

Queries are the same as Set I, except three words are added to the query in every relevancy feedback iteration.

Set III:

1. Query: "'olympic games.'"
The specific information is the sporting events in the Olympic.

2. Query: "'montana park.'"
The specific information is the Glacier National Park.

3. Query: "'washington map.'"
The specific information is the map of places in Washington D.C.

4. Query: "'southeast asia country.'"
The specific information is the country of Indonesia.

5. Query: "'melbourne university.'"
The specific information is computer science related information at the University of Melbourne.

Set IV:

Queries are the same as Set I, except three words are added to the query in every relevancy feedback iteration.

5.2 Test Result
Set I:

1. Initial query: ‘‘java island.’’
   - Precision: 0.2
   - Details: * Two documents contain the correct semantic identity.
     * Four documents contain information on the Java programming language.
     * Four documents contain information on a travel agent named Java.

First round:
   - Word added to query: ‘‘indonesian.’’
   - Precision: 0.9
   - Details: * Nine documents contain the correct semantic identity.
     * One document is empty.

Second round:
   - Word added to query: ‘‘jakarta.’’
   - Precision: 1.0
   - Details: * All documents contain the correct semantic identity.

2. Initial query: ‘‘ada organization.’’
   - Precision: 0.2
   - Details: * Two documents contain the correct semantic identity.
     * Two documents contain information about the town of Ada in Oklahoma.
     * Six documents contain information about the ada programming language

First round:
   - Word added to query: ‘‘employment.’’
   - Precision: 1.0
   - Details: * All documents contain the correct semantic identity.

3. Initial query: ‘‘nick wilde.’’
   - Precision: 0.4
   - Details: * Four documents contain the correct semantic identity.
     * Two documents contain information about Ian Wilde.
     * One document contains information about the Wilde flower family tree.
     * One document contains information about an internet site called Wilde.
     * Two documents contain information about Oscar Wilde.

First round:
   - Word added to query: ‘‘lake.’’
   - Precision: 0.3
   - Details: * Three documents contain the correct semantic identity.
     * One document contain a TV station in the Netherland.
* Two documents contain information about Oscar Wilde.
* One document contains information about Ian Wilde.
* One document contains information about Stuart Wilde.
* One document contains information about Nick van Exel.
* One document contains information about Steve Nicks.

Second round:
- Word added to query: ‘‘miles.’’
- Precision: 0.3
- Details: * Three documents contain the correct semantic identity.
  * One document contains information about Eddie Wilde.
  * One document contains information about weather.
  * One document contains information about Steve Nicks.
  * One document contains information about the X-Files.
  * One document contains information about storyteller audio book.
  * One document contains information about Kim Wilde.
  * One document contains information about Oscar Wilde.

Third round:
- Word added to query: ‘‘worse.’’
- Precision: 0.3
- Details: * Exactly the same as the third round.

Fourth round:
- Word added to query: ‘‘rain.’’
- Precision: 0.3
- Details: * Exactly the same as the third round.

Fifth round:
- Word added to query: ‘‘cycle.’’
- Precision: 0.3
- Details: * Exactly the same as the third round.

4. Initial query: ‘‘roosevelt.’’
- Precision: 0.3
- Details: * Three documents contain the correct semantic identity.
  * Two documents contain information about Franklin Roosevelt.
  * Two documents about Roosevelt college.
  * One document about Roosevelt Dam.
  * Two documents about Theodore Roosevelt high school.

First round:
- Word added to query: ‘‘theodore.’’
- Precision: 0.7
- Details: * Seven documents contain the correct semantic identity.  
  * One document about Roosevelt Dam.  
  * Two documents about Theodore Roosevelt high school.

Second round:  
- Word added to query: ‘‘wilson.’’  
  - Precision: 0.8  
  - Details: * Eight documents contain the correct semantic identity.  
    * One document contain information about Franklin Roosevelt.  
    * One documents about Theodore Roosevelt high school.

Third round:  
- Word added to query: ‘‘world.’’  
  - Precision: 0.8  
  - Details: * Eight documents contain the correct semantic identity.  
    * One document contain information about Franklin Roosevelt.  
    * One documents about Theodore Roosevelt high school.

Fourth round:  
- Word added to query: ‘‘presidency.’’  
  - Precision: 0.9  
  - Details: * Nine documents contain the correct semantic identity.  
    * One document contain information about Franklin Roosevelt.

Fifth round:  
- Word added to query: ‘‘america.’’  
  - Precision: 0.9  
  - Details: * Exactly the same as round four.

5. Initial query: ‘‘jordan.’’  
- Precision: 0.4  
- Details: * Four documents contain the correct semantic identity.  
  * Four documents contain information about Robert Jordan.  
  * Two documents contain information about Michael Jordan

First round:  
- Word added to query: ‘‘travel.’’  
  - Precision: 0.8  
  - Details: * Eight documents contain the correct semantic identity.  
    * Two documents contain information about Jordan & Jordan, Inc. travel agent.

Second round:  
- Word added to query: ‘‘israel.’’  
  - Precision: 0.7  
  - Details: * Seven documents contain the correct semantic identity.
* Three documents contain information about the river of Jordan in Israel.

Third round:
- Word added to query: 'http.'
- Precision: 0.7
- Details: * Seven documents contain the correct semantic identity.
  * Three documents contain information about the river of Jordan in Israel.

Fourth round:
- Word added to query: 'www.'
- Precision: 0.0
- Details: * No document returned from Infoseek.

Set II:

1. Initial query: 'java island.'
   - Precision: 0.2
   - Details: * Two documents contain the correct semantic identity.
     * Four documents contain information on the Java programming language.
     * Four documents contain information on a travel agent named Java.

First round:
- Word added to query: 'indonesian jakarta city.'
- Precision: 1.0
- Details: * All documents contain the correct semantic identity.

2. Initial query: 'ada organization.'
   - Precision: 0.2
   - Details: * Two documents contain the correct semantic identity.
     * Two documents contain information about the town of Ada in Oklahoma.
     * Six documents contain information about the ada programming language

First round:
- Word added to query: 'employment accommodation training.'
- Precision: 1.0
- Details: * All documents contain the correct semantic identity.

3. Initial query: 'nick wilde.'
   - Precision: 0.4
   - Details: * Four documents contain the correct semantic identity.
     * Two documents contain information about Ian Wilde.
     * One document contains information about the Wilde
flower family tree.
* One document contains information about an internet site called Wilde.
* Two documents contain information about Oscar Wilde.

First round:
- Word added to query: "lake miles worse.''
  - Precision: 0.3
  - Details: * Three documents contain the correct semantic identity.
    * One documents contain information about Eddie Wilde.
    * One documents contain information about weather.
    * One document contains information about Steve Nicks.
    * One document contains information about the X-Files.
    * One document contains information about storyteller audio book.
    * One documents contain information about Kim Wilde.
    * One documents contain information about Oscar Wilde.

Second round:
- Word added to query: "rain cycle hill.''
  - Precision: 0.0
  - Details: * No document returned by Infoseek.

4. Initial query: "roosevelt.''
  - Precision: 0.3
  - Details: * Three documents contain the correct semantic identity.
    * Two documents contain information about Franklin Roosevelt.
    * Two documents about Roosevelt college.
    * One document about Roosevelt Dam.
    * Two documents about some Roosevelt high school.

First round:
- Word added to query: "theodore world time.''
  - Precision: 0.8
  - Details: * Eight documents contain the correct semantic identity.
    * Two documents about some Roosevelt high school.

Second round:
- Word added to query: "state chapter wilson.''
  - Precision: 0.0
  - Details: * No document returned by Infoseek.

5. Initial query: "jordan.''
  - Precision: 0.4
  - Details: * Four documents contain the correct semantic identity.
    * Four documents contain information about Robert Jordan.
Two documents contain information about Michael Jordan

First round:
- Word added to query: "'travel israeli guide.'"
- Precision: 0.8
- Details: * Eight documents contain the correct semantic identity.
  * Two documents contain information about the river of Jordan in Israel.

Second round:
- Word added to query: "'http www useful.'"
- Precision: 0.0
- Details: * No document returned from Infoseek.

Set III:

1. Initial query: "'olympic games.'"
   - Precision: 0.2
   - Details: * Two documents contain the specific information about the sporting event in the summer Olympic.
     * Five documents contain information about housing rentals in the Atlanta area for the Olympic.
     * Three documents contain advertisements from companies that support the Olympic.

First round:
- Word added to query: "'sites'"
- Precision: 0.3
- Details: * Three documents contain the specific information about the sporting event in the summer Olympic.
  * Three documents contain the information about the city of Atlanta and its role as the host.
  * Two documents contain advertisements from companies that support the Olympic.
  * Two documents contain information about the 2002 winter Olympic.

Second round:
- Word added to query: "'information'"
- Precision: 0.3
- Details: * Three documents contain the specific information about the sporting event in the summer Olympic.
  * Three documents contain the information about the city of Atlanta and its role as the host.
  * Two documents contain advertisements from companies that support the Olympic.
  * One document contains information about the special
olympic in Canada.

* One document contains information about general sporting events that mention olympic games but it is not related to event in the Atlanta olympic games.

Third round:
- Word added to query: "'1996'"
  - Precision: 0.3
  - Details: * Three documents contain the specific information about the sporting event in the summer olympic.
    * Three documents contain the information about the city of Atlanta and its role as the host.
    * Two documents contain advertisements from companies that support the olympic.
    * One document contains information about the special olympic in Canada.
    * One document is empty.

Fourth round:
- Word added to query: "'Atlanta'"
  - Precision: 0.3
  - Details: * Exactly the same as the third round.

Fifth round:
- Word added to query: "'guide'"
  - Precision: 0.0
  - Details: * No document returned from Infoseek.

2. Initial query: "'montana park.'"
  - Precision: 0.7
  - Details: * Seven documents contain the info about glacier park but mostly just name only.
    * Three documents contain information about other parks in Montana.

First round:
- Word added to query: "'river'"
  - Precision: 0.5
  - Details: * Five documents contain the info about glacier park but mostly just name only.
    * Four documents contain information about other parks in Montana.
    * One document contains information about river in the Madison County.

Second round:
- Word added to query: "'fishery'"
  - Precision: 0.5
  - Details: * Five documents contain the info about glacier park
but mostly just name only.
* Three documents contain information about fishing in Montana.
* Two documents contain information about fishing in general.

Third round:
- Word added to query: ‘‘little’’
  - Precision: 0.0
  - Details: * No document returned from Infoseek.

3. Initial query: ‘‘washington map.’’
  - Precision: 0.4
  - Details: * Four documents contain the specific information of maps of various sites in Washington D.C.
    * One document contains the specific information of maps of various towns in the state of Washington.
    * One document contains the specific information of maps of Washington University in St. Louis.
    * Two documents contain information about University of Washington.
    * One document contains information about a city of Washington in Penn.

First round:
- Word added to query: ‘‘subway’’
  - Precision: 0.0
  - Details: * None of the document talk about map of Washington D.C. Rather they talk about the subway system in Washington D.C. and other cities.

4. Initial query: ‘‘southeast asia country.’’
  - Precision: 0.5
  - Details: * Five documents contain information about Indonesia, but they are not very specific information.
    * Five documents contain information about other information related to Southeast Asia in general.

First round:
- Word added to query: ‘‘list.’’
  - Precision: 0.7
  - Details: * Seven documents contain information about Indonesia, but they are not very specific information.
    * Three documents contain information about other information related to Southeast Asia in general.

Second round:
- Word added to query: 'ed.
- Precision: 0.5
- Details: * One document contains specific information about Indonesia.
  * Four documents contain information about Indonesia, but they are not very specific information.
  * Two documents contain information about other information related to Southeast Asia in general.
  * Two documents talk specifically about the country of Thailand.
  * One document contain specific information about the Philippines.

Third round:
- Word added to query: 'indonesian.'
- Precision: 1.0
- Details: * Two document contains specific information about Indonesia.
  * Eight documents contain information about Indonesia, but they are not very specific information.

5. Initial query: 'melbourne university.'
- Precision: 0.3
- Details: * Three documents contain information about computer science related material at the University of Melbourne.
  * Six documents contain information about other departments at the University of Melbourne.
  * One document contain information about Monash University located in Melbourne.

First round:
- Word added to query: 'ormond.'
- Precision: 0.3
- Details: * Three documents contain information about computer science related material at the University of Melbourne.
  * Six documents contain information about other sites related to the University of Melbourne.
  * One document contains information about a college located in Melbourne, Florida.

Second round:
- Word added to query: 'centre.'
- Precision: 0.3
- Details: * Three documents contain information about computer science related material at the University of Melbourne.
* Four documents contain information about other sites related to the University of Melbourne.
* Two documents contain information related with the city of Ormond in Melbourne.
* One document contain information related to some general education information in Australia.

Third round:
- Word added to query: "facilities."
  - Precision: 0.3
  - Details: * Three documents contain information about computer science related material at the University of Melbourne.
  * Seven documents contain information about other sites related to the University of Melbourne.

Fourth round:
- Word added to query: "improvement."
  - Precision: 0.3
  - Details: * Exactly the same as the third round.

Fifth round:
- Word added to query: "technical."
  - Precision: 0.3
  - Details: * Three documents contain information about computer science related material at the University of Melbourne.
  * Four documents contain information about other sites related to the University of Melbourne.
  * Three document contains information about a college located in Melbourne, Florida.

Set IV:

1. Initial query: "olympic games."
   - Precision: 0.2
   - Details: * Two documents contain the specific information about the sporting event in the summer olympic.
   * Five documents contain information about housing rentals in the Atlanta area for the Olympic.
   * Three documents contain advertisements from companies that support the olympic.

First round:
- Word added to query: "sites information 1996"
  - Precision: 0.3
  - Details: * Three documents contain the specific information about the sporting event in the summer olympic.
* Three documents contain the information about the city of Atlanta and its role as the host.
* Two documents contain advertisements from companies that support the Olympic.
* One document contains information about the special Olympic in Canada.
* One document is empty.

Second round:
- Word added to query: ‘guide network national’
- Precision: 0.0
- Details: * No document returned from Infoseek.

2. Initial query: ‘montana park.’
- Precision: 0.7
- Details: * Seven documents contain the info about glacier park but mostly just name only.
  * Three documents contain information about other parks in Montana.

First round:
- Word added to query: ‘guide river national.’
- Precision: 0.5
- Details: * Five documents contain the info about glacier park but mostly just name only.
  * One document contains information about other parks in Montana.
  * Two documents contain information about fishing in Montana.
  * Two documents contain information about kayaking in Montana.

Second round:
- Word added to query: ‘trails 95.’ Only two words can be added.
- Precision: 0.3
- Details: * One document talk very specifically about glacier national park.
  * Two documents contain the info about glacier park but mostly just name only.
  * Two documents contain information about adventure in the northwest.
  * One documents contain information about fishing in Montana.
  * One documents contain information about fishing in the Northwest.
  * Three documents talk about outdoor recreation in general.
3. Initial query: ‘‘washington map.’’
   - Precision: 0.4
   - Details: * Four documents contain the specific information of
     maps of various sites in Washington D.C.
     * One document contains the specific information of
       maps of various towns in the state of Washington.
     * One document contains the specific information of
       maps of Washington University in St. Louis.
     * Two documents contain information about University
       of Washington.
     * One document contains information about a city
       of Washington in Penn.
     * One document contains information about Mount
       Washington in New Hampshire.

First round:
- Word added to query: ‘‘site national subway’’
- Precision: 0.3
- Details: * Three documents contain the specific information of
  maps of various sites in Washington D.C.
  * Two documents contain general information about
    Washington D.C.
  * Two documents contain information about subways
    in different cities.
  * Three documents contain other unrelated information.

Second round:
- Word added to query: ‘‘data http online’’
- Precision: 0.0
- Details: * No document returned from Infoseek.

4. Initial query: ‘‘southeast asia country.’’
   - Precision: 0.5
   - Details: * Five documents contain information about Indonesia,
     but they are not very specific information.
     * Five documents contain information about other
       information related to Southeast Asia in general.

First round:
- Word added to query: ‘‘list study indonesians.’’
- Precision: 1.0
- Details: * Two document contains specific information about
  Indonesia.
  * Eight documents contain information about Indonesia,
    but they are not very specific information.

5. Initial query: ‘‘melbourne university.’’
   - Precision: 0.3
Details: * Three documents contain information about computer science related material at the University of Melbourne.
* Six documents contain information about other departments at the University of Melbourne.
* One document contain information about Monash University located in Melbourne.

First round:
- Word added to query: ‘‘ormond access centre.’’
- Precision: 0.3
- Details: * Three documents contain information about computer science related material at the University of Melbourne.
  * Four documents contain information about other sites related to the University of Melbourne.
  * Two documents contain information related with the city of Ormond in Melbourne.
  * One document contain information related to some general education information in Australia.

Second round:
- Word added to query: ‘‘facilities improvement technical.’’
- Precision: 0.3
- Details: * Three documents contain information about computer science related material at the University of Melbourne.
  * Four documents contain information about other sites related to the University of Melbourne.
  * One document related to an internet service in Australia.
  * One document related to a computer site in Africa.
  * One document contain information related to some general education information in Australia.

Third round:
- Word added to query: ‘‘facilities improvement technical.’’
- Precision: 0.0
- Details: * Query is too long for Infoseek to handle.
Chapter 6

Discussion

6.1 Test Cases Evaluation

The discussion is separated into four subsections, each dealing with one of the test sets.

6.1.1 Set I

The result of this set of tests can be classified into two groups. The first group consists of four cases (cases 1, 2, 4, and 5) that show huge precision improvements in the first round of the relevancy feedback loop. The second group consists of only case 3, that shows a small decrease of precision in the first round of relevancy feedback loop. For cases 1, 2, 4, and 5, the one additional word added by SeRIF seemed the distinguish the correct semantics of the words found in the initial queries. Although the first iteration of cases 1, 2, 4, and 5 give a similar amount of precision improvement, the rest of the iterations resulted in some very different amount of precision improvement. A more specific description of how precision changes for each of these five cases are as follows:
1. In the first round of refinement of Case 1, SeRIF found the word “Indonesian.” This additional word eliminates the other two semantic identities of Java island that occur in the initial retrieval. The word “indonesian” specified that the semantic identity of the words “Java island” is the island of Java in Indonesia. The precision resulting from the first round is 0.9. The only document that doesn’t contain the java island in the correct semantic identity is an empty document. The title of the page, however, reflects that the page probably contained information about Indonesia. In the second round of Case 1, SeRIF found a more discriminating word to add to the query. Since the added word is “jakarta” which is a city in the island of Java in Indonesia, the semantic identity of “java island” became clearer. The precision of retrieval using “java island indonesian jakarta” is one.

2. The initial retrieval of Case 2 resulted in documents with three different semantic identities of “ada organization.” Two of these documents contain the correct semantic identity, that of the American Disabilities Act. The other documents related to the Ada programming language and the town of Ada in Oklahoma. The first round of refinement added the word “employment.” This additional word retrieved ten relevant documents. “Employment” might not be a very distinguishing word specifically, since it is possible to find the word “employment” in relation to the Ada programming language or the Ada town in Oklahoma. However, it seems that the word “employment” has a closer relationship to the American Disability Act identity than to the other two identities of ada. The second round refinement is not needed for Case 2 since the first round refinement already resulted in 1.0 precision.
3. Case 3 results in a very different result compared to the other four cases. The first iteration of Case 3 decreases the relevant documents number by one. The following refinement iterations do not add any relevant documents at all. The non relevant documents, however, keep changing over each iteration. The reason for the failure in this case might be caused by the very limited number of documents that contain information about “Nick Wilde.” It might also be the case that this is a poor query for SeRIF to be executed on. “Nick Wilde” is really not a subject per se, but rather represents a collection of documents owned by an individual on the WWW. As such there is no precise semantic identity associated with the words “Nick Wilde” on the WWW.

4. The first iteration of Case 4 increases the precision from 0.3 to 0.7. The additional word found in the first iteration immediately identified the correct semantics. The word “Theodore” distinguished the information about Theodore Roosevelt from the other Roosevelts. However, this additional word does not distinguish documents that contain information about the president Theodore Roosevelt with the other material about subjects named after the president (i.e. school, dam, etc.). The remaining iterations (2 through 4) took out documents containing information not directly related to President Theodore Roosevelt. These iterations add in words that describe Theodore Roosevelt as related to his position as president. However, iteration two to four also resulted in the inclusion of a document related to Franklin D. Roosevelt. The inclusion of this document is due to the fact that this document discusses the relationship between Franklin Roosevelt and Theodore Roosevelt. More importantly, this document also contains information about a president (although it is a different president).
5. The first iteration of Case 5 increased the precision from 0.4 to 0.8. The addition of the word "travel" eliminated the other two identities of Jordan found in the initially retrieved documents. As with the word "education" in Case 2, "travel" might also apply to Robert Jordan or Michael Jordan. However, there are likely more documents that talk about traveling in the country of Jordan than traveling done by these two persons. Although the word "travel" eliminates two semantic identities of Jordan, it includes another identity - the Jordan and Jordan travel service. The word added to the query in the second iteration removes the Jordan and Jordan travel service documents. However, in this process it adds in yet another semantic identity of Jordan, that is the river of Jordan in Israel. SeRIF does not seem to be able to identify the difference between the river of Jordan and the country of Jordan.

6.1.2 Set II

The results of queries in this set of test are very similar to the results of the same queries of Set I. However, the final results came faster (in fewer iterations) in this set than in Set I. What follows are more specific descriptions of what happened in each case:

1. The first round of Case 1 finds the words "indonesian jakarta city" which distinguishes the correct semantics and results in a 1.0 precision.

2. The first iteration of Case 2 finds the words "employment accommodation training." These words certainly relate better to the American Disability Act organization than to the Ada programming or the town of Ada. This first iteration also result in a 1.0 precision.
3. The problem that SeRIF has with Case 3 is the same as the problem in the Case 3 of Set I. However, instead of taking more than five iterations to find out that SeRIF cannot improve the precision, SeRIF only need two iterations. When given the query “nick wilde lake miles worse rain cycle hill,” Infoseek does not return any document.

4. The first iteration through Case 4, gives results with 0.8 precision. In the first iteration, SeRIF is able to identify that the correct semantics are Theodore Roosevelt, but it fails to distinguish between Theodore Roosevelt the president and Theodore Roosevelt the school. The second iteration results in an empty list of documents returned by Infoseek.

5. The behavior of SeRIF in Case 5 is very similar to its behavior in Case 4. The first iteration results in a very good precision improvement. However, SeRIF could not distinguish between the country of Jordan and the river of Jordan. The successive iteration, however, also results in an empty document list returned by Infoseek.

6.1.3 Set III

SeRIF does not give as satisfactory results for this set of test cases. Cases 1, 2, 3, and 5 all resulted in low precision results. SeRIF could not find the correct specific words that we were searching for. The words chosen in each relevancy feedback iteration were common words that resulted in the retrieval of some more general information documents. The final precision for case 4, however, is 1.0. In case 4, SeRIF was able to find the specific words that we were looking for. Using these words, SeRIF was able to retrieve more specific documents. Following are specific descriptions about what happened in each cases:
1. The first case attempted to find the sporting events held in the summer Olympics in Atlanta. The initial query given expressed the general idea that we wanted to find some information related to the Olympics games. The first two iterations resulted in the addition of the words “sites” and “information”. These words do not have any specific meaning related to the sporting events of the summer games. The addition of these new words create queries that retrieved more general information concerning the Olympics. Iterations number three and four, specify which specific games we want, namely the Atlanta 1996 Olympics. However, these words still do not express the specific information that we are looking for. The major reason for the failure is because the relevant documents returned by the initial query also contain information about other (somewhat related) topics. For example, the relevant document that contained the sporting events also contained information about the many sites where other Olympics related activities are happening.

2. Case 2 started out with seven out of the ten documents that mentioned about Glacier National Park. However, all of these seven documents also mention other parks in Montana along with various things (river, fishing, camping, etc.) that are related to parks in general. The additional words added by SeRIF are words that describe the activities that can be done in these parks. SeRIF did not find the name of the park (i.e. Glacier Park) that we were actually seeking.

3. The initial retrieval of the Case 3 query found four documents that contain maps of the Washington D.C. area. However, these documents contained the picture with little textual explanation about the map. From the small sets of words found in these map documents, SeRIF picked the word “subway.” This caused the retrieval of documents related to the subway systems in many
different cities. This has taken away even the general idea of finding the map of a city. The main reason for failure in this case is the limitations of a text based indexing system in dealing with a graphical document.

4. Case 4 is the only case that gives a satisfactory result. The success, however, did not come until the third iteration when SeRIF found a very specific word (i.e. Indonesian) to add to the query. This specific word caused the retrieval of the ten relevant document. The success in this case was because of the finding of one document in the second iteration that specifically contains the information about the country of Indonesia.

5. SeRIF does not perform well in Case 5 for the same reason as what happened in Case 2. Instead of finding a specific information, SeRIF found more general information out of the general documents returned by the initial retrieval.

6.1.4 Set IV

The results of this set of test cases are very similar to the result on Set III. However, the final results are achieved in less iterations. The following are more detailed descriptions of what happened in each case:

1. The first iteration of Case 1 resulted in documents that are related to the 1996 Olympics, but most of them are not related to the sporting event of the 1996 summer Olympics in Atlanta. The second iteration resulted in empty list from Infoseek.

2. The first iteration of Case 2 also resulted in the addition of general meaning words. In the second iteration, SeRIF could only add two words because the
relevant documents do not have any more distinguishing words. The weights of the remaining words on the documents are too low.

3. The result of Case 3 is almost the same as Case 1. The first iteration retrieved non specific documents and the second iteration did not retrieve any documents.

4. Case 4 is an exception, since in Case 4, SeRIF was able to find specific information that we are looking for, and it did it through one iteration only.

5. In Case 5, SeRIF kept adding words to the query and retrieved general information documents until the length of the query exceed 80 characters long. This stop the SeRIF iteration because SeRIF cannot accept any query longer than 80 characters.

6.2 General Discussion

In general, we can see that SeRIF worked for Set I and Set II. SeRIF was able to eliminate the wrong semantic identities of the words in the query. SeRIF does this by identifying distinguishing words in the relevant documents. These distinguishing words, when added to the original query, can be used to specify the correct semantic identities of the words in the original query. Thus the newly formed query retrieves a higher precision result.

For Set III and Set IV, however, SeRIF did not perform very well. There are two reasons for the failure:

1. SeRIF, generally, was unable to identify the specific information available in the general information documents. The specific information is available in the documents, but the specific information is not the major content of documents.
So, the most distinguishing words found in the general information document are not those words that can be used to find the specific documents.

2. SerIF does not have a mechanism to correct a wrong path. Once SerIF chose a wrong word to add to the query, there is no mechanism to undo the word addition. So once a wrong path is taken, there is no way to go back.

The speed of SerIF is also not very satisfying. The bottleneck is the indexing system. When the size of the documents are 20K or lower, the speed of the whole system is acceptable. It takes about the same time for the user to read all the documents and assign relevances as for the SerIF indexing system to build the index. However, when SerIF tries to index a document with size greater than 100K, speed becomes an issue. For ten small documents, SerIF takes about 3-10 minutes to build the index. However, it may take SerIF a few hours to build an index for a really large documents.

SerIF does not have a mechanism to deal with documents that have been moved to new URL's. The SerIF indexing system does not try to identify the new location of a moved document. If SerIF cannot find a document as referred to by the result of Infoseek, SerIF will ignore that document and marked it as totally non relevant. This caused Infoseek and SerIF to have a different index for the same document. SerIF also limits the length of the query to 80 characters long. This is used to conform to the same limitation of Infoseek.

6.3 Suggestions For Future Work

Following are some suggestions for improving SerIF:
1. Incorporating SeRIF into an existing WWW search system. This will take away the current speed bottleneck, since the search system will have the existing index of documents available to it.

2. Providing a mechanism for backtracking. Currently, SeRIF does not allow the user to go back to the previous refinement iterations. So, when a step in the refinement process goes wrong, there is no way for the user to go back to the previous iteration and makes changes to the documents relevancy values.

3. Providing a mechanism for more user intervention in the query reformulation process. This improvement may be very useful in improving the performance of SeRIF when it is used find some specific documents based on some general documents. The mechanism for user intervention can be done in two ways:
   (a) Allowing a user to specifically eliminate words that SeRIF adds to the query.
   (b) Giving a user a list of distinguishing words to add to the query and letting the user choose words to add to the existing query.

4. Providing a mechanism to filter out “anchor pages.” An anchor page is WWW document that act as a set of links to other more specific documents. An anchor page is usually very short (doesn’t contain too much textual information) and very general in content (acts only as a link to more specific documents). Therefore, this kind of WWW document is not a good document for a search system that uses a text indexing system. By filtering out anchor pages, it may be possible for SeRIF to find more specific documents.

5. Providing a mechanism for tracking moving documents. The location of documents in the WWW are dynamic. Often documents are moved from one server
to another server. Sometimes, the old server will give the information about the new location of the document. It would be nice if an improvement could be made to SeRIF to allow it to find the document on the new server site.

6. More thought is needed when forming test cases. Since the document collection of the WWW is very dynamic, it is hard to find a set of test cases that can be used to comprehensively test search and retrieval over that document collection. As far as I know, there are currently no published works on test cases that can be used to comprehensively test search and retrieval over the WWW documents collection.
Chapter 7
Conclusions

7.1 Conclusion

The following are my conclusions as to how the modified relevancy feedback mechanism, as implemented in SeRIF, worked in solving the problems identified with a typical WWW search system:

1. Given a query that contains words with multiple semantic identities, along with an initial list of documents and the user's relevancy feedback on those documents, SeRIF is able to find additional words to be added to the initial query. These additional words "explain" the correct semantic identity of the words found in the initial query. When these additional words are added to the initial query, this newly created query results in a higher precision retrieval. SeRIF, however, does not work well when the number of documents on the WWW that carry the correct semantic identity are very limited. SeRIF will still return the documents with the relevant semantic identity, but it will also return other documents that carry the wrong semantic identities. SeRIF also has a limited success in distinguishing two semantic identities of the same word if those
two semantic identities are closely related to one another (i.e. distinguishing the “Jordan” in the country of Jordan and the “Jordan” in the river of Jordan).

2. Given a query that contains words with general information about some specific topics, SeRIF is unable to add words into the initial query that will narrow down the meaning. The reason for the failure is because the initial documents are very general and contain information about many different subjects. These subjects may be loosely connected to each other. Unless there is a document(s) that primarily contain the specific information that the user is looking for, SeRIF cannot find that specific words that can be used to narrow down the subject area.

3. We can usually see how well SeRIF performs against a specific query after the first or second iterations. What this means is that after the first or second iterations, if SeRIF does not increase the precision, it will probably never increase the precision. The reason is because the words added during those first couple of iterations are not the right words. These words caused the retrieval of less relevant documents. SeRIF does not have a mechanism for backtracking that will enable the user to manually throw away some words in the query.

4. The addition of three words at a time to the query will speed up the relevancy feedback process. This implies two things. For the cases where SeRIF is able to find more relevant documents, the addition of three words at a time will find these relevant documents faster (in less iterations through the relevancy feedback process). However, for the cases where SeRIF fails to find relevant documents, the addition of three words at a time will quickly cause the Infoseek to return an empty list. Adding three words at a time also causes the length of
the query to reach 80 characters faster. Once the length of the query reaches 80 characters, Infoseek will not work and returns an empty list.

5. At this time, SeRIF is not very practical for real time searching. This is due mainly because of the time needed to build the index of each documents in the document list. One way to make SeRIF practical for the real time searching is to incorporate SeRIF into the actual WWW search system. This way, the document's indices are already available.

6. Most documents found in the WWW are very different in their content and arrangement compared to the traditional documents. The WWW documents are usually short documents with links to other WWW documents. There are also WWW documents that serve only as anchor documents. These anchor documents usually contains broad and general information with links to some more specific documents, which may serve to lead search systems like SeRIF astray.

7. The testing and characterization of the behaviour of search systems over a document collection as broad, heterogenous, and dynamic as the WWW is an open question. More research needs to be done on providing benchmarks for WWW search systems, and categorizing the behaviour of new and existing search systems for the WWW.

### 7.2 Summary

Currently available WWW search systems return low precision results. The low precision is due to the search systems failure to identify the correct semantic identity of the words in the query and the failure of the user to supply a specific and well
defined query. These two reasons for search system's failure may not be very clear
to the user initially. However, once the user receives the documents retrieved by the
search system, the user can identify additional information that can help refine the
initial query to get higher precision retrieval. However, there is no mechanism in
the WWW search systems that will allow for user feedback. A possible solution for
the problem is to build a system that will combine the current WWW search system
technology with the relevancy feedback method, developed in the field of information
retrieval, to automatically form new queries based on the user feedback on some
previously retrieved documents.

The first part of this thesis was devoted to describing the WWW search system
and the relevancy feedback method. A new system called SeRIF (Search Refinement
Incorporating relevancy Feedback System) was built based on the technology found
in the search system and the relevancy feedback method. Following the description
of SeRIF, we presented the results of running queries on SeRIF. The result showed
that SeRIF was able to help in identifying the correct semantic identity of the words
used in the initial query. However, SeRIF was not as successful in finding documents
on some specific topics based on documents on more general topic.
Bibliography


