Keyword-based Approaches to Improve Internet Search

Manolo Dulva Hina

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

January 2003

0-612-77713-8

Abstract

Keyword-based Approaches to Improve Internet Search

Manolo Dulva Hina

Technology keeps on evolving and so must the science of information retrieval. This thesis presents keyword-based approaches to improve information retrieval from the Internet. Focused and unfocused queries to search engines are considered, and means of obtaining relevant documents are presented. For focused queries, techniques are provided to obtain a high precision score from the hit documents; these documents do contain the exact answers to the focused query, which is usually a question. User queries are subjected to ambiguity test to determine if it is ambiguous, and if it is so, provide direction so as the user's intended meaning is the one that is actually searched. The queries are modified to form a new clear and unambiguous. Query is sent to several search engines at the same time, and hit documents from each of these search engines are collated and merged. Hit documents to an ambiguous query are analyzed and ranked based on their actual relevance to the query. Term frequency is used, along with popularity score, to determine the total score of a relevant document. Every relevant hit document is classified based on its academic relevance. A few academic categories are considered – (1) Course Notes, (2) Frequently Asked Questions, (3) Research Paper, (4) Technical Report, (5) Thesis, (6) Tutorial, (7) Review, and (8) Research Paper/Technical Report. Once a search is done, a set of relevant documents is presented, along with each document's academic relevance category (if any).

Acknowledgements

I would like to express my sincerest gratitude to Dr. R. Jayakumar, my thesis supervisor, for all the help, suggestion, understanding and guidance, and moral support. His constructive criticisms and keen ability to pinpoint important issues and weaknesses have helped tremendously in shaping this thesis. And so with Dr. Leila Kosseim, an examiner, who helped a lot in shaping up this thesis through her comments during the defence.

My gratitude goes as well to members of my family for their moral support and to my good friends, Mario Lemire, Gilles Gauthier and Allan Porlares, who were there to help me – in one way or another – in developing this thesis up to its completion. My gratitude as well to some BCEE students – Mr. Ji Li, and Mr. Nikhil – who were there and rendered some support during the defence up to the final modifications of the thesis.

Lastly, thanks to God Almighty for all the tangible and intangible helps that I have been receiving -- the completion of this thesis included -- and will still continue to receive.

This work is dedicated to my mother.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web is growing and as the Internet gets bigger each day, more and more people get information from the Internet as well. More and more people search information or transact business online. As far as searching information is concerned, more and more people, like students and researchers – from all over the world – are using search engines to help themselves find the information they need.

The design of the present search engines has to improve somehow in order to cope with the present and future demands of the Internet users – more results to the query, higher return of relevant documents to the query, being capable of handling both focused and unfocused queries, and guiding the user where to go in case of a confusing query.

The intention of this thesis is to try to move one step higher to the present crop of search engines. There are proposals of new concepts in improving the design of search engines, and facts and figures are presented to show that these concepts are feasible.

1.1 Internet Search Engines

An Internet *search engine* is a glorified index, a database that contains index terms. Search engines, in general, rely heavily on software robots to update their indices. A robot visits an HTML document, finds important keywords that could be considered as representative of the content of the document, and stores these keywords along with the Internet address of the document in the database.

The search engine providers keep a large database, such that whenever a user uses such search engine and inputs a query, the search engine itself finds relevant document within its database. It displays a list of titles of relevant documents, along with each document's Internet address, relevance score, and a short summary, usually the first few words that appear in the document.

An Internet search engine is an information retrieval tool, very similar in concept to information retrieval in library science. The only difference is that the scope of the search for the search engine is global. To keep the search engine database up-to-date, the robot or the crawler part of the search engine must continue to visit more documents, usually once or twice a month to keep its database fresh and up-to-date.

One slight variation of a search engine is a *directory*. In concept, it is very similar to a search engine except that the entries in the directory are completely entered by human, usually the one maintaining the directory. In contrast, the index database of the search engine is machine generated, as it is in general case. When the robot or crawler visits and analyzes a particular Internet document, the result of this process becomes an input to the index database, usually without direct intervention from the search engine provider.

There are pros and cons of using a search engine over a directory and vice-versa. A search engine does not need, to a certain extent, human intervention to update its database, but its scoring and ranking of document are occasionally misleading. This is because usually people find a completely irrelevant document getting a high score and a very relevant document with a quite low relevance score. A directory, on the other hand, being subject to human intervention can be biased on one or two documents in which the directory provider has a vested interest. In effect, it is not completely remote that a document with lesser relevance could be ranked higher than one with higher content relevance in a directory.

An index database that comes in between a search engine and a directory is called a *hybrid*. That is to say that its index database is partly based on the search engine's way of ranking document by a ranking algorithm without human intervention, and also partly similar to a directory's way of ranking document with human intervention.

As of now, there are many search engines available in the Internet, but only about 5 to 7 are popular because they are automatically invoked as default search engines by the Internet browsers (i.e. Netscape, Internet Explorer) whenever the search buttons in these browsers are clicked. The popular ones are *Google, Netscape, Excite, Hotbot,* *Looksmart, Lycos*, and *AltaVista*. On the other hand, the most dominant directory is *Yahoo!*

While the present crop of search engines is helpful in identifying documents relevant to a user's query, they have their own pitfalls and limitations to wit:

(i) Some of the documents presented in the hits list of user's query are "*irrelevant*", the way an ordinary user applies the concept of "*relevance*".

(ii) Some of the documents presented in the hits list of user's query do not deserve their relevance scores.

(iii) Some of the documents presented in the hits list of user's query are dead or non-existing documents. This could only be found out when a dead Internet document is being surfed or downloaded – the user waits for a few minutes, then he gets a message saying that the document is not responding or non-existing.

(iv) The majority – if not all – of the search engines do not address the problem associated when the user fields a semantically ambiguous query – a query keyword that has more than one definition. The results of this is that a greater majority of the documents in the hits list are irrelevant as far as the user's desired intention of the query is concerned.

(v) There is still no way to determine in advance the category of a document -- whether academic, commercial or any other category -- when a document is presented in the hits list of a user's query. A user with an academic interest (e.g. professor, student, researcher, etc.) would have to open and download every document, read every piece of document in order to determine each one's academic relevance.

(vi) Search engines apply the concept of information retrieval on text documents alone. There is no search engine at present that indexes documents in other media format such as audio and video.

1.2 Objective of Thesis Work

The original aim of this thesis is to come up with a multimedia search engine. As time passed by, we realized that the work is too huge for a single researcher. The scope of the thesis itself is too huge and the needed tools and facilities are not available. We first had some restrictions in the computing facilities of Computing Science Department – limited amount of memory space quota for students, no access to the server, absence of speech recognition tools available. We then decided to limit the scope to one media – the HTML text document.

Our frustration on many irrelevant results in our search engine queries inspired us to conceptualize ways in improving search engines, particularly in order for them to produce relevant hit documents. Our first problem is that we cannot go to commercial search engines and ask them to modify the ways they search relevant documents nor impose upon them whatever we think is the good way of searching relevant documents. Next, we cannot make our own search engine – we have limited time for studies, and limited memory space quota to work on. Indeed, we are to find techniques to improve relevance of search engine hit results, but within the given constraints.

We eventually come up with our own solution – a front-end, a link between the user himself on one hand, and the search engines on the other. The objectives of this thesis work are twofold:

1. To have a means that ensures that the user query is clear and unambiguous before the user's query is passed to the search engines.
2. To collect, collate, and find only relevant documents from the set of hit documents returned by search engines, and return these relevant documents to the user.

We believe that search engines return irrelevant hit documents due to two reasons:

1. That the quality of user's query is bad – either because the user's query is poorly constructed, or it is ambiguous. We all do not expect to get good output when the input is also not good.
2. That the search engines (or directories or hybrids) themselves have inferior methods of finding relevant documents – poor ranking algorithm, or bias on a particular document(s), etc.

We believe that by blocking the user's query first and analyzing it for some ambiguity before submitting it to the search engine will ensure that the quality of user's query is indeed good. As such, there will be a good possibility that relevant hit documents will be returned. On the other side of the front-end, we believe that by collating search engines hit results, then finding only relevant documents among them, and finally returning these relevant documents to the user will ensure that the user's time in surfing the Web is a time spent wisely and productively.

1.3 Thesis Contribution

The aim of this thesis is to present some new concepts that will address the majority of the search engine limitations presented above.

One aim of this thesis is to present and strictly implement the relevance scoring policy to Web documents. In doing so, the study intends to give a justifiable evaluation of document relevance, giving the user more results that could otherwise be obtained by multiple searches.

The above-mentioned objective can be realized by submitting the user query to multiple search engines, and collating the results and ranking them accordingly. This tool is generally called a "*metacrawler*". This concept was original until *Copernic* came into the scene in year 2000. The researcher of this thesis had this concept since 1997 but failed to submit a complete thesis due to many reasons.

Among the present crop of search engines, it is always possible that one document could appear in the database of more than one search engine, let us say two. It is likewise possible that this same document would have different relevance score to each of these search engines even for the same query. This is due to the fact that every search engine has its own ranking algorithm that could be different from the ranking algorithms of other search engines.

There are a few ranking algorithms available in information retrieval. This thesis intends to present a combination of TF (Term Frequency) algorithm along with Most-Cited ranking algorithm. To be very specific, a default value of 80% of the relevance will be based on TF policy, and 20% is based upon Most-Cited policy. These are the relevance scoring distribution by default. The proposed system does provide, however, an option by which the user could modify this score distribution.

When merging and re-ranking combined documents that are presented as hits by two or more search engines, it pays to ignore the relevance scores assigned to the documents by the individual search engine. This is because it is not wise to merge documents and retain their scores when these scores are calculated using different formulas. The search engines themselves are not even informing the users of what exactly their ranking algorithm is based upon. The best way to deal with this inconsistent and different ranking algorithms used by different search engines is to use our own relevance scoring formula and apply it to all hit documents.

To ensure a high probability that only the most relevant hit documents for the query are obtained, only 10 documents (the default value) per search engine are obtained among the many documents presented by each search engine in the hits list. That is to say that if a user submits a query to 3 search engines, at most 30 distinct documents are going to be generated to be scored and to be ranked for the user. The rationale behind this is that the 10 highest-ranked documents are the most relevant to the user's query as far as the search engine is concerned. And there is a high probability that it is rightly so. Our reasons for selecting only the first 10 documents are twofold: (1) the first page of the hit results of the majority of search engines contain the top 10 documents, and (2) we limit the number of documents to 10 because we simply intend to test if our concept is feasible.

The cons of taking only 10 documents from search engine hits result is that these 10 are not necessarily the highest ranked documents if they are ranked independently. But no one knows the relevance score of the other documents ranked 11th and downward. There is a slight probability that a few of these documents presented as top 10 hits by a search engine are irrelevant. However, once a ranking algorithm is imposed upon each of these documents, an irrelevant document will obviously find itself at the bottom of the list or, in the worst case, be discarded altogether.

Many ranking algorithms are used in information retrieval. It is wise to use an algorithm based on the condition that best suits that algorithm. We used term frequency because it measures the relevance of a document based on the presence and the frequency of keywords in the document. These keywords are the keywords in the user's query. Usually, term frequency is associated with Inverse Document Frequency

(IDF) which increases relevance score if the term appears frequently in the document, but decreases when this term appears in too many documents that the term becomes so common and therefore insignificant. The IDF policy is ignored because the situation here deals with a very small number of documents, about 30 documents if 3 search engines are chosen to about 50 documents if 5 search engines are chosen. It is very unlikely that IDF will play a vital role since not a large database is involved to compare presence of keywords in other documents, as the search engine providers do.

The ranking algorithm used also involves most-cited ranking scheme (a.k.a. popularity measure). Theoretically, it assigns documents larger scores to a relevant document that is referenced by another relevant documents. In concept, one relevant document will not put in its body a link to another document of same subject if the referenced document is not worthy to be cited. This is viewed as a bonus (20% maximum) to a document that is well done that other documents in the Web discussing the same subject do provide link to it in their pages.

In theory, one person practicing a specific field will not be popular among his colleagues if he is not interesting in their own domain. He becomes popular because he has done meritorious things. The same principle is used in our most-cited ranking scheme.

Potentially, the small number of test documents will most likely produce a scenario that none of these documents is popular among documents listed in the top hits.

Hence, the popularity of a document will most likely be zero. In such a case, a document relevance score will be that of its TF score alone, which will give it a likely score of 80% as the maximum.

The relative percentages assigned to the TF scheme (80%) and most-cited scheme (20%) are default values. The user has a way of changing these scores to desired value in the proposed implementation of this scheme. If the user so desires, popularity can be omitted altogether (by giving it a 0% weight) and simply the term frequency may be used to rank documents.

Another aim of this thesis is to categorize documents in the Internet. Only that this is implemented in a limited scope – academic application. With this in mind, this thesis would like to contribute to the speedy access of relevant academic documents available in the Internet. It is hoped that after doing this, some other document categorization will follow – such as identifying the commercial or economic relevance of a particular document, just to cite an example.

One benefit of informing a user that a document is a course note or a FAQ document, or any other category even before such document is downloaded saved time in identifying document of academic relevance. Having seen the academic category as list of documents are presented as hits to a user's query, the user then has the option of downloading only category of his choice.

This contribution is important in the sense that there is too much information on the Net nowadays that if there is no tool like this, the user might end up spending too much time just to find too little of subjects of his interest. Some information in the Net are simply not interesting, some are relevant. This study intends to give users relevant documents.

One more contribution that this thesis would intend to render is to provide directions to user's query whenever such query is ambiguous in nature. This work intends to identify if the query is ambiguous, and if it is so, then informs the user of available options (actually, additional keywords) that would potentially lead to a more meaningful query. That is the query and additional keyword combination will lead to giving results whose contents are likely to be the definition the user wishes or means in such an ambiguous query.

The thesis also makes contribution by giving distinction between a focused and unfocused queries based on query keywords entered by the user. It does provide ways to come up with meaningful and relevant hits to the queries whether the query itself is focused or unfocused. By finding a way to find hits to unfocused queries, the search engines will be more relevant to the needs of people, notably students, who try to find quick answers based on easy, non-complicated queries.

In summary, this thesis intends to contribute in:

(i) Providing a scheme to detect an ambiguous query, and provide user choice of keywords that when added to the original query will lead to a more meaningful, non-ambiguous query. It helps in obtaining documents relevant to the definition the user actually mean.

(ii) Providing relevant hits to both focused and unfocused queries.

(iii) Providing more hit documents to a user's Internet query by submitting the same query to multiple search engines and collating their results.

(iv) Providing a ranking algorithm that will calculate the relevance scores of the documents based on term frequency and popularity.

(v) Providing a categorization of documents in relation to its academic relevance. Categories include course notes, FAQ, thesis, technical report, etc.

1.4 Organization of Thesis

The thesis is basically divided into five major parts:

(i) Identifying and providing directions to ambiguous queries,

(ii) Finding relevant hits to focused and unfocused queries,

(iii) Classifying Internet documents as per academic relevance,

(iv) Ranking and merging documents from several search engines, and

(v) Appendix.

Identifying and providing directions to ambiguous queries discusses the theoretical concepts and implementation details. It discusses how user's query is identified as ambiguous using a dictionary. Parsing of query and keyword definition is discussed, together with the tools used in implementing it – lists of stop words and stop symbols. The user is eventually provided with a list of keywords, and a choice of one keyword is appended to the original query to form a newer (and more meaningful) query. If results to this query is significantly low, the user is provided with some choice of keywords that could possibly mean the same as his original query, but may possibly produce more hits had it been used instead of the original query word. This is realized through the use of a thesaurus.

The second part of the thesis is a discussion of how to find relevant documents for focused and unfocused queries. It discusses an unfocused query as one that generally ask questions, usually beginning with keywords *what, who, when, where, why*, and *how*, and as such the relevant document should be one that answers the question. A focused query is one in which user enters some keywords, so the relevant documents generated by the search engines are those that are related to the keywords that was entered by the user. For unfocused queries, the study discusses ways and means to trim the question into tokens, select meaningful keywords from these tokens, and use these keywords to search documents from the Internet. The selection of document is then decided based on the contents of these documents – specifically, if its contents answers the question posed by the user. The focused query part is basically the same as that of the unfocused query, except that there is no question to process as tokens of keywords; hence, all the user's keywords are accepted as parameters to find relevant documents.

The third part – *Classifying Internet documents as per academic relevance* – begins with discussion of what qualifies a document category – course note, for instance. Eight categories are discussed, namely:

(1) *Course Notes,*
(2) *FAQ (Frequently Asked Questions),*
(3) *Research Papers,*
(4) *Technical Report,*
(5) *Theses,*
(6) *Tutorials,*
(7) *Reviews, and*
(8) *Research Papers/Technical Reports.*

It discusses the methods used in finding relevant keywords – (1) in the title, (2) in the heading, (3) in the body, and (4) a test of scoring a document probabilistically with reference to keywords associated with a document category in the database.

Along with the methods of searching relevant keywords is an explanation of tools used in its implementation in Java language – stop words, stop symbols, stop tags, essential HTML tags, and the term frequency scoring.

The fourth part – *Ranking and merging documents from several search engines* – begins with analyzing keywords that appear in user's query, and the possible Boolean operation involved. A list of probable Boolean operations are presented along with their definitions, and how it can be accomplished in the program.

Also presented within this part are some ranking algorithms that are existing in the field of information retrieval. Each of these techniques is presented in general concepts. Eventually a choice is made that TF and Most-cited techniques are combined together and the combination is used as the algorithm to rank a document.

The concept of multithreading in Java programming language is presented as a tool to realize the task of downloading and analyzing Internet documents at the same time. This is also to say that the usual technique of sequentially processing a task of retrieving and analyzing multiple documents is not feasible in the Internet.

Finally, the scoring of each document is done, and the hits are presented back to the user. The proposed final result of this process is a list of relevant documents out of a non-ambiguous query, taken from multiple search engines, analyzed and ranked accordingly, with its academic category, if any, attached to it.

The appendix of this thesis contains miscellaneous topics. The appendix on stop list enumerates stop words and stop symbols that are used in discriminating what constitutes a keyword that represents a document, and what does not.

The ranking algorithms and their meanings and mathematical calculations are presented as well. Also, a method and some lists of keywords that identify a course note, or the other seven more document categories are listed down.

A list of algorithms (instead of actual Java program codes) is presented in Appendix D. A few of these algorithms are about the academic categorization of document based on its title, headings, and contents. An algorithm on how to determine an ambiguous query is also presented. The parsing of a HTML document is also presented to show how to manipulate the HTML tags to obtain some useful information about the document – title, headings, and anchors or referenced documents.

Chapter 2

Identifying and Providing Directions to Ambiguous Queries

An *ambiguous query* is a user query to the Internet search engines that could potentially lead to producing irrelevant document hits because the query itself has more than one meaning. In this thesis, we would like to identify if a query is ambiguous, and, if it is so, provide the proper direction to come up with a more meaningful, non-ambiguous query that will potentially lead to produce good and meaningful hit results.

2.1 Rationale

It is important that whenever a user inputs an ambiguous query, a search engine should detect that the query is indeed ambiguous, and provide direction to clearly indicate the user's intention. The rationality of such a thing is that Internet is open to everyone – including primary school kids who may not be well versed in English, or even more matured people whose mother tongue is not English – who could query for subjects that may have more than one meaning. We intend that our work should accommodate the greater majority, whether or not the user could enter a clear query or an ambiguous query. We would like to suggest that an Internet search engine should include a feature that would identify when a query is ambiguous. If so, it should provide the most, if not all, of the possible meanings of such a query. Some choices should be presented to the user to inform him of the possible meanings or definitions of his original query, then allows him to select one meaning, and the search engine then returns, subject to the limitation of its

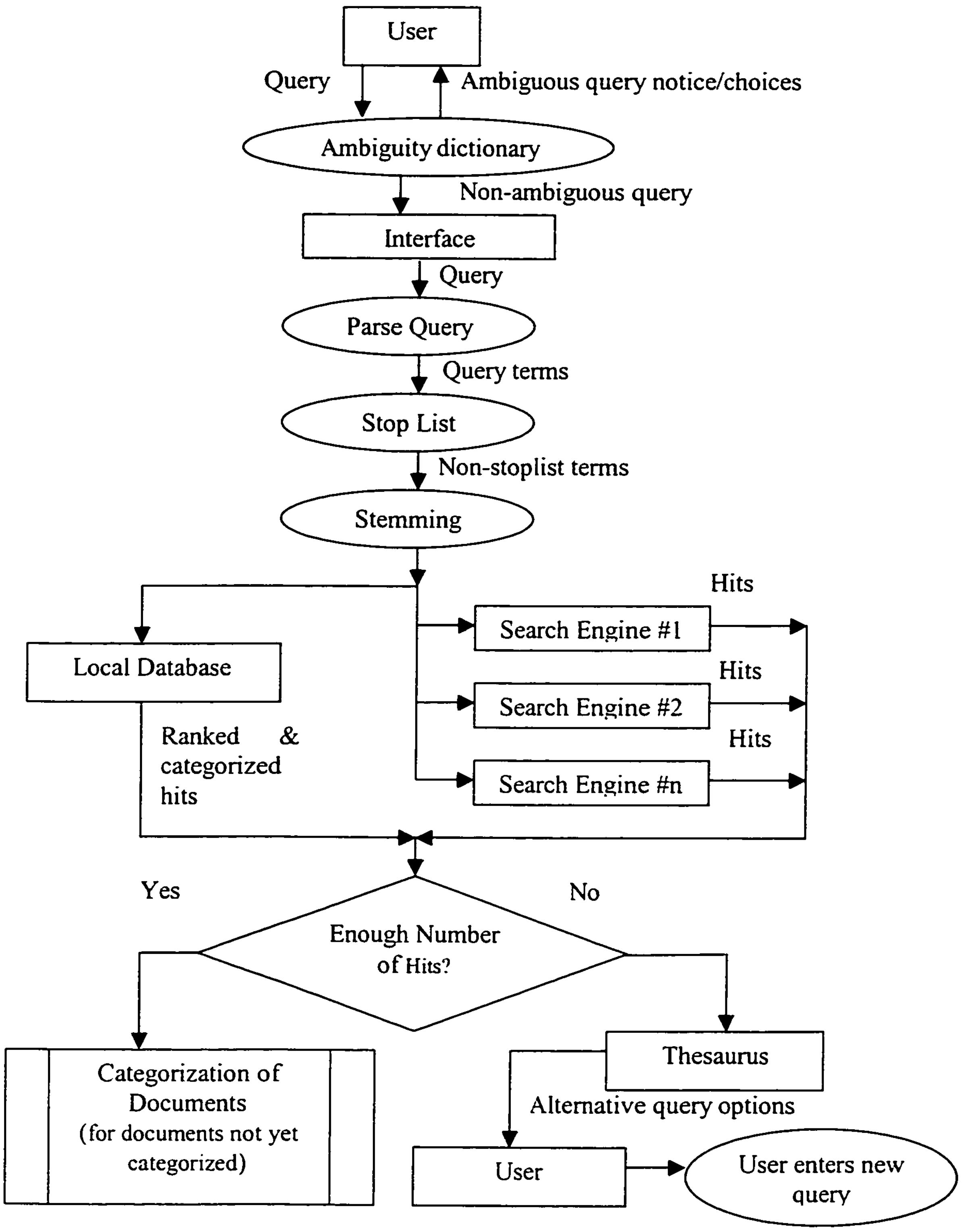


Figure 2.1 Flowchart showing detection and directions for ambiguous queries

resources, some hit documents whose contents are semantically relevant to the definition the user has earlier opted.

The flowchart in Fig. 2.1 shows the intended front-end that will address user's ambiguous query. The user inputs a query. A one-word query is subjected to ambiguity test. The ambiguity dictionary is used to determine the query's ambiguity. If the query is found in the dictionary, then the query is ambiguous, otherwise the query is considered clear and unambiguous.

For the moment, an ambiguity dictionary does not exist, and so the online WordNet system is temporary used to detect the query's ambiguity. In the event that the query is ambiguous, the user is given the choices of the query's senses or definitions, and the user is expected to choose one definition or another keyword that is not in the list of definitions that the user wishes to add. The user's intended definition is added to the original query forming a new modified, non-ambiguous query.

A query can either be focused or unfocused. Chapter 3 of this thesis discusses focused queries. In the event that the query is unfocused, the query is fed to the local database to obtain hits (this is assuming that the query results already exist because the user made such query in the past). If the query is something new, the query is sent to the search engines to obtain hit results.

The number of hit results to the query is important. If the number of hit documents are enough the documents are categorized, and eventually sent to the user. If the query hit results are insufficient, a suggestion is made by the system asking the user to re-enter query using any of the keywords similar to the original query, or enter a completely new query. The first option in which the system asks the user to re-enter the query by using another keyword is made possible using the help of the thesaurus.

2.2 Related Work and Literature Survey

Query ambiguity is a generally recognized problem, particularly in the Web environments where queries are commonly short, and are only one or two words in length [1, 2]. Some work were done in determining the user's intention when user enters a query. The work in [1] investigated a technique for resolving ambiguity by using part-of-speech pattern. It assumes all one-word query is ambiguous, and uses part-of-speech pattern to deduce all possible semantic variations of the query, and asks the user to supply some information. The work in [2] is even more complex because it tried to determine the user's activity, like reading some of the user's Microsoft Word document(s) to deduce the user's profession or interests, and then use this information to determine the user's intention in case of an ambiguous query. In most of the cases presented in earlier related work, one or two-word query are often considered ambiguous because there are less information to the query. As number of words in the query increases, the ambiguity of the query reduces because the additional words add further clarity to the query.

In general, a one-word query is likely to be ambiguous. For example, take "*bird*" as an example. It is ambiguous because there are simply too many topics on bird, and the bird family itself is huge – is it about a dove, a parrot, a pigeon, or an eagle? We do not know. However, whether the hit documents deal with migratory bird, or bird of prey, or feeding bird, or a singing parrot, the fact remains that the document is discussing something related to an animal with feathers, that have wings and usually could fly. When we take this thing into account, we could then say that a one-word query like "*bird*" is not at all that ambiguous.

However, consider a case when a one-word query does have multiple meanings or definitions, say "*java*". The fact that "*java*" could mean a certain place in Indonesia, or an object-oriented programming language, or a beverage could make this query word capable of producing conflicting and confusing hit results – in all possible definitions of java. Hence, we do consider a one-word query as ambiguous only when it has multiple definitions, each definition having no relation to other definitions.

Our concept of query ambiguity is based on keyword, and not on the context. Basically a keyword is searched in a dictionary to determine if it has one or more different meanings. Two or more different definitions for the keyword make the keyword ambiguous.

Obviously there are many other ways to check if the query is ambiguous or not. And it is not limited to one-word query only. Other approaches could be implemented to determine the context of a two- or three- or more words in the query. For example, a two-word query like "*interest rate*" is ambiguous because there are many kinds of "*interest rate*". But the approach here is contextual, so a tool like a dictionary is not going to detect the query's ambiguity. Language expertise is essential to deal with ambiguity on query with two or more keywords. We believe that this part of our thesis can be extended further to accommodate two or more-word query, however, we leave it as future work.

2.3 Theories and Concepts Applied

In this thesis, we adopt the dictionary supplied by the WordNet system reference to decide if a term is ambiguous or not. WordNet is an online lexical reference system developed by the Cognitive Science Laboratory at Princeton University under the direction of Professor Georges A. Miller. The single word query is submitted to the WordNet system for a definition (or "*senses*" as WordNet calls it). If the WordNet system gives back more than one definition (or senses), the query is considered ambiguous.

The WordNet system is a preferred choice as an online dictionary because this system is a product of work and research of some of the best linguists in the United States. That being said, our work is flexible enough that an online dictionary other than WordNet system could be used and still our front-end system is still going to work.

We propose that in the future, the researcher should probably construct his own *ambiguity dictionary*. This ambiguity dictionary is a database of words that have multiple meanings in the English vocabulary. To use the ambiguity dictionary, the user should supply a keyword. The keyword is searched within the dictionary, and the ambiguity dictionary will return all possible meanings of such keyword. For example, supplying an ambiguous word "*bat*" would return some keywords like *"animal", "bird", "stick", "sports"*, and "*game*". One of these keywords would be necessary to identify which "*bat*" the user really mean – the animal bat or the bat used in sports such as baseball.

The use of ambiguity dictionary is *more efficient* than using the online WordNet system. For one, the list of words in this dictionary is limited to ambiguous words only, hence, a shorter search in a smaller database. Next, the definition words are available right away; there is no need to take a whole bunch of words and phrase, trim it to find the right definition keywords, as is presently being done using the WordNet system.

2.3.1 Identifying if a query is ambiguous or not

Terms such as *"orange"*, *"bat"*, and "*plant*" are semantically ambiguous words because they have more than one definition, more often these definitions are miles apart from one another. Consider for example the word "*orange*". It could mean a *fruit*, or a *colour*, or a *tree*, or a *place* such as a specific county or town.

2.3.1.1 What constitutes a non-ambiguous query?

Whenever a user enters a *single-word query*, such a query will be subjected to a test of whether it has one or multiple meanings. No such test will be conducted if a query is a two- or three-word query, or if the query is a phrase. If a user inputs a query that has more than one keyword, its meaning is determined by the Boolean operation between the words. For example, a query like "*air pollution*" (or "*air + pollution*") will mean find documents that contains topic on "*air*" or "*pollution*" or "*air pollution*". A query like "*air & pollution*" (or "*air and pollution*") will mean find documents that contain both the keywords "*air*" and "*pollution*" within the same document. When a user enters a query in quotes, like "*air pollution*", this will mean find documents that contain the phrase "*air pollution*" in the document. A two-word (or more) query will not be subjected to ambiguity check.

Logical operation	**Example(s)**	**Meaning**
OR	"air pollution" "air + pollution" "air or pollution"	Find documents that contain topic on "air", or "pollution", or "air pollution".
AND	"air & pollution" "air and pollution"	Find documents that contain topic both on "air" and "pollution".
Phrase	" "air pollution" "	Find documents that contain the exact phrase "air pollution".

Figure 2.2 List of logical operations and their meanings

2.3.1.2 What constitutes an ambiguous query?

A *single-word user query* is automatically subjected to an ambiguity test. To determine if a particular keyword has multiple meanings, we create a Java process that will find online the definitions (or senses) of the query. Assuming that the user single-word query is "***query***", then the online definition (or sense) of query in WordNet system is given by the URL:

http://www.cogsci.princeton.edu/cgi-
bin/webwn1.7.1?stage=2&word=**query**&posnumber=1&searcht
ypenumber=2&senses=

*Figure 2.3 The WordNet system online URL for the definition of a term "**query**"*

where ***query*** should be replaced by any one-word query, such as orange, bat, etc. The content of the URL is a HTML document that contains the definitions (or senses) of the word in question. We created a process that opens up a URL, save the content as a local text file, and is then parsed to obtain the meanings (or senses) of the word in question. In WordNet system, every meaning of the word in question is always indicated as "***Sense n***" where ***n*** is a numeric figure. Hence, if there is only Sense1 in the text file, it means that the word has only one definition, hence query is assumed to be clear and unambiguous. However, if there are many definitions or senses (at least there is Sense 2 in the text file), the query word is considered as ambiguous, and then meanings or senses are collected. These senses are collated and are presented as keywords to be added to the query.

If the query is not ambiguous, the search for relevant documents would proceed in the normal way. If the query is ambiguous, the user will be asked for a specific definition or sense that he would like to attach to the ambiguous word. With the user's response, the extra keyword is added to the original query to form a new query. The new query words that will be formed will be the original query word '***and***' the new keyword the user has just entered. The new query formed will then be resubmitted.

When the part-of-speech category of a one-word query could be either that of a noun or a verb (e.g. dance), we are likely to end up with many senses of the word – one sense or many senses of the word as a noun, and another sense(s) as a verb. In such a case, we will take the sense of the query word as a noun. The reason for this is our perception or intuition that the user is likely to query on a subject – which is generally a noun – rather than an action.

To cite an example, let us consider the ambiguous query "*orange*". The following steps will be undertaken:

1. The user enters the query word "*orange*" (without the double quotes).
2. The query, which is single-worded, is subjected to ambiguity test. A process to obtain the meanings or senses of orange is enabled. Hence, the online URL to obtain the senses or definitions of "*orange*" will be http://www.cogsci.princeton.edu/cgi-bin/webwn1.7.1?stage=2&word=**orange**&posnumber=1&searchtypenumber=2&senses=
3. The output of the process in step 2 yields a text file showing the *definitions of "orange"*, which is given below:

Results for "Synonyms, ordered by estimated frequency" search of noun "orange"

5 senses of orange
Sense 1
orange
    => citrus, citrus fruit, citrous fruit
Sense 2
orange, orangeness
    => chromatic color, chromatic colour, spectral color, spectral colour
Sense 3
orange, orange tree
    => citrus, citrus tree
Sense 4
orange
    => pigment
Sense 5
Orange, Orange River
    => river

Figure 2.4 The WordNet system's definitions or senses of "orange"

4. The number of senses is equal or greater than 2 in the text file above denotes that the query keyword ("*orange*") has 2 or more meanings, and therefore ambiguous. In this case, there are 5 senses, therefore there are 5 meanings.

5. The definitions or senses are taken and presented to the user. For the above example, using "*orange*", the modified, unambiguous query would be any of the following:

1. *orange and citrus*
2. *orange and "citrus fruit"*
3. *orange and "citrous fruit"*
4. *orange and "chromatic color"*
5. *orange and "chromatic colour"*
6. *orange and "spectral color"*
7. *orange and "spectral colour"*
8. *orange and "citrus tree"*
9. *orange and pigment*
10. *orange and river*

Figure 2.5 List of possible clear and unambiguous queries about "orange"

We note that athough "*citrus fruit*" and "*citrous fruit*" are exactly the same, we do consider them as two separate possible queries, knowing fully well that the "*citrus fruit*" in the query will yield documents that uses American English, while "*citrous fruit*" will yield documents that uses British English.

6. The user selects one of those definitions. Assume that, for the sake of just giving example, the user enters or chooses "*pigment*". The new query formed is "*orange and pigment*". The search is on for documents that contain both keywords "*orange*" and "*pigment*" in the same document. The same logic will be used if for example the user chooses "*citrus tree*" instead of "*pigment*". The newly formed query then will be *orange and "citrus tree"*.

7. Hit documents for this query are then presented to the user.

2.3.2 Searching the Internet for hit results

After the new query is deemed complete, that is, having undergone the ambiguity check and the user having entered the added keyword if the query was ambiguous, all is set to submit the modified, more meaningful query to the search engines. The search engines in which this query will be submitted shall be the following:

1. *Excite*
2. *Hotbot*
3. *Google*

Why these three search engines? Basically, because these are some of the popular search engines online, and are usually automatically invoked when we click on the "*Search*" button of the Web browser Netscape Navigator. The number of search engines was limited to three (3) because we primarily wanted to test if it works or not on limited search engines. Of course, it is always a possibility that we can add some other search engines to the list later, or modify these default search engines into something else. And since the Internet keeps growing and changing every single day, it is wise that search engine tools should be updated often – and that includes changing, adding, or modifying the default search engines as in this case.

The default of this system, as implemented in the software that is used for the implementation of the thesis concepts, is that the query will be submitted to three search engines, namely: *Excite, Hotbot, and Google*. The user, however, is given a list of search engines, and hence could change this default and select the search engines of his choice from the given list.

How is the actual search itself is done? We first determined the necessary URL address corresponding to the query for a particular search engine. This is determined by doing some series of tests to find the URL base address of a particular search engine, to which the keywords in the query will be attached to form a complete URL that will yield some hit results to the query. For example, if we are to submit the query "*operating system*" using Hotbot search engine, we will get the hit results of such a query, and the Hotbot URL (Uniform Resource Locator, or the document address in the World Wide Web) for such a page. The URL itself contains the base address of the search engine that was used and the query words. If we reverse the process -- by taking out the keywords from the URL -- what will be left will be the base address of the URL for such search engine.

Hence, any query is made by attaching the query words to the base address of a search engine, and downloading and browsing this address will lead us to the hit results of a query. Of course, the address formulation of one search engine would always be different from other search engines. That is to say that the URL address that produces the hit results of a query like, say "*operating system*" in Hotbot would be different from the URL address for the same query in another search engine, like Alta Vista.

It should be noted that the URL of a particular search engine may change from time to time. That being said, the URL for some search engine will not stay eternal, and the software that accompanies this thesis will not work at some period of time in the future because it is basically dependent to the correctness of the URL of a search engine. To solve this probable problem, the software must be maintained to make sure that the URL of search engines used in the software is always correct, and, if necessary, add some more search engines and take out some search engines that are no longer in existence. This is basic software maintenance, a part of software life cycle.

2.3.2.1 Description of software that searches the Internet for query results

In our Java program that accompanies this thesis, a URL thread is created to submit the user query online, receive the response to the query online, and save the query response as a local text file. The online query is applied to all above-mentioned search engines. In order to query three (3) search engines, we realized that this process should be done in parallel – submitting the user query to three search engines in parallel -- making the complete process run faster in comparison to the one running as a sequential process.

Since there are three queries, there will be three responses to our query. The query response from individual search engine is saved as a local file. Each of the 3 local files, representing query results from the 3 search engines, is parsed and analyzed to get the URL list of query hits.

2.3.3 Insufficient hits

When the query production is successful, there is a strong possibility that some hit results will be returned by a search engine. However, some search engines are very good to use in areas concerning technology, while others may be very good for the areas concerning entertainment. Having said that, it is a possibility that a query will produce some hits from one search engine, while getting none from another search engine.

There will be rare cases where a query will produce zero result from all search engines combined. The reasons for the lack of resulting documents are twofold:

(1) The subject is a rare commodity that not even one or only very few documents on this subject exist in the Internet. It could also mean that some enterprise may be doing something related to the subject of the query, but the enterprise itself has no working website.

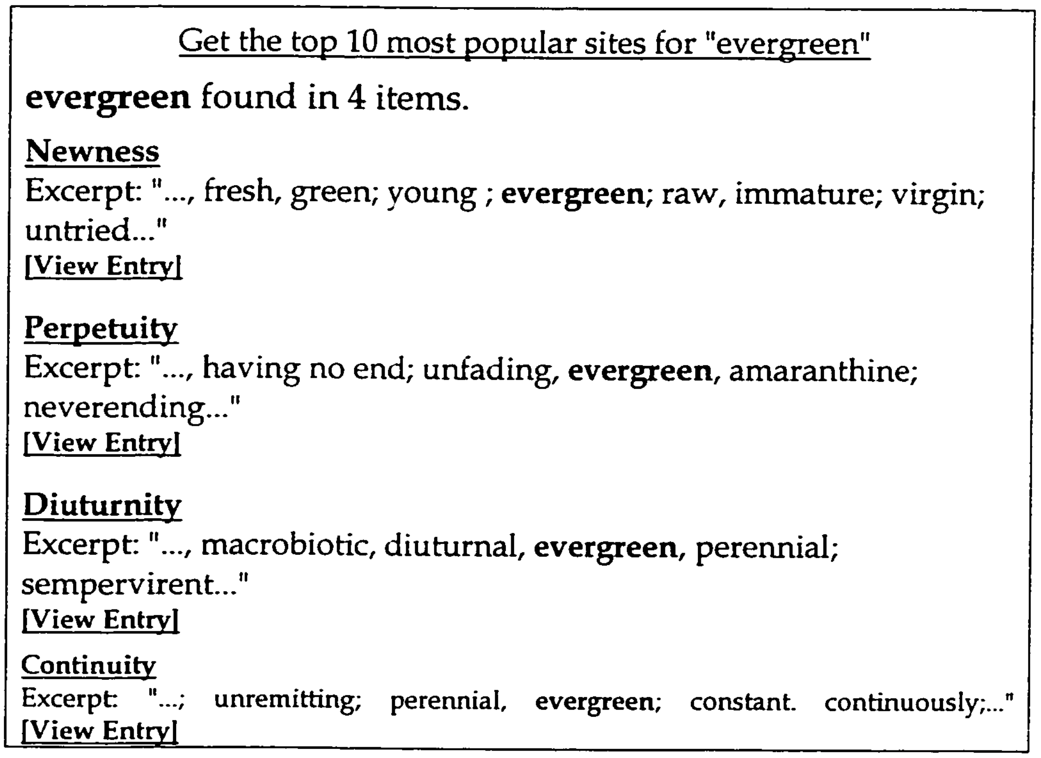
(2) The query is poorly constructed that search engines databases could not find relevant documents.

In any case, the output to the query is a limited document hits and probably a suggestion on the improvement of, or reconstruction of the query itself. When the hits are very few (less than 50% of the expected number of results), a suggestion is presented. The user is asked to resubmit the query with different keywords.

The original single-word query is resubmitted to the site "*www.thesuarus.com*" and the keyword is searched in the online thesaurus. The site www.thesaurus.com is just one of the many thesaurus available on the Internet. In general, our system can adopt any online thesaurus with very little modifications. For www.thesaurus.com, if keyword "*keyword*", for example, is to be submitted to the above-mentioned site, the site's output is given by the website "*http://www.thesaurus.com/cgi-bin/search?config=roget&words=**keyword***". The bold-faced keyword in the URL is simply replaced by whatever keyword the user is intending to search. Consider two cases of keywords that can and cannot be found in the thesaurus.

Case 1: Keywords that have some entries in the thesaurus.

Assume that the keyword "*evergreen*" is submitted to the online thesaurus. It can be deduced that the result of the query is given by the URL "*http://www.thesaurus.com/cgi-bin/search?config=roget&words=evergreen*". The site lists down 4 entries for "*evergreen*". Given below is the resulting document (minus the HTML tags).



Get the top 10 most popular sites for "evergreen"

evergreen found in 4 items.

Newness
Excerpt: "..., fresh, green; young ; **evergreen**; raw, immature; virgin; untried..."
[View Entry]

Perpetuity
Excerpt: "..., having no end; unfading, **evergreen**, amaranthine; neverending..."
[View Entry]

Diuturnity
Excerpt: "..., macrobiotic, diuturnal, **evergreen**, perennial; sempervirent..."
[View Entry]

Continuity
Excerpt: "...; unremitting; perennial, evergreen; constant. continuously;..."
[View Entry]

Figure 2.6 Thesaurus output for query "evergreen", a sample keyword that has entries in the thesaurus.

Saving this document on the disk and parsing it appropriately leads to producing four possible keywords as substitute for "*evergreen*":

1. *Newness*
2. *Perpetuity*
3. *Diuturnity*
4. *Continuity*

Figure 2.7 Suggested keywords to replace "evergreen"

Case 2: <u>Keywords that have no entries in the thesaurus.</u>

Assume that the keyword "*everglade*" is submitted to the online thesaurus. Again, it can be deduced that the result of the query is given by the URL "*http://www.thesaurus.com/cgi-bin/search?config=roget&words=**everglade***". The site lists down no entries for "*everglade*". Given below is the resulting document (minus the HTML tags).

everglade not found.

Try your search for "everglade" at:

- Amazon.com - Shop for books, music and more
- AskJeeves.com - Get the top 10 most popular sites
- Dictionary.com - Search for definitions
- Electric Library - Search thousands of newspapers and magazines
- Google - Search the Web for relevant results
- Google Groups - Search Usenet messages back to 1995

Overture - Search the Web

Figure 2.8 A sample keyword query that has no entry in the thesaurus

2.3.4 Modified query

Since the original query produces very little result, depending on the result of the search on the online thesaurus, the user is given the option of formulating a new query all by himself, or choose any of the words suggested by the system as a new query word.

The latter option is realized when the query on the online thesaurus produces some entries. These entries are synonymous to the original query word, and therefore could be used by the user as a new query word. The user, if he so desires, could choose one of the synonymous words, and submit it as a new query word.

When the user re-submits his new query, the same process as discussed above will be repeated – analysis of query, determining if query is ambiguous or not, and then finding relevant hits for such a query.

2.4 Experimental Results

We conducted ten (10) one-word query searches, and each query is ambiguous. We listed down the different definitions of each of the query, as given by the WordNet system. We sent the query to three search engines – Excite, Hotbot, and Google – and obtain the first top ten hits of the individual search engine. Each hit document is analyzed to determine which definition of the query the document is all about. At the end, a frequency tabulation is made to illustrate how many documents are dealing with what definition of the query.

The tabulation in the experimental results is meant to measure the rate of precision our front-end tool if the query is ambiguous on one hand, and the rate of precision if the query is modified and becomes an unambiguous query. We cannot measure the recall because we fix the number of hit documents to 30.

2.4.1 Experimental Test Results

The results of our ambiguity tests – one without the use of our front-end tool, and therefore ambiguous, and another with the use of our front-end tool and hence modified and unambiguous – are as follows:

Ambiguous Query	Intended Definition	Results of Ambiguous Query (30 documents)		Modified, Unambiguous Query	Results of Unambiguous Query (30 documents)
orange	orange as fruit	orange as fruit : 15/30 orange as county: 7/30 orange as browser/website: 4/30 orange as an electronic company: 2/30 orange as bicycle store: 1/30 dead link: 1/30 **Precision: 50%**		***orange and fruit***	orange as fruit: 30/30 **Precision: 100%**
orange	orange as color	orange as fruit : 15/30 orange as county: 7/30 orange as browser/website: 4/30 orange as an electronic company: 2/30 orange as bicycle store: 1/30 dead link: 1/30 **Precision: 0%**		***orange and color***	orange as color: 30/30 **Precision: 100%**
bat	bat as a mammal	bat as an animal: 16/30 bat as a baseball equipment: 10/30 bat as a softball equipment: 2/30 bat as an email system: 2/30 bat as a tobacco group: 2/30 bat as an Israeli women group:1/30 **Precision: 53%**		***bat and mammal***	bat as a mammal: 30/30 **Precision: 100%**

bat	bat as a cricket instrument	bat as an animal: 16/30 bat as a baseball equipment: 10/30 bat as a softball equipment: 2/30 bat as an email system: 2/30 bat as a tobacco group: 2/30 bat as an Israeli women group:1/30 **Precision: 0%**		***bat and "cricket instrument"***	bat as a cricket instrument: 30/30 **Precision: 100%**
washington	washington as President of the United States	washington as an American state: 12/30 washington as a city in District of Columbia, USA: 11/30 washington as actor: 1/30 washington as magazine: 1/30 washington as a mountain: 2/30 washington as a university 2/30 washington as a brewery: 1/30 washington as a basketball league (NBA) member: 1/30 **Precision: 0%**		***washington and "President of the United States"***	washington as a President of the United States: 24/30 **Precision: 80%**
washington	washington as US national capital	washington as an American state: 12/30 washington as a city in District of Columbia, USA: 11/30 washington as actor: 1/30 washington as magazine: 1/30 washington as a mountain: 2/30 washington as a university 2/30 washington as a brewery: 1/30		***washington and "national capital"***	washington as an American state: 25/30 **Precision: 83%**

		washington as a basketball league (NBA) member: 1/30 **Precision: 37%**			
china	china as a dishware	china as an Asian country: 29/30 china as a dishware: 1/30 **Precision: 3%**		***china and dishware***	china as a dishware: 27/30 **Precision: 90%**
china	china as an Asian nation	china as an Asian country: 29/30 china as a dishware: 1/30 **Precision: 97%**		***china and "asian nation"***	china as an Asian country: 29/30 **Precision: 97%**
turkey	turkey as a poultry	turkey as animal/poultry: 13/30 turkey as a country: 16/30 Irrelevant hit: 1/30 **Precision: 43%**		***turkey and poultry***	turkey as a poultry: 24/30 **Precision: 80%**
turkey	turkey as a country	turkey as animal/poultry: 13/30 turkey as a country: 16/30 Irrelevant hit: 1/30 **Precision: 53%**		***turkey and country***	turkey as a country: 27/30 **Precision: 90%**
quebec	quebec as a province	quebec as a Canadian province: 23/30 quebec as a city: 6/30 Irrelevant hit: 1/30 **Precision: 77%**		***quebec and province***	quebec as a province: 30/30 **Precision: 100%**
quebec	quebec as a city	quebec as a Canadian province: 23/30		***quebec and city***	quebec as a city: 26/30

		quebec as a city: 6/30 Irrelevant hit: 1/30 **Precision: 20%**			**Precision: 87%**
mercury	mercury as a planet	mercury as an insurance business: 1/30 mercury as a car dealer: 2/30 mercury as car financing agent: 1/30 mercury as a car: 3/30 mercury as a hotel: 1/30 mercury as mortgage network: 1/30 mercury as a boat engine: 3/30 mercury as an Australian newspaper: 3/30 mercury as Washington news bureau: 1/30 mercury as a software company: 2/30 mercury as Sacramento news bureau: 1/30 mercury as Kansas newspaper: 1/30 mercury as San Jose newspaper: 2/30 mercury as music group band: 1/30 mercury as women's basketball team: 2/30 mercury as US record label: 1/30		*mercury and planet*	mercury as a planet: 30/30 **Precision: 100%**

		mercury as a programming language: 1/30 mercury as a planet: 1/30 mercury as a magazine: 1/30 irrelevant: 1/30 **Precision: 3%**			
mercury	mercury as a liquid metal	mercury as an insurance business: 1/30 mercury as a car dealer: 2/30 mercury as car financing agent: 1/30 mercury as a car: 3/30 mercury as a hotel: 1/30 mercury as mortgage network: 1/30 mercury as a boat engine: 3/30 mercury as an Australian newspaper: 3/30 mercury as Washington news bureau: 1/30 mercury as a software company: 2/30 mercury as Sacramento news bureau: 1/30 mercury as Kansas newspaper: 1/30 mercury as San Jose newspaper: 2/30 mercury as music group band: 1/30 mercury as women's basketball		***mercury and metal***	mercury as a metal: 29/30 **Precision: 97%**

		team: 2/30 mercury as US record label: 1/30 mercury as a programming language: 1/30 mercury as a planet: 1/30 mercury as a magazine: 1/30 irrelevant: 1/30 **Precision: 0%**			
mouse	mouse as an animal/rodent	mouse as a computer peripheral: 11/30 mouse as an animal: 11/30 mouse as a music group: 2/30 mouse as an internet service provider company: 1/30 mouse as a Spanish newspaper: 1/30 mouse as a computer learning school: 1/30 Irrelevant: 3/30 **Precision: 37%**		***mouse and rodent***	mouse as a rodent: 29/30 **Precision: 97%**
mouse	mouse as a computer peripheral	mouse as a computer peripheral: 11/30 mouse as an animal: 11/30 mouse as a music group: 2/30 mouse as an internet service provider company: 1/30 mouse as a Spanish newspaper: 1/30 mouse as a computer learning		***mouse and computer***	mouse as computer peripheral: 29/30 **Precision: 97%**

		school: 1/30 Irrelevant: 3/30 **Precision: 37%**			
java	java as a place in Indonesia	java as a computer programming tool/language: 30/30 **Precision: 0%**		***java and Indonesia***	java as a place in Indonesia: 29/30 **Precision: 100%**
java	java as a drink	java as a computer programming tool/language: 30/30 **Precision: 0%**		***java and drink***	java as a drink: 29/30 **Precision: 97%**
georgia	georgia as a US state	georgia as a US state: 24/30 georgia as an Asian country: 2/30 georgia as a search engine: 2/30 georgia as a paper manufacturer: 2/30 **Precision: 80%**		***georgia and USA***	georgia as a US state: 28/30 **Precision: 93%**
georgia	georgia as a country	georgia as a US state: 24/30 georgia as an Asian country: 2/30 georgia as a search engine: 2/30 georgia as a paper manufacturer: 2/30 **Precision: 7%**		***georgia and republic***	georgia as an Asian republic: 27/30 **Precision: 90%**

Table 2.1 – Experimental results on ambiguous queries, and on modified and unambiguous queries

2.4.2 Conclusion from the Experimental Results

We note that sending an ambiguous query to the present crop of search engines will give one all kinds of results, usually each document deals with a specific definition of the query word. An extreme example is the query word "*mercury*" in which we were able to obtained nineteen (19) possible "*meanings*" of mercury. We did not expect that mercury would mean a women's basketball team, for example. Another extreme example is the query word "*java*". Of the 30 documents that we tested, all of them dealt with only one meaning of java – as a programming language or software application. We did not get even a single document that deals with the Java place in Indonesia. Hence, we conclude that without any tool available on search engines to interpret the user's intention in case of an ambiguous query, a user is likely to receive all kinds of documents – most of them irrelevant.

The use of our tool – the front-end that detects user's ambiguity query and the user's additional keyword entry – is helpful in reducing the amount of irrelevant documents the user obtained from the Internet. To be specific, out of 20 experimental tests, 93% of the returned hits are relevant. Only a small 7% of the returned hits were irrelevant.

Chapter 3

Finding Relevant Documents for Focused Queries

A query to find relevant documents could come in two forms – focused or unfocused. We would like to address both.

3.1 Focused and Unfocused Queries

An *unfocused query* is one in which the user usually enters one– or two-or-more- word query or even a phrase to a search engine, and usually expects hit results in a form of documents whose contents are, to a general extent, dealing with the subject the user has queried. The contents of the hit documents are of general information to the query just entered, and as such the user is forced to read the document, and decide for himself which part of the document is worth gathering. An unfocused query is likely to be asked by an information seeker or anyone who wants to read or extract information on the subject being queried. An example of a focused query is a query like: "*air pollution*".

A *focused query* is usually a query that comes in form of a question, whether or not the query itself is terminated by a question mark. Such query usually needs documents containing precise answer as the query question itself is precise. Anyone who wants quick, on-the-spot, exact answer is likely to pose a focused query. An example of an unfocused query is a question like: "*Who is John F. Kennedy?*"

3.2 Rationale

With the Internet being used often as an educational tool, more and more students – from elementary, up to university level – are using the Internet to find information that are in one way or another related to their academic work. And so do others like researchers, entrepreneurs, and many others. Instead of staying too long in a local library finding one book from one shelve in one floor of the library building, to another book, to another shelve, to another floor of the same building – a feat that is indeed very exhausting – people now are more inclined to use the Internet to do just that, researching. And the research is not limited to a book alone, but on other media as well, such as newspaper, publications, reviews, and others.

The Internet, or for that matter the search engine, should address this continuing urge of people to know and get more information. Some of these needs are quest for general information, but others are quest for specific information. And the Internet should address both. However, the present crop of search engines are not equipped enough to handle some precise yet simple query questions such as: "*What is the atomic number of hydrogen?*" or "*Where is Taj Mahal located?*" This may not be very important to those who already know it, but they are indeed very important to those who are just beginning to learn it.

This thesis would like to contribute by giving some concepts and directions in finding relevant documents for unfocused queries.

3.3 Related Work and Literature Survey

Many attempts were made in the past to deal with exact question and to retrieve exact answer to the question. Most of the earlier work are all related to question answering category where exact answers rather than the document containing the exact answer is being retrieved. These work are general on exact answers rather than document retrieval. MURAX [3] is just one the many work that belongs to this group. It uses an online encyclopedia to answer the query. Also, it uses the context and linguistic analyses such as part-of-speech tagger and a lexico-syntactic pattern matcher in finding exact information. MURAX is not complete and so it can only be evaluated interim. In general, it is 74% capable of finding the right answer for questions "*who*" and "*what*".

Our approach in focused query is indeed document retrieval per se because it finds the documents, yet it is also information retrieval because it highlights the correct information within the document.

There are cases when document retrieval is preferable for focused query because although the query itself is precise and clear, the answer to the question may not be straightforward or may not be exact as the answer itself is relative to time. Take for example the focused query "*Who is the richest man on Earth?*" The richest man on earth in the year 2002 may not be the richest man in the year 2001. And so, if the hit document was written in 2001, the richest man would probably be Mr. X and another document written in 2002 may claim that the richest man on Earth is Mr. Y. Is any of the document wrong or misleading? The answer is no, both documents are correct. Because the answer is relative to time, then it is no accident that we have two different answers to the same query.

3.4 Theories and Concepts Applied

We consider that any query that begins with keywords *what*, *when*, *where*, *how* and *why* are automatically considered as a focused query. A focused query may or may not end with a question mark (?) but because the sentence or query begins with a question keyword, it is completely considered a question, and therefore a focused query. Whether or not the question keyword begins with a capital letter is immaterial. The first step therefore is getting a query from the user and deciding if such query is focused or not.

```
Algorithm for extracting keywords from the focused query
    assign count ← number of words in the query
    query ← ""                              //(null)
    n ←1
    while n ≤ count do
    begin
      word ← word #n
      if word ≠ negative word    // (refer to negative dictionary)
         then query ← query + " " + word
      end if
      n ← n +1
    end
```

Figure 3.1 Algorithm for extracting keywords from focused query

The next step to the process is to find out what the user wants to extract. This is done by eliminating some unnecessary words in the query, and leaving only the important keywords. Parts of speech such as linking verb (or the verb to be), preposition, and articles are taken out, and what is left then are the important keywords. Again, this is accomplished by parsing every single word (delimited by whitespace). Each of these words is compared with the words in the stop words list (please see Appendix C), if it is one of those keywords in the stop list, it is considered irrelevant in the query and is therefore taken out. The process is repeated until all words in the query are subjected to negative stop list test. Figure 3.1 shows the algorithm of the preceding steps.

Given below is an example of an unfocused query and the resulting query after the query is subjected to the algorithm given above:

Original query:	*Who is the richest man on earth?*
Question asked:	*who*
Resulting query:	*"richest", "man", "earth"*

Figure 3.2 Sample original query and the resulting modified query

The objective of the algorithm in Figure 3.1 is to edit the original query and come up with a modified query that includes as much as possible all the most important keywords contained in the original query. These keywords are essential because they have to be found in at least one paragraph in the HTML source code of the hit documents. The concept is that these keywords would not only all appear within the hit document, but also show up in positions almost adjacent to one another within a paragraph of the HTML source code of the hit document in order to consider that such document is addressing the user's query. Therefore, these keywords are very essential, and the selection of keywords should guarantee that only non-essential keywords are eliminated, and essential keywords are kept.

So, once the keywords are chosen, a query is submitted to the search engines, making sure all keywords appear in the hit document. From the given example in Figure 3.2 above, where we were trying to find the richest man on earth, the query that will be submitted to the search engines would be "**richest** *AND* **man** *AND* **earth**" or "**richest** *&* **man** *&* **earth**". We make use of AND logical operator within the keywords to make sure that at least all hit documents would contain these keywords. Once such a query is submitted to the search engine, we do expect to get some hit documents. These documents are saved as local files and analyzed almost simultaneously using multithreading.

The next challenge would then be finding which of these hit documents are relevant and which are not. It is important to take note that even if the keywords all appear in the documents, they may appear in different positions. Our hypothesis is that keywords position that is close enough or adjacent enough to each other (e.g. keywords in the same sentence) connote that the document is addressing our query, and keywords that are far enough from each other (e.g. keywords appear in separate paragraphs) are likely to connote that the document is not addressing our query and is therefore irrelevant.

It is worthy to mention that once those hit documents are saved as local files, the content of these HTML documents are not subjected to negative dictionary and negative tag test. This is done in order to keep the entirety of the document because some answers to the questions contain some less important keywords yet very important in answering the query. For example, the word "*because*" is generally not important in the general context of the document, but when the query begins with "*Why*", the presence of "*because*" is so important in determining the relevance of a document.

3.5 Answering the Query

Finding the answers to an unfocused query begins by finding all the keywords within the same paragraph of the hit document. Hence, the HTML tag pair *<P>* and *</P>* for paragraph are kept to be used as delimiter in finding the keywords and their location within the paragraph. The ideal condition is to find all these keywords within the same sentence, and even more ideal than that is finding these keywords within the title of the HTML document. However, relaxing the rule to finding the keywords within the same paragraph instead of same sentence will increase the likelihood of finding documents that answer the query. Failing to find these keywords with the same paragraph of a document will make such document irrelevant.

3.5.1 Answering the Query "Who"

When a user comes up with an unfocused query such as "*Who is Mahatma Gandhi?*" we know then that the resulting modified query would simply be "*Mahatma Gandhi*", and in all likelihood we will be getting all kinds of answers from the hit documents about Mahatma Gandhi. Indeed, it is so for there are so many ways to answer the question "*Who is Mahatma Gandhi?*" There is no single precise answer to such question as one can easily come up with answers like *"Mahatma Gandhi was the father of Indian nation."* or "*Mahatma Gandhi was a rebel Indian who liberated India from British colonization.*" or *"Mahatma Gandhi was a topnotch lawyer"*. All of these possible answers are correct.

Therefore, all documents that contain *"Mahatma Gandhi"* in its contents, or in general all documents that contain the modified resulting keywords in its contents are relevant. Since there is likely to be lot of hits to this query, document hits have to be ranked. The documents ranking would be similar to the way we ranked hit documents on a focused query, namely: (1) through the appearance of keywords in the title, (2) through the appearance of keywords in the headings and body, and (3) using our ranking algorithm, which in general favors documents in which keywords appear the most.

3.5.1.1 Experimental Results

We tested five (5) experimental queries to test our concept, and to determine if the concept itself is producing documents that are relevant to our standard. Here are the results:

Focused "Who" Query	***Modified Query***	***Search Engine Results considered Relevant by our system (30 documents)***	***Irrelevant Documents considered as Relevant by our system***	***Relevant Documents considered Irrelevant by our system***
Who is Mahatma Gandhi?	Mahatma and Gandhi	30/30	0	0
Who killed John F. Kennedy?	killed and John and F. and Kennedy	30/30	0	0
Who is the richest man on earth?	richest and man and earth	18/30	0	0
Who is the father of modern chemistry?	father and modern and chemistry	23/30	0	0
Who are the Acadians	Acadians	29/30	0	0

Table 3.1 – Experimental results on focused query "Who"

3.5.1.2 Performance Analysis of Experimental Results

In general, a front-end tool that is able to accept "Who" question from the user and in return be able to return back hit documents that contain the answer to the question is a step forward in the improvement of features of a search engine. The tests are limited, only five of them, yet the concept is working. More tests could be conducted, but may not be necessary because it is obvious that the future results would be consistent to the results we had obtained. Although it is based on keywords presence alone, we are able to obtain an average of 87% relevance hits.

3.5.2 Answering the Query "What"

By using our algorithm on extracting keywords on unfocused queries, a query such as *"What is liquid nitrogen?"* would yield a modified query that is "*liquid AND nitrogen*". The resulting modified query is very much the same as a focused query on "liquid nitrogen", and that, there can be so many possible answers to such question, and too many relevant documents.

Our approach to this kind of query is exactly the same as our approach on 3.5.1 Answering the Query "Who" – collect the hit documents and rank them accordingly.

3.5.2.1 Experimental Results

We tested five (5) experimental queries to test our concept, and to determine if the concept itself is producing documents that are relevant to our standard. Here are the results:

Focused "What" Query	*Modified Query*	*Search Engine Results considered Relevant by our system (30 documents)*	*Irrelevant Documents considered as Relevant by our system*	*Relevant Documents considered Irrelevant by our system*
What is the atomic number of oxygen?	atomic and number and oxygen	22/30	0	0
What elements compose water?	elements and compose and water	25/30	0* (see next page)	0
What is the capital of Greenland?	capital and Greenland	29/30	0	0
What is an alchemist?	alchemist	27/30	0	0
What science deals with the study of plants and animals?	science and study and plants and animals	28/30	21/28	0

Table 3.2 – Experimental results on focused query "What"

3.5.2.2 Performance Analysis of Experimental Results

For the focused query "what", the results are a mix of satisfactory and unsatisfactory results. Based on statistical results, 87% of the documents returned by search engines are picked up by our system and returned them as relevant. However, not all these documents are truly relevant. In the first place, relevance here through the presence of keywords. Although focused questions like "*What is the capital of Greenland?*" and "*What is an alchemist?*" returned hits that are truly relevant, queries like "*What elements compose water?*" and "*What science deals with the study of plants and animals?*" returned hits that are confusing.

For the query "*What elements compose water?*", 2/25 = 8% documents considered relevant by our front-end system deals with water being composed of atoms of hydrogen and oxygen, and 23/25 = 92% documents deals with the elements present in mineral water or commercially-bottled water. The reason behind this is that Internet is presently being used as a marketing tool, more than it is used as an educational tool. Therefore when we query about water, there are more hit results that concern mineral or commercially-bottled water than the usual water (H_2O) as discussed in chemistry courses.

We can say that water is now an ambiguous one-word query. Also on the question of the science that deals with the study of plants and animals, only about 7/28 = 25% of the documents considered relevant deals with biology or ecology. The rest are considered relevant due to the presence of the required keywords in the documents.

3.5.3 Answering the Query "Why"

Consider the unfocused query: "*Why was JFK assassinated?*" Following our algorithm to extract keywords from the unfocused query, the resulting query would be "*JFK AND assassinated*" (also, "*JFK & assassinated*") which can be transformed to "*JFK and assassination*" or "*JFK & assassination*" by converting the verb "assassinated" to its present tense form without s (assassinate) and converting the verb into its noun form. Easily, we would come up with three possible variations of the modified query keywords that are all correct in substance:

1. "*JFK and assassinated*" (also, "*JFK & assassinated*")
2. "*JKF and assassinate*" (also, "*JFK & assassinate*")
3. "*JFK and assassination*" (also, "*JFK & assassination*")

All three would be used as keywords to the query to be submitted to search engines. Again, we save these hit documents to the query and analyze them so that only those relevant documents would be selected and returned back to the user.

To answer this kind of query, we begin by finding the presence of the keywords (in our example, "*JFK*" and "*assassinated*" or "*assassinate*" or "*assassination*") within the same paragraph, or better still within the same sentence. Our search is limited to those contents bounded by *<P>* and *</P>* paragraph tag pair. If any document would not have the keywords in any of its paragraphs, the document is right away considered irrelevant.

Once a document contains the keywords within one of its paragraphs, we then begin searching for the presence of keywords like "*reason*" or "*because*" or "*due to*" or "*in order to*" or "*so that*" or a combination of any of them within the same paragraph. If such keywords appear within the paragraph, the document has a likelihood of being relevant, and considered relevant. The contrary would mean the document is irrelevant.

reason, because, due to, in order to, so that, need to

Figure 3.3 Keywords that denote reason for the query "why"

Assuming that some documents meet our selection criteria, the ranking of relevance would then depend on the position of the keywords within the paragraph. The highest ranking would be given to the document that contains the query keywords (in our example, "*JFK*" and "*assassinated*" or "*assassinate*" or "*assassination*") **plus** any of selected keywords like "*reason*" or "*because*" or "*due to*" or "*in order to*" or "*so that*" or a combination of any of them within the same sentence.

If assuming that two or more documents would meet those criteria mentioned above, the highest rank would be given to the document that has more occurrence of these in its paragraphs.

The lowest rank is therefore given to documents that have the query keywords plus the selected keywords, all appearing in the same paragraph but not in the same sentence.

3.5.3.1 Experimental Results

We tested five (5) experimental queries to test our concept, and to determine if the concept itself is producing documents that are relevant to our standard. Here are the results:

Focused "Why" Query	***Modified Query***	***Relevant Hits from Search Engines (30 documents)***	***Search Engine Results considered Relevant by our system***	***Irrelevant Documents considered as Relevant by our system***	***Relevant Documents considered Irrelevant by our system***
Why is smoking bad to our health?	smoking and bad and health	26/30	26/26	0	0
Why was JFK assassinated?	JFK and assassinated	25/30	3/25	0	2/25 documents
Why did George W. Bush reject the Kyoto Protocol?	George and W. and Bush and reject and Kyoto and Protocol	29/30	6/29	0	12/29 documents
Why is the ocean salty?	ocean and salty	29/30	28/29	0	0
Why is the color of sky blue?	color and sky and blue	27/30	17/27	0	0

Table 3.3 – Experimental results on focused query "Why"

3.5.3.2 Performance Analysis of Experimental Results

The focused query "Why" is more strict than the previous questions in considering documents that are relevant. For one, apart from the usual keywords in the query, we also are looking for extra keywords that denote reason, these keywords must be present within the same sentence or the same paragraph where the query keywords appear. Failure to have this pattern would mean that some documents are likely to end up being considered irrelevant. The table of results in the previous page confirm this. For example, for the focused query "*Why did George W. Bush reject the Kyoto Protocol?*", 12/29 = 41% of relevant documents are classified irrelevant by our system because the extra keywords we are looking for that denote reason do not exist.

We do acknowledge that some document writers (e.g. newspaper editors) do write with styles that do not usually follow the conventional pattern of answering the question "why". This is one weakness of a front-end tool that rely on keywords alone, such as our system. Another approach that could be incorporated in our design is the analysis of document by context. However, we do consider this as further future work.

The table above shows big discrepancy between the relevant results returned by search engines and those returned by our front-end system. Our system is more discriminating, but the ones considered relevant by our system really answer the question.

3.5.4 Answering the Query "When"

Assume that the unfocused query starts with the keyword "when" such as: "*When was the Canadian Federation founded?*" As with the other unfocused queries above, we will deal with this query by transforming the original query to a modified query using our algorithm to truncate unnecessary words, which in effect will yield the keywords "*Canadian Federation founded*". This will then be converted into a concatenated keywords with logical operator AND in between the keywords, giving us "*Canadian and Federation and founded*". We send the query to the search engines and have the hit documents stored as local file.

The next challenge is to find which documents among our hit results are relevant. By relevance, we mean a document that really answers this question. We begin our search for relevant documents by finding all the keywords within the paragraph of each document. If a document does not contain all of these query keywords within the same paragraph, then such document is considered irrelevant. If a document does have this, such document is a candidate to being considered relevant.

The next thing we do is to obtain the sentence(s) within the paragraph where the query keywords are found. We then look at some keywords that signify or connote time in this sentence. Obviously, when someone is asking <u>*when*</u>, the answer must be related with <u>*time*</u> or <u>*date*</u> or <u>*period*</u>. This is the reason why we look for some keywords that denote time.

We parse every single word in the sentence and match it with our time keywords database summarize below:

Day:	*Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday (also Mon.,Tue., Wed., Thu., Fri., Sat., Sun.)*
Month:	*January, February, March, April, May, June, July, August, September, October, November, December (also Jan., Feb., Mar., Apr., May, Jun., Jul, Aug., Sep., Oct., Nov., Dec.)*
Day:	*from 1 to 31*
Year:	*four digits*
Hour:	*numeric figures from 00:00 up to 23:59* *numeric figures from 12:00 up to 11:59*
Other Keywords:	*AM, PM, day, night, evening, noon, afternoon, night, o'clock*

Figure 3.4 Keywords that denote time for unfocused query "when"

Indeed, the most relevant document to the unfocused query is one that contains the query keywords and any of the keywords that denotes time (see Figure 3.4) within the same sentence. Other less relevant documents are those that contain the query keywords and any of the keywords that denote time within the same paragraph, but not necessarily in the same sentence. All other documents that only contain the query keywords but not any of the keywords that denote time will be considered irrelevant.

3.5.4.1 Experimental Results

We tested five (5) experimental queries to test our concept, and to determine if the concept itself is producing documents that are relevant to our standard. Here are the results:

Focused "When" Query	*Modified Query*	*Relevant Hits from Search Engines (30 documents)*	*Search Engine Results considered Relevant by our system*	*Irrelevant Documents considered as Relevant by our system*	*Relevant Documents considered Irrelevant by our system*
When is McGill University founded?	McGill and University and founded	26/30	23/26	0	0
When was JFK assassinated?	JFK and assassinated	30/30	30/30	0	0
When was the Eiffel Tower constructed?	Eiffel and Tower and constructed	27/30	23/27	0	0
When was the Montreal Summer Olympic Games?	Montreal and Summer and Olympic and Games	28/30	28/28	0	0
When did Apollo 11 landed on the moon?	Apollo and 11 and landed and moon	30/30	30/30	0	0

Table 3.4 – Experimental results on focused query "When"

3.5.4.2 Performance Analysis of Experimental Results

The system on "when" is less discriminating than the one on "why" query. This explains more why it has less rejection rate than the "Why" focused query. In the case of our experimental results, the rejection rate is zero.

The reason for good returns on "when" query is that usually the writer of the document is simply stating a fact. This is not the case for "why" query. For example, on the subject of President Kennedy's assassination, almost all documents about the subject could state exactly when was JFK assassinated, but very, very few would dare write about the reason why JFK was assassinated. And their reason may not necessarily be the fact - it could be the writer's mere opinion or perception.

Based on our tabulation in the previous page, we have 95% success rate of finding relevant documents on focused query "when".

3.5.5 Answering the Query "Where"

An unfocused query that begins with the keyword *"where"* is indeed one searching for answer that connotes a *"place"* or *"location"*. As there are so many possible questions that could begin with "where", and *so many distinct places* within the same house or building or town or city or country or continent, *it is impossible to come up with a* *database that denotes place*. And as such, we come up with different approach to answer the query *"where"*.

We all know that answers to the question that begins with "*where*" could be answered by a place or location, and in answering the question, we generally affix some proposition to the place or location. Consider a certain query to a little boy: "*Where is your mother?"* Let us consider a few possible answers. The little boy could say: "*She's in the bathroom*", or "*She is in the kitchen cooking*" or "*She's in the neighborhood gossiping with everybody"*. As we can see, the possible answers are *"bathroom*" or "*kitchen*" or "*neighborhood"*. But it is obvious that in the English language, these places are said with *preposition*.

in, on, at, to, along, beyond, above, about, above, across, after, against, along, among, around, at, before, behind, below, beneath, beside, between, beyond, but, by, despite, down, during, except, for, from, in, inside, into, like, near, of, off, on, onto, out, outside, over, past, since, through, throughout, till, to, toward, under, underneath, until, up, upon, with, within, without

Figure 3.5 Preposition keywords that usually appear with places or locations for unfocused query "where"

Indeed, we will be looking for the occurrence of some prepositions (keywords such as *"in"*, *"on"*, *"at"*, *"to"*, *"along"*, *"beyond"*, *"above"*, *"with*", etc.) along with the query keywords to determine the document's relevance. As before, we consider a document to be of highest relevance if the query keywords and one or more of these prepositions all appear in the same sentence of a paragraph. A less relevant document is one in which all the query keywords and the preposition(s) all appear in the same paragraph, but not in the same sentence. Other than these two, all other documents are irrelevant.

The steps in getting the user query, eliminating unnecessary keywords in the query, and submitting this modified query to search engines and getting the hit results stored as local file – all these processes will be done in here, and its details are all the same as before.

3.5.5.1 Experimental Results

We tested five (5) experimental queries to test our concept, and to determine if the concept itself is producing documents that are relevant to our standard. Here are the results:

Focused "Where" Query	*Modified Query*	*Relevant Hits from Search Engines (30 documents)*	*Search Engine Results considered Relevant by our system*	*Irrelevant Documents considered as Relevant by our system*	*Relevant Documents considered Irrelevant by our system*
Where is Taj Mahal located?	Taj and Mahal and located	27/30	26/27 * (see next page)	0	1/27 document
Where can I find the smallest fish in the world?	find and smallest and fish and world	26/30	10/26	3/10 documents	0
Where is the office of the President of Russia?	office and President of Russia	23/30	0/23	0	0
Where can I buy the cheapest helicopter in Canada?	buy and cheapest and helicopter and Canada	20/30	4/20	0	1/20 document
Where is the largest diamond deposit in Africa?	largest and diamond and deposit and Africa	24/30	18/24	0	0

Table 3.5 – Experimental results on focused query "Where"

3.5.5.2 Performance Analysis of Experimental Results

The focused query "Where" is supposed to be as discriminating as the focused query "When", but it is more vulnerable because the extra keywords that it is looking for is the preposition. Preposition, although more often attached to places, it could also be attached to other parts of speech such as noun, pronoun, and others. And hence, we do expect to obtain "noise" here, meaning irrelevant documents being considered as relevant by our system.

However, with some tough queries like "*Where is the office of the President of Russia?*" or "*Where can I buy the cheapest helicopter in Canada?*", our system is more likely to find irrelevant documents rather than relevant ones. The reason is there are very, very few documents that would answer such questions. On the specific focused query "*Where can I find the smallest fish in the world?*" which is not a tough question, our not so strict rule was able to find 3/10 = 33% irrelevant documents as relevant.

On the specific query "*Where is Taj Mahal located?*", it is not so surprising that "*Taj Mahal*" now is not only a famous landmark in Agra, India. The name also connotes a hotel, a restaurant, and a casino (and counting). And not so surprising as well is that these documents exactly tell us where the Trump Taj Mahal is located, as so is the Taj Mahal restaurant and Taj Mahal hotel. Since the query was simply "*Taj Mahal*", all these extra "baggages" are also considered.

3.5.6 Answering the Query "How"

An unfocused query that begins with the keyword "<u>*how*</u>" is usually demanding an answer for (1) some steps or procedures to follow; (2) the condition or situation; (3) the explanation or detail of something (an event or a mechanism). There are many questions that fit those patterns, such as: (1) "*How to make yogurt?"*; (2) "*How did the twins' operation go?"*; and (3) *"How does the exposure to radioactive material affect a person?"*. There are many ways to answer just one of the three questions presented above. For example, to make yogurt, one would say: "*Step 1: Do this, Step 2: do that, and so on"*. Another person would probably answer this way: *"First, you do this, then you do that, and so on*". And yet another person would answer the same question by saying, *"Get a measuring glass, a pint of milk, etc."*. Indeed, there is *no precise pattern* by which human would answer the question "how" in natural language.

To answer the query "how", we could not come up with a list of precise keywords that answers "how" because there are infinite possibilities. The alternative solution then is finding the modified query keywords in the title or heading or in the same sentence of the hit document's paragraph. To answer the unfocused query "How to make yogurt?", we will simply be looking for "make" (or its variation, "making") and "yogurt" in the title, in the heading, or within the same sentence in the paragraph(s) of the hit document.

The ranking of the hit documents then will be based on ranking as if the query is a focused query.

3.5.6.1 Experimental Results

We tested five (5) experimental queries to test our concept, and to determine if the concept itself is producing documents that are relevant to our standard. Here are the results:

Focused "How" Query	*Modified Query*	*Relevant Hits from Search Engines (30 documents)*	*Search Engine Results considered Relevant by our system*	*Irrelevant Documents considered as Relevant by our system*	*Relevant Documents considered Irrelevant by our system*
How does the operating system work?	operating and system and work	28/30	28/28	0	0
How does alcohol affect our judgment?	alcohol and affect and judgment	29/30	29/29	0	0
How does the ship float in water?	ship and float and water	30/30	30/30	0	0
How to invest in stock market?	invest and stock and market	30/30	30/30	0	0
How to bake a croissant?	bake and croissant	28/30	2/28	0	2/30 documents

Table 3.6 – Experimental results on focused query "How"

3.5.6.2 Performance Analysis of Experimental Results

The experimental results of five (5) focused queries on "how" indicate high return of relevant documents. This is due to the fact that there are no additional discriminating keywords that are needed, apart from the usual query keywords, that have to appear in every document. With more lax rules, more hits are obtained. For the specific query "*How to bake a croissant*", we obtained 2/30 = 7% relevant documents considered irrelevant because the documents are written in languages other than English. And since we could not find the required English keywords in the documents, these documents are considered irrelevant, although we must admit that their contents are relevant.

It is a very common experience that we obtain documents written in French for focused queries written in English. Probably because Canada is a bilingual country that we obtain most documents in English and a few documents in French. Our front-end tool is not equipped with French translation so documents written in French that do not contain the required keywords are automatically rejected and treated as irrelevant.

Other than the above-mentioned problem, the focused query on "how" generally obtains a good number of relevant documents.

3.6 Future Direction

The concepts on focused queries mentioned in this chapter address way and means of finding the relevant documents that answers the question. This, we believe, is a step forward into making Internet information search an agreeable experience. We can also implement the query disambiguation in the future. This is possible to do, although we did not put into reality for this thesis due to the time constraint in our study at Concordia University. A focused query can also be ambiguous. Consider for instance a focused query of a non-Canadian individual: "*Where can I find Quebec?*". It is possible that he is asking "*Where can I find Quebec Province*" and equally possible that he is asking "*Where can I find Quebec City?*". Another ambiguous focused query from a child could be: "*What is the capital of Georgia?*". Is he asking Georgia, the country or Georgia, the US state?

The concept of disambiguation of a focused query and obtaining a high precision score of retrieve documents would truly be a good direction to proceed.

Chapter 4

Classifying Documents as per Academic Relevance

This chapter proposes that it is possible and feasible for Internet search engines to add this feature – classifying Internet documents. This work proposes an academic classification of documents since this thesis is academic in nature. This concept can be expanded to accommodate other forms of classification, say as per business, commercial, military relevance, etc.

4.1 Rationale

When one searches documents on the Internet using commercial search engines (e.g. *Google, Hotbot, AltaVista, Infoseek, Excite*, etc.), he/she does enter a query, then gets a list of document hits, along with every hit document's title, relevance score, and the first few words from the document. One has to download each of these hit documents to know more about the document's identity since commercial search engines do not "*identify*" or "*classify*" these documents other than those parameters already mentioned above. What is lacking is a kind of document identification or classification that identifies the nature of the document. Moving towards this goal, it is suggested that along with the relevance score and title of the document, a category should be attached to the document. This will provide Internet users information of the nature of the document even before browsing it.

It is believed that a search engine that provides document hits to the query, and also each document's title, URL address, first few words from the document, and the classification of a document would give Internet users some idea about the document even before downloading or browsing it. In effect, this would enhance efficiency and productivity on the part of the users because they will be spared of the some unnecessary downloading and browsing time. Time spent on needless browsing of documents is wasted time that should be minimized, if not downright eliminated. And this work believes that providing classification of document on search engine hits is a contribution that will reduce these unnecessary downloading and browsing time spent on undesirable documents.

Assume, for example, a case of a student who would like to browse documents related to "*software engineering*". He enters this query to a search engine, and in return the search engine provides him hits documents that list down not only the relevance score and title of documents, but also the classification of documents (i.e. Frequently Asked Questions, Course Notes, etc.) he would most likely download those documents whose classification is related to his own academic needs. He would spare himself of time downloading documents that are not related academically.

This study believes that documents can be classified in many ways based on many variables. Every document can be classified as to whether or not it is a military document, a commercial one, an academic one, even a medical or entertainment one. Inasmuch as this study cannot cover all the above-mentioned classifications on a limited time, it is then decided to limit the classification on only the academic nature of the document.

4.2 Academic Classification of Documents

To make up for our limited stay in the university, our classification of documents is aimed at purely academic classification only. We have limited the academic classification to be any of the following:

(i) *Course Notes,*
(ii) *FAQ (Frequently Asked Questions),*
(iii) *Research Papers,*
(iv) *Technical Reports,*
(v) *Theses,*
(vi) *Tutorials,*
(vii) *Reviews, or*
(viii) *Research Papers/Technical Reports.*

The rationale for the choice of the eight classes of academic category above is based on our perceived importance of these categories to students and academic researchers.

The content of a document categorized as "*Course Notes*" is assumed to be written by a professor or a lecturer of a course. Taken from this perspective, a course note document should be very important to students and researchers.

A "*FAQ (Frequently-Asked-Questions)*" document is kind of lecture in itself, containing questions that could usually come from someone unfamiliar with the subject, and also contains answers that could usually come from an expert. Since a student or a researcher is usually interested in finding information, it would make a lot of sense to include FAQ as a class within an academic category.

Research Papers and *Technical Reports* are some of the stuff that a student does in a college or university. A document categorized as any of the two would definitely be an important piece of information for students and researchers.

Thesis is a category that researchers in masters and PhD graduate study level would be interested in. It therefore makes logic to add it as additional academic category. A *tutorial* document would be similar to course notes – another important academic piece of information.

A "*review*" category may probably be of lesser academic significance as compared to other categories mentioned above. A review nonetheless is important since some students in as early as high school are usually required to write some critical review of a book or film or other media of academic significance.

Of course looking beyond, if the intent is to upgrade this work in the future, further categorization – perhaps on the field where the document is intended (e.g. commercial publicity, academic, etc.) – would be added to this work. Likewise, it can be foreseen that this document classification would not only be limited to text document, as how it is right now. It is always possible to classify documents on other media as well – media such as audio, video, and still images.

4.3 Related Work and Literature Survey

Previous work on document classification were generally done using Bayesian classifier. The work of Borgelt, Duda, and Hart, et al [7, 8, 9] made full use of Bayesian classifier or the naïve Bayes classifier to produce comprehensive results in many data analysis tasks.

Naïve Bayes classifiers [2, 8, 9] are an old and well-known type of classifiers. Classifiers, in turn, are programs that automatically classify a case or an object, i.e. assign it according to its features to one of several predefined classes. For example, if the cases are patients in a hospital, the attributes are properties of the patients (e.g. sex, age, etc.) and their symptoms (e.g. fever, high blood pressure, etc.), the classes may be diseases or some drugs to administer. The naïve Bayes classifiers use a probabilistic approach to classify data. The classifiers try to compute the conditional probabilities of the different classes based on the values of the attributes, and then predict the class that has the highest conditional probabilities.

Our approach is a little similar in concept to the Bayes classifiers because some parts of our work rely on the frequency of keywords present in a document, and the summation of the score of these keywords are used as an indicator to probabilistically infer that such a document belongs to a particular category.

The training of the corpus of academic documents is very important as the keywords are essential in determining the classification of a document. Unlike in Bayes classifiers where everything is based on the calculation of the probability score, our categorization work is partly based on the probabilistic score, and also on the presence of keywords on important locations in a HTML document. We infer that some presence of the keywords in important locations (e.g keywords appearing in the title) does not need calculation to be categorized in a particular class. Rather, the mere presence of keywords in such a location is already a confirmation that a document belongs to such a category.

The drawback of this approach is that we always believe in the veracity of the title or heading of the document's writer. If the document's writer or author is poor in choosing the right title or headings in his documents, that would be reflected in our categorization of his document as well.

4.4 Theories and Concepts Applied

Classifying a document means figuring out its syntactic content. However, each document, course notes for example, does not follow a particular format or pattern since every document of the same nature is unique and different from others. The question that we could possibly asked now is something like, "*among these different documents of the* *same nature, what is their common characteristic so that each one would be classified accordingly, and they all be classified as belonging to the same class?"* Classifying a document as to its academic use involves finding keywords in the right places.

4.4.1. Keywords in the title

In analyzing HTML documents, there is an utmost importance to the title of the document. The title of the document, in one way or another, summarizes its content. If for example, one goes to visit the URL *http://www.cs.concordia.ca*, and open up its page source, one shall see that it's document title is "*Department of Computer Science*". True enough, the document itself is nothing but a documentation of Concordia University's Computer Science Department. In general, the titles of the HTML documents that one finds in the Internet do mirror or summarize their contents. Hence, if the title, for example, is "*FAQ on Unix and Other Operating Sy*stems", then one can believe that it is a FAQ (Frequently Asked Questions) document even without going through its content. One can believe that if a document is correctly done, then the document title indeed mirrors or summarizes the document contents. And so the document title is important, and this same title will be one factor to consider in classifying a document. Having that in hand, the challenge now is narrowed down to identifying the rightful keywords in the title that will assist to correctly classifying a document.

Some documents are very easy to classify as their titles will say it all. Consider for example titles such as:

"*Programming Course– Course notes*"

(http://www.strath.ac.uk/CC/Courses/Ccourse/Ccourse.html),

or

"*CMPUT 201 Course Notes*"

(http://wwwcs.ualberta.ca/~mark/c201/index.html).

The presence of "*course notes*" signifies that these documents are indeed course notes documents. The rule, therefore, is to find the document category keyword in the title. However, there are cases when two or more keywords identifying a document appear in the title. In such a case, the position of the keywords and of the preposition "***on***" will assist us in the classification.

Consider, for example, a document with a title "*Course Notes on Thesis and Technical Report Writing*". Is this document a technical report? A Thesis? Or Course Notes? All three keywords are deemed essential in this case, but the position of the keyword is equally essential as well. The position of "*course notes*" and the preposition "*on*" make this document logically correct to be classified as a course notes, and not a thesis nor a technical report. In the presence of the preposition "ON", the keyword(s) preceding it becomes important and the keyword(s) following it non-important. It is therefore important that document title be parsed to determine the presence of the preposition "ON". In such a case, keywords preceding it have to be taken, while keywords following it have to be discarded.

It is also common to see that two or more keywords appear on the title without a conflict. Consider the title *"Unix Operating System - Course Notes, FAQ"*. As the title suggests, it is all about Unix operating system, and in this document, it contains course notes on the subject, and a FAQ as well on the subject. And so, what is the classification of this document? It is both a course notes, and a FAQ document.

4.4.2 Finding the Right Keywords in the Right Places

All text documents that we find in the Internet through web browsers like Netscape or Internet Explorer do follow a format. This format is called HTML (Hypertext Mark-up Language). An HTML document is written with text and appropriate tags associated on the text. The title, for example, of a document is sandwiched within the tag pairs *<TITLE>* and *</TITLE>*. Once the tag pairs are properly identified, the next task is to find the text within these delimiters and parse the text accordingly.

The parsing begins with a search for tag pair *<TITLE>* *</TITLE>*. Anything that is inside this tag pair is the title of the document. The classification of document by title will only take place if these tags are present. In the absence of these tags, the document is considered "*Untitled*". Any "untitled" document therefore fails in the classification test in the title, and will be subjected to the test in the second option – Finding keywords in the heading.

The keyword, if it appears as the first word, confirms the document's category. For example, a title like, "*FAQ on Information Retrieval*" will be taken that this document is indeed a FAQ document.

The keyword if it appears anywhere but first, should be taken that the document is indeed of the expected category provided that it satisfies some prepositional requirement. For example, a title like "*Information Retrieval – Frequently Asked Questions*" will be taken as a FAQ document.

The presence of the preposition ON before the keyword (e.g. *Lectures on Thesis Writing*) will be considered as a negation that the document is so even in the presence of the keyword. The example above contains the keyword "Thesis", but the document cannot be considered as a thesis, rather it should be considered as a lecture.

Likewise, it is also possible that two keywords could appear in the title and accordingly, the document should be classified as both. For example, the document with a title "*Operating System - Course Notes and Frequently Asked Questions*" should be classified both as a course note document and as a FAQ document.

Success in the classification of documents based on its title ends the process, and will proceed to finding the relevance score of the document. In case of failure, further tests have to be conducted, the second of which follows.

4.4.3 Keywords in the headings

There are cases when a document exhibits characteristics of a particular document, yet failed the first test above simply because important keywords did not appear in its title. Consider for example a document entitled:

"*Department of Computer Science - Cpsc 244 - Don S. Bidulock*"

(http://www.cpsc.ucalgary.ca/~dsb/Cpsc244/).

This document is all about course Cpsc 244 – Introduction to Computer Science II at University of Calgary under Prof. Don S. Bidulock. Browsing this document, one will find out that it is all course notes on Cpsc 244, the course title they use to denote Introduction to Computer Science II in the same university. However, it failed the first test (Keywords in the title) because its title does not say that it is a collection of course notes. However its headings say so, hence this second test.

The search for keywords in this regard will be restricted to tag pairs in the following order:

<*H1*> and </*H1*>,
<*H2*> and </*H2*>,
<*H3*> and </*H3*>
<*H4*> and </*H4*>,
<*H5*> and </*H5*>,
<*H6*> and </*H6*>.

A priority scheme is established such that tags <H1> and </H1> will be searched for the keywords. If the keywords are present and enough to classify a document, then the parsing is done and the document is properly classified. Only on its failure the search will go onto <H2> and </H2> pair. The same rule will be applied before the next search will go onto <H4> and </H4> pair, and so on.

The method of identifying keywords in heading follows the same rule as the first one, identifying keywords in the title.

4.4.4 Keywords in the body

There are few cases where documents exhibit a category characteristics yet are not classified simply because their titles or headings do not include the keywords that we are looking for. In cases like these, we subject the document to the final test – keywords in its body.

Classifying a document category based on keywords in its body will only be implemented if finding necessary keywords in the title and in the heading fails to identify a document.

4.4.5 Identifying keywords that describe the document

The process begins with parsing the HTML document and taking its text, in a way transforming an HTML document into a text document. The text then is subjected to stop words list and stop symbol lists. In here, words that are frequent in any document but in no way describe a document are taken out. Words such as *is, are, in, of, by, among, which, that*, etc. are words that play no meaning to the document, and therefore have to be omitted. The same thing applies to stop symbol list like punctuation, and symbols like $, %, ", ?, ., etc. have to be taken away. When all these things are done, the leftover of the document are list of keywords that to a certain extent describe a document.

Let us apply the concept discussed above over statement *like:*

"*The quick brown fox jumps over the lazy dog near the bank of the river*".

Getting rid of common words, the above statement would be left as:

"*quick brown fox jumps lazy dog bank river*".

4.4.6 Eliminate non-essential keywords

Consider a course-notes document in which the word "*executable*" appears once, the word "*array*" appears twice, and the word "*computer*" six times, and "*course notes*" appear 18 times. It is safe to say that although "executable" is not a common word and not belonging to stop words, it is a non-essential word in this case. Deleting it will not at all affect the fact that this document is a course note that discusses among others computer and array.

It is therefore believed that distinct keywords that *appear only once* in the document are non-essential keywords, and omitting them from the list of document keywords would not affect that much. After their omission, what is left are distinct keywords that truly represent the document.

4.4.7 Minimum score requirement

Among the keywords left behind by the series of deleting non-essential words, there is a keyword that appears the most. Let us call the frequency of this keyword **TFi_max**. This is that keyword whose frequency is the highest among all the other keywords left to describe the document.

Next, we determine the frequency of our search keyword(s), call it **TFi_j** (*see detailed calculation of TFi_j on section 5.2.6*). The ratio of TFi_j relative to TFi_max is the relevance score, that is:

relevance score = TFi_j / TFi_max.

This study believes that for a document to be classified as belonging to a certain category, the relevance score should be at least 60%. We then say that *this document has a 60% probability of being in this document category*. 60%, therefore, is the cut-off percentage limit. Lower than this score, the document fails to be classified as belonging to this document category.

4.4.8 Frequency of document keywords vs. Keywords in the database

As an ultimate test, if all else fail, we will subject the document keywords to a test of whether or not they correspond to some of the document category keywords in the database, and if so whether their frequency in the document is dominant for the document to be classified as one belonging to that document category. It is to be noted, however, documents do not need to undergo this test in order to be classified in certain categories. Please refer to *Appendix B – Keywords and Methods Used in Categorizing Documents By Academic Relevance* for more details.

What are then the keywords used to identify a particular document category? Well, we would have a database of keywords and each HTML document is subjected to this keyword test. A minimum amount of occurrence of these keywords must be present in a document for it to be so classified.

The keyword database is based on the most common keywords that usually appear in documents that belong to a certain academic category. These keywords must appear at least 60% in the body of the document, and can then be said to have 60% probability of belonging to this academic relevance category.

The keywords in the database is not based on training but rather based on our perception or intuition of what common words appear in academic documents that belong to such a category. For example, irregardless of what the subject matter is all about, a FAQ document always contains questions and answers, and the most dominant keywords in a FAQ document are question, answer or a variation of them such as Q: (for question) and A: (for answer). We took sample documents that belong to the same category and listed down common keywords. These keywords are the ones stored in our database.

4.5 Experimental Tests

We subjected our thesis concepts for this chapter in some tests to prove that our idea is feasible. There were only five (5) subjects we queried about, but each subject is considered in two ways – one being the subject area itself (e.g. pattern recognition), and the other being the subject area plus a particular category (e.g. "pattern recognition" and "research paper"). The reason is because we would like to analyze in which manner would we be getting more relevant academic documents. We analyzed the results and suggested ways and means to facilitate retrieval of relevant academic documents.

4.5.1 Experimental Test Results

We have tested a pair of five (5) original and modified queries on subjects related to computer science, and evaluate the academic category of each of the 30 documents that were returned by search engines for the query. The table below shows the results of the test that we had conducted. Although the tests are limited to a pair of 5 original and modified queries, we do expect to obtain consistent results as the table below should be conduct more tests, say 10. Here are the results of our experimental test:

General Query	***Document Classification (30 documents)***	***Modified Query :***	***Document Classification (30 documents)***
operating system	Course Notes: 0/30 FAQ: 0/30 Research Paper : 0/30 Technical Report : 0/30 Thesis: 0/30 Tutorial: 0/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 29/30 Irrelevant: 1/30	*"operating system" and tutorial*	Course Notes : 1/30 FAQ: 2/30 Research Paper : 0/30 Technical Report: 0/30 Thesis: 0/30 Tutorial: 18/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 9/30 Irrelevant: 1/30
artificial intelligence	Course Notes: 6/30 FAQ: 6/30 Research Paper: 0/30 Technical Report : 0/30 Thesis: 0/30 Tutorial: 0/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 14/30 Irrelevant: 1/30 Dead link: 2/30	*"artificial intelligence" and "course notes"*	Course Notes: 22/30 FAQ: 1/30 Research Paper: 0/30 Technical Report: 0/30 Thesis: 0/30 Tutorial: 2/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 4/30
pattern recognition	Course Notes: 5/30 FAQ: 0/30 Research Paper: 0/30 Technical Report : 0/30 Thesis: 0/30 Tutorial: 2/30	*"Pattern recognition" and "research paper"*	Course Notes: 8/30 FAQ: 0/30 Research Paper : 20/30 Technical Report: 1/30 Thesis: 0/30 Tutorial: 0/30

	Review: 4/30 Research Paper/Technical Report: 10/30 Unclassified: 14/30 Dead link: 1/30		Review: 0/30 Research Paper/Technical Report: 1/30 Unclassified: 5/30
data structures	Course Notes: 6/30 FAQ: 0/30 Research Paper: 0/30 Technical Report: 0/30 Thesis: 0/30 Tutorial: 0/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 23/30 Irrelevant: 1/30	*"data structures" and FAQ*	Course Notes: 3/30 FAQ: 21/30 Research Paper : 3/30 Technical Report : 0/30 Thesis: 0/30 Tutorial: 0/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 3/30
C++ programming	Course Notes: 0/30 FAQ: 0/30 Research Paper: 0/30 Technical Report: 0/30 Thesis: 0/30 Tutorial: 4/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 26/30	*"C++ programming" and "course notes"*	Course Notes: 21/30 FAQ: 2/30 Research Paper: 0/30 Technical Report: 0/30 Thesis: 0/30 Tutorial: 3/30 Review: 0/30 Research Paper/Technical Report: 0/30 Unclassified: 4/30

Table 4.1 Experimental results of classification of documents as per academic relevance based on user's query

4.5.2 Performance Analysis of Experimental Results

The result of the tabulation of our tests, Table 4.1, indicates important findings: (1) if we rely on search engines to obtain academic documents based on a mere simple query (e.g. unmodified query such as "pattern recognition), we are likely not to get enough relevant documents based on the category that we want, and (2) there is a need for an academic category to be present in the search engine – ready to be ticked by the user, and the category itself would be added to the original query.

As cited earlier in this thesis, the Internet is being used for commercial purposes – publicity, marketing, etc. The academic documents on the Web are usually authored by university professors and lecturers. If we compare the recall of some documents, there is likelihood that we are going to obtain more documents written by business establishments than academic institutions. For sample query like "operating system", there are more commercial topics on selling a particular operating system than a lecture or course notes on operating system. Hence, we hypothesize that there is little chance that we get good recall of academic documents on such areas as operating system, pattern recognition, etc.

One way to improve recall and precision to obtain academic documents is to attach a category to the query word (e.g "operating system" and FAQ). The recall and precision will be much better than a mere "operating system" query. Also, it is possible that some features like this be present in search engines that a user may click or tick off after he inputs a query. This is one way to revolutionize the present crops of search engines.

Chapter 5

Focused Queries, and Ranking and Merging Documents from Search Engines

5.1 Rationale

It is important that user's query in the Internet gets as much relevant hits as possible. Different search engines have different databases, so chances are each search engine provides unique hit results, different from other search engines.

One of this thesis's goals is to come up with relevant hits to a user's query, meaning that each hit that it will provide would have its contents contain among others subject which the user has entered as his query. To be more specific, the relevance score of each hit document would be not less than 60%. And this relevance score would be dependent on the document content, not on the score that the commercial search engines provide.

The 60% relevance score is based on a common perception that a score should not be too high and not to be too low as well. Minimum score of 80%, for example, would mean that the discriminating score is too high, and that we are going to take out many documents from consideration. On the other hand, the score of 50% is very low – it has a meaning that it is 50% likely that the document is relevant, yet 50% likely as well that the document is irrelevant. The score 60% is somewhere in between. Besides, this score is not fixed at all times; the user could modify the front-end to suit his/her minimum scoring needs.

Another objective of this study is to come up with as many hits as possible. That is to say that document hits would not come up one search engine result alone, but rather to three to five search engines. The results of these search engines are then collated and are arranged accordingly.

5.2 Theories and Concepts Applied

To begin with, the relevance of a document is a factor of the presence or absence of important keywords in the document. In all likelihood, such important keywords should include the query keywords. It is therefore imperative that the researcher should apply TF (term frequency) document relevance scoring as one component in the document scoring scheme. The TF method measures the number of appearances (frequency) of a term or group of terms in comparison to all other terms that appear in the document. As the nature of documents is evaluated, other component(s) could be added to the relevance scoring scheme.

5.2.1 Choose the search engines to work on for the query

This work intends to act as an interface between the user searching relevant documents in the Internet on one hand, and the commercial search engines on the other hand. That is to say that users will enter query and the system will find the documents – relevant documents – for him.

The process begins by determining in which search engines should be used to find relevant documents. It is decided that the system could have either default values or an option for the user to choose the search engines to work on.

As a default value, the proposed system has three to five commercial search engines to work on. They are as follows:

Names	Main Site
Excite	*www.excite.com*
Hotbot	*hotbot.lycos.com*
Googles	*www.googles.com*
Webcrawler	*www.webcrawler.com*
AltaVista	*www.altavista.com*

Fully aware that some user's may not necessarily be satisfied with restrictions on which search engines to search from, the study has decided to come up with an option of allowing user to choose search engines to work on. Should the user decide on choosing the later, all subsequent searches will be based on the list user's choice of search engines.

5.2.2 User's query is analyzed – is it a focused query, an ambiguous query, or an unfocused, non-ambiguous query?

The user will be allowed to enter query. Based on the user's query, an action to be done is decided, which is one of the following:

1. If the query is a single-word query, it is subjected to an ambiguity test. If it is found to be an ambiguous query, the steps to be done is same as that discussed in *Chapter 2 - Identifying and Providing Directions to Ambiguous Queries.*
2. If the query is an focused query, in a form of a question, the steps to be done is the same as that of *Chapter 3 - Finding Relevant Documents for Unfocused Queries.*
3. If the query is a unfocused, non-ambiguous, the steps to be done are further discussed in *this chapter.*

The next step involves finding out if the query is a single word, or two or more words with the necessary Boolean operation, or whether the query is a phase. This is important because each of them must be treated differently.

A single-word query is complete in itself. A query with more than one keyword will probably have a one or more boolean operations involved, or probably the query may come in as a phrase. Consider the following variations of user query:

	Format	Example	Meaning
1	*<keyword1>*	pollution	Search will be concentrated on pollution
2	*<word1>* + *<word2>*	noise + pollution	Search will be concentrated on either just noise, or pollution, or noise pollution.
3	*<word1>* & *<word2>*	operating & system	Search will concentrate on finding keywords operating and system. They may not necessarily come one word after another
4	*"<phrase>"*	"research paper"	Search will concentrate on looking for the exact match of the phrase in the document.
5	*<word1>* OR *<word2>*	thesis OR review	Search will concentrate on either word1, or word2, or both. Similar to action for format #2.
6	*<word1>* AND *<word2>*	research AND paper	Search concentration and action are similar to that in format #3.
7	<word1> <word2>	course notes	Search will be concentrated on either word1, or word2, or both.

Table 5.1 The Boolean operations in users query and their implications in information retrieval

It is expected that user's query will generally fall in any one of the seven formats given above. The keywords in the query are examined and the boolean operation is determined as well.

The boolean operator *OR* or + will be treated as it is – just the presence of one of the keywords in the query is enough to qualify that the document is a hit, although the document gets a probable higher ranking if it contains two or all of the keywords in the query.

The boolean operator *AND* or & will mean that the search for a matching document for the user query will concentrate on the presence of both keywords, or in general all the keywords in the query, but these keywords would not necessarily come one after another in the sequence. If only one of the keyword in the query appears in the document, then the document is considered irrelevant.

The phrase notation (" ") will mean that the search will be concentrated in finding exact copies of the phrase in the document. The absence of exact copy of the phrase in the document will mean that the document is irrelevant to the query.

Using the queries about *"air pollution"* and the possible Boolean operations involve, here are some sample focused queries, and the search engines websites that contains the hit results to these queries.

No.	***Boolean Operation***	***Sample Query***	***Resulting Hotbot URL***
1	*OR*	***air*** *OR* ***pollution*** *air + pollution*	*http://www.hotbot.com/?MT=**air%2Bpollution**&_v=2&Ops=MDRTP*
2	*AND*	***air*** *AND* ***pollution*** *air & pollution*	*http://www.hotbot.com/?MT=**air%26pollution**&_v=2&Ops=MDRTP*
3	*Phrase*	***"air pollution"***	*http://www.hotbot.com/?=%22**air+pollution**&_v=2&Ops=MDRTP*
4	*None*	***pollution***	*http://www.hotbot.com/?comefrom=nspanel-search&MT=**pollution***

Table 5.2 Possible combinations of queries about "air pollution" and the Hotbot URL's that contain hit documents on the queries

No.	***Boolean operation***	***Sample Query***	***Resulting Lycos URL***
1	*OR*	***air** OR **pollution*** *air + pollution*	<u>*http://www.lycos.com/cgi-bin/pursuit?cat=dir&maxhits=10&query=**air** %2B**pollution**&npl=*</u>
2	*AND*	***air** AND **pollution*** *air & pollution*	<u>*http://www.lycos.com/cgi-bin/pursuit?cat=dir&maxhits=10&query**air**%26**pollution**+npl=*</u>
3	*Phrase*	***"air pollution"***	<u>*http://www.lycos.com/cgi-bin/pursuit?cat=dir&maxhits=10&query=%22**air+pollution**%22&npl=*</u>
4	*None*	***pollution***	<u>*http://www.lycos.com/cgi-bin/pursuit?query=**pollution**&cat=dir&maxhits=10*</u>

Table 5.3 Possible combinations of queries about "air pollution" and the Lycos URL's that contain hit documents on the queries

No.	**Boolean operation**	**Sample Query**	**Resulting Excite URL**
1	*OR*	***air** OR **pollution*** *air + pollution*	<u>*http://search.excite.com/search.gw?search=**air** %2B**pollution**&tsug=-*</u> *1&csug=10&sorig=netscape*
2	*AND*	***air** AND **pollution*** *air & pollution*	<u>*http://search.excite.com/search.gw?search=**air***</u> *%26**pollution**&sorig=netscape*
3	*Phrase*	***"air pollution"***	*http://search.excite.com/search.gw?search=%22 **air+pollution**%22&sorig=netscape*
4	*None*	***pollution***	<u>*http://search.excite.com/search.gw?search=**pollution**&trace=1&src=nsl&sorig=netscape*</u>

Table 5.4 Possible combinations of queries about "air pollution" and the Excite URL's that contain hit documents on the queries

5.2.3 Search engines are searched for document hits; documents' URLs are collated.

The query given in 5.2.2 will be searched on using the search engines given in 5.2.1. This is in effect somewhat like as if the user submitted the query himself to these search engines simultaneously.

In order to accomplish the task mentioned above, the study involved sampling different queries of different formats on these search engines and found the URL of the search engine result. From this, the researcher deduced the necessary URL for any given query of any format. This is done by truncating the query keywords from the resulting URL address.

The actual Java program implementation of this concept begins by attaching the query words and the operation to a fix URL address (each URL address is different for every search engine employed). The URL is searched online and an HTML copy of this document is saved as a text file locally. The same thing is applied to the results of the queries to other search engines.

The actual implementation of this process is noted, and the researcher found out that doing this process sequentially – submitting query on one search engine and saving the HTML document as a local file, one at a time – is very slow. To improve this performance, the researcher has created a thread for each of search engine activity, making the whole process run in parallel.

Each search engine would usually provide the first 10 document hits to the query. We believe that in general, although not always true, the first 10 document hits are very good representative sample of the hits for that particular query. And since these first 10 documents are the ones with the highest scores as far as the particular search engine is concerned, the researcher has therefore decided to take the first 10 hit documents from a particular search engine. Hence, if the query is submitted to three search engines, we expect to get about 30 hit document results. If it is submitted to 5 search engines though, then we expect to get about 50 hit document URLs.

The hit document result of one search engine is not mutually exclusive to the result of the second or third search engine. That is to say that there may be hit document that could appear in more than one search engine. We intend not to come up with duplicate copies of the same URL. Hence, we examine each and every hit URL to make sure that it appear only once before inserting it onto our hit URL vector – an array-based list that grows and shrinks based on its content.

5.2.4 Each URL is searched online and copied on the local disk

The result in the previous activity presumes that we will have a maximum of 30 documents (for 3 search engines) to 50 documents (for 5 search engines) to work on. That is, these documents must be read online and each one be analyzed.

To do the above, each URL must be read online and once it is read via URL class in the Java implementation, it must be saved as a local document on the local disk. Reading and saving 30 to 50 documents will consume quite a lot of time, hence, it is very important that this whole process be done in parallel. Each thread that comes into completion will proceed with the next step, without unnecessarily waiting for other threads to complete.

It is also important to mention that it is always a possibility that a particular URL may exist as a result to a user's query by one or more search engines, yet in reality such URL no longer exists. Hence, we should not completely rely on the assumption that once we read a particular URL online we tend to get something tangible from it. The reasons for such failure could be that:

1. The URL no longer exists.
2. The computer or the server that contains the URL document is down when the request is made.
3. The traffic in the Internet is so heavy that it takes so much time downloading the said URL document.

Considering all these possible scenarios, the researcher implements each process of reading a URL online and saving its HTML document as local file as a thread. Each thread runs in parallel with other threads. And each thread runs on a time limit. Once the time limit is reached, the process is abandoned completely and the thread considered completed. On the possibility that the URL is no longer existing or the server holding the document is down, the thread would be considered erroneous and is stopped immediately.

Assuming that none of these things happen, then all threads will come into completion sooner or later. Once a thread completes, it is ready to proceed to the next process (step 5.2.5).

5.2.5 Local disk copy of the hit document is examined and its essential keywords and tags analyzed.

The result of the completed process above assumes that we will have around 30 to 50 local files to examine, these files themselves being HTML documents which are hit results to the user's query.

In this process, the following files were created and used in the analysis of HTML documents:

1. a file containing negative words, also called stop words.
2. a file containing negative symbols, also called stop symbols.
3. a file containing the key HTML tags
4. a file containing tags that are considered unimportant, also called stop tags.

The process of analyzing the HTML document begins by reading one line (usually of length 80 characters) of the document. This line of characters is analyzed by parsing it into tokens of words. Each word token is then subjected to a test of whether or not this word belongs to words in the negative dictionary. Recall that negative dictionary contains words that are so common that they are not representative word content of the document

– that omitting them from the document will still make the document content intact. If a word is part of the negative dictionary, then this word is struck down. If the word is not in the negative dictionary, then it is added to a string (initially empty) of distinct words.

A line of characters read from the document contains not only words but HTML tags as well. In the same time that words are analyzed if they are distinct or not (with reference to the negative dictionary), HTML tags are collected and analyzed as well. HTML tags are subjected to a test of whether they belong to the key tag file or non-important tag file. The HTML tag is abandoned right away if it appears in the negative tag file. If the tag happens to belong to tags in the essential tags file, then the tag, its operands and attributes are analyzed. Usually, the words that follow the tag are collected until the closing tag is encountered.

Consider, for example, the HTML tag <title>. The words following this tag are remembered until the closing tag </title> is encountered. The tags <title> and </title> are included in the essential tags file. The pair is needed in knowing the title of the document and is likewise used to categorize document through their title.

The tags pair of *<center>* and *</center>*, for example, are included in the *non-essential tags file*. When they appear in the document, we outright reject them for we are not interested, at this point, of whether a word or a group of words, and image or a plug-in appear in the center or not.

This process of reading a line of 80 characters one at a time is repeated until all of the content of the document is fully read.

The results of this process are:

1. A string which contains all the distinct keywords found in the document
2. Essential tag pairs are obtained. The tag pairs are usually used as delimiters (starting and ending point) to keywords in the document to analyze the category in which the document belongs.

5.2.6 The document's relevance score with respect to the user's query is calculated.

From the resulting string that contains all distinct keywords found in the document obtained from step 5.2.5 above, we then proceed to create a list of distinct keywords within this string. This is accomplished by parsing this string into words separated by whitespaces. We initially created an empty array-based list. We parse the string to get a keyword. Then the list is searched whether the keyword is already there or not. If it is not in the list, the keyword is inserted in the list, and its count is set to 1. If the keyword is already in the list, then the keyword is not inserted into the list, but its frequency count is incremented by 1. This process is repeated until all the tokens in the string are analyzed.

The result of the process mentioned above is a list containing all the keywords (sorted in ascending order, the case of the keyword – lowercase or uppercase or a combination of both – ignored). Another resulting data structure of the above process is an array of integer equivalent to the frequency count of the keywords in the document. The array index of the list of keywords and the frequency count must match for us to find which keyword is stored in a certain index and what is its frequency count.

In order to calculate the document's relevance in relation to the user's query, we have to find the keyword with the highest frequency count. Let this keyword's frequency be **TFi_max**.

We then determine the user's term(s) query, and the frequency result, based on the boolean operation involved, be called **TFi_j**. Consider the following cases of boolean operations involved:

Case 1: <u>No boolean operation involved</u>.

We simply determine the frequency count of the keyword involved.

Example: *if the user's query is "eagle", then we simply determine how many time "eagle" keyword appears in the document. This frequency count is TFi_j.*

Case 2: <u>OR operation</u>

We simply get the higher frequency count of the two or more keywords involved in the user's query.

Example: *Suppose that the user's query is "course notes" (double quotes not included) or "course + notes" or "course OR notes". We then find the term frequency of "course" and "notes". Assume that the term frequency of course = 55, and the term frequency of notes = 58. Then, we take the resulting **TFi_j** score of the combination as 55, since 55 is greater than 58.*

Case 3: <u>AND operation</u>

We simply get the smaller frequency count of the two or more keywords involved in the user's query.

Example: *Suppose that the user's query is "information & retrieval" (double quotes not included) or "information AND retrieval". We then find the term frequency of "information" and "retrieval". Assume that the term frequency of information = 55, and the term frequency of retrieval = 58. Then, we take the resulting **TFi_j** score of the combination as 55, since 55 is smaller than 58.*

Case 4: <u>Phrase</u>

We simply find the number of times the exact copy of the phrase appears in the document.

Example: *Suppose that the user enters "air pollution" as his query. In the parsing of document in the very beginning, we have to get the frequency count the phrase "air pollution" (double quote included when the user typed the query). If no exact copy appears in the document, the value of TFi_j = 0.*

We then compute the relevance score using the formula:

Relevance Score = 0.5 + 0.5 * (TFi_j / TFi_max)

Take note that if the document predominantly contains the user's query in frequency count, then TFi_j and TFi_max would be the same, and the document will obtain a score of 100%.

The relevance score based on the document's keywords-frequency score in the document counts 80% (the default value) of the total mark it gets. The other 20% (default value) is based on its popularity among all relevant documents. Together they account for 100% total score.

Two or more documents could probably have the same score, say all of them have 100% scores. In cases like this, a numeric value of TFi_j (which is in this case equal to TFi_max) are compared to determine which one gets a higher ranking than the other, even if both have the same relevance score.

Popularity is defined here as whether the document is popular among other documents of the same content. If other documents (at least one of them) has linked to document X, then document X is said to be popular, aside from the fact that it is also a relevant one. Then document X gets a 20% score for popularity. If, on the other hand, document Y gets no link from other documents of the same nature, then document Y is said to be not popular, and obtains a score of 0% in popularity. In effect, popularity score is either 1 or 0, either 20% or nothing.

It is important to note that in reality it is very hard for a document to be popular to some documents that also discuss the same subject. This of course will be possible if such document is really important and relevant that it is worthy to mention it, let alone have a link towards it, in one's HTML document. The popularity score here is an extra bonus given to a document for making it a worthy and a relevant document on the subject concerned.

Overall, the total score of a document is given by the relationship:

Total document score = relevance score + popularity score.

As mentioned earlier, the default values for relevance score and popularity score are 80% and 20%, respectively. In case that a user would like to modify this relative weight, a graphical user interface is provided for the user to modify it.

5.2.7 Documents are arranged accordingly based on their total score

If only to conclude the results of all these processes, documents must be arranged accordingly with respect to their overall score.

The listing of hits is arranged based on their overall document score, along with the title of the document, its document type, and its URL address. The document title is made to be a clickable link to the document itself. Clicking on the title will open up the document to the user.

5.2.8 Measuring Efficiency

There is sufficient evidence that concept presented in this chapter will provide Internet inquirers with more hits and be assured that these documents hits are relevant to their queries. It should be noted, however, that using this technique online would be a lot slower as compared to, say, inquiring on Hotbot about "*information retrieval*". The speed by which the hit results are presented to the user is faster in the latter, because all it does is to look up at its database and present the results right away. Contrast this to the techniques presented in this chapter where the query is sent to many search engines, results are collated, each document analyzed, and each document categorized before the result is presented to the inquirer.

With a big disparity in speed, it is best that the techniques presented in this chapter be done offline. Instead of having a web crawler taking a snapshot of every indexable HTML document in the world, the techniques presented here have to be implemented, its results kept in the database, and the process keeps repeating to the point that a database of information retrieval system is achieved, most probably similar in size to the database of a regular commercial search engine. It is through this that it can be said that the concept presented in this study achieves its aim of efficiency in full capacity.

Chapter 6

Conclusion

Things are evolving. Technologies are discovered and re-invented over time, and they keep on evolving because the environment and the society demand them to be. It was not long ago that Internet was created, yet over time we see the technology re-invented that we see now wireless communication, telephones being used like computers, and so on. Due to the higher demand for information dissemination and retrieval, the tools that we use to speed up information retrieval must also evolve. Internet search engines must evolve to keep up with the demand of the society for relevant information retrieval.

This thesis present some features that we believe will contribute to finding relevant information retrieval in the Internet, and proposing categorization of documents in order to minimize unnecessary browsing of unrelated documents. Our aim is twofold: to find real relevant hit documents to the user's query, and to improve efficiency in information retrieval by informing the user of the document's category through which the user could decide to browse or skip downloading the document based on its category.

We begin by proposing that search engines should be able to detect if user's query is ambiguous. This makes sense because many of Internet users are not native English-language speakers; they do write ambiguous queries unintentionally. By identifying and providing directions to ambiguous query, we have turned an ambiguous query into a clear, non-ambiguous query, and in doing so, increasing the likelihood of finding relevant documents based on the user's definition of the ambiguous term.

We also have presented our proposal for finding relevant documents based on focused queries. We detailed how an focused query is trimmed down into few essential keywords which are then sent as modified user query to search engines. The hits documents are individually analyzed to find if they are indeed relevant to the query. We detailed how focused queries in the form of *"what"*, *"who"*, *"when"*, *"where"*, *"why"* and *"how"* questions are addressed. This contribution is useful to all Internet users who send precise query and wanted to get the documents that contains the exact answer to the query. Students in general will benefit from this feature because they do ask exact question most of the time.

Unfocused queries are also addressed in this thesis. We then proposed that HTML documents be categorized. In this thesis, the category is based on academic relevance. By finding the necessary keywords in the title, or in the heading, or in the body, it is possible to categorized documents. We conclude that academic categorization is possible and feasible, and we recommend that future categorization can be made on any HTML document based on its commercial, or military, or entertainment content.

Over the course of preparation of this thesis, we sampled many queries on many search engines, and found out that many hit documents are of very little relevance to the query, and on the average about 13% of the hits are completely irrelevant. With this as a driving force, we then find a ranking algorithm that really measures the relevance of a document based on its content. We looked at many ranking algorithms compare each of them, and finally picked out TF (term frequency) as one component of the scoring scheme. We added popularity or cited scheme as another measure of relevance. The choice of term frequency measure will guarantee that all of the possible hit documents will all contain the user's query keywords. This is a guarantee that no hit document is completely irrelevant to the user's query. The popularity and cited document score is based on the notion that relevant document cited by another relevant document based on the user's query gets extra points or scores for relevance.

The concept of submitting the user's query to many search engines at the same time and collating and merging their hit documents is our way of finding more relevant documents and giving them back to the user. Our work guarantees that no duplicate documents are presented to the user as hit documents.

Our work does consider inputs from the user to modify default values in the implementation of some of the features of the thesis. For example, the user can use the default search engines provided in our work, or modify it by selecting his desired search engines from the given list. The user can also modify the weight of the popularity score in determining the relevance of a document. A default value of 20% is assigned to this scheme, but the user can minimize or remove this scoring altogether if he wishes to.

As mentioned earlier, we believe that technology must keep evolving to keep up with the pace of the demands of the society. That said, the restriction that hit documents are always in the form of text documents only should cease. It must accommodate other media as well, such as audio, video and still image. Information retrieval is possible in these forms of media. Speech recognition technology could be used to convert audio into text. The "*close captioning*" feature of a video could also be utilized to find information is such a medium. Eventually, audio and video content could be translated to text, and the resulting text could be explored, in the same way we explore HTML text documents to find information.

We do suggest that for still image, the filename of the image itself be used as a way to determine its content. That said, we do recommend that the filename of an image should exactly summarize what the image is all about. With that we mean that the filename will exact tell what the image is all about. This scheme works well in Unix system, but not in Windows because of the restriction in naming file in Windows. In order for this scheme to work under Windows, the naming of an image file in Unix should be implemented in Windows as well.

References

1. James Allan, and Hema Raghavan, "Using Part-of-speech Patterns to Reduce Query Ambiguity", ACM, 2002, pp.307-314

2. David B. Laeke and Ryan Scherle, "Towards Context-Based Search Engine Selection", ACM, 2001, pp. 109-112

3. Julian Kupiec, "MURAX: A Robust Linguistic Approach For Question Answering Using an On-line Encyclopedia", ACM, USA, 1993, pp. 181-190.

4. Ellen M. Voorhees, "Overview of the TREC 2001 Question Answering Track, National Institute of Standards and Technology, ACM , USA 2001, pp. 101-110

5. William B. Frakes, Ricardo Baeza-Yates, *Information Retrieval: Data Structures and Algorithms*, Englewood Cliffs, N.J. : Prentice Hall, 1992.

6. David Hawking, Nick Craswell, Peter Bailey, and Kathleen Griffihs, "*Measuring Search Engine Quality*", Information Retrieval Volume 4, pp 33 – 59, Kluwer Academic Publisher, The Netherlands, 2001.

7. Christian Borgelt, "A Naïve Bayes Classifier Plug-In for Data Engine", 2001, www://fuzzy.cs.uni-magdeburg.de/~borgelt/papers/nbpluin.pdf

8. R.O. Duda and P.E. Hart." Pattern Classification and Scene Analysis", J. Wiley & Sons, New York, NY, USA 1973

9. J. Good. "The Estimation of Probabilities: An Essay on Modern Bayesian Methods", MIT Press, MA, USA, 1965

10. A.R. Moghrabi, and R.A. Makholian , "A New Approach to Clustering Records in Information Retrieval Systems", Information Retrieval Volume 3, pp 105 – 126, Kluwer Academic Publisher, The Netherlands, 2000.

11. Wendy G. Lehnert, and Martin H. Ringle, "*Strategies for Natural Language Processing*", Hillsdale, N.J. : L. Erlbaum Associates, 1982

12. Frank Fallside, William A. Woods, "Computer Speech Processing", Englewood Cliffs, NJ : Prentice-Hall International, 1985

13. Mathew Joseph Palakal, "Computer Recognition of Human Speech" -- Thesis (M.Comp.Sc.), Concordia University, 1984

14. Stephen Robertson, "Comparing the Performance of Adaptive Filtering and Ranked Output Systems", Information Retrieval Volume 5, Kluwer Academic Publisher, The Netherlands, 2002.

15. Mary Dee Harris, "Introduction to Natural Language Processing", Reston, Va, USA, Reston Pub. Co.,1985

16. Alison Cooke, " A Guide to Finding Quality Information on the Internet: Selection and Evaluation Strategies", 2nd edition, London: Library Association, 2001.

17. Michael W. Berry and Murray Browne, "Understanding Search Engines: Mathematical Modeling and Text Retrieval", Philadelphia, PA: Society for Industrial and Applied Mathematics, c1999.

18. Candy Schwartz, "Sorting Out the Web: Approaches to Subject Access", Westport, Conn: Ablex Pub., 2001.

19. Wes Sonnenreich and Tim Macinta, "Guide to Search Engines", Wiley Computing Publishing, Boston, MA, USA, 1998.

20. Harvey M. Deitel, " Java: How to Program", 4th edition Upper Saddle River, N.J.: Prentice Hall, 2002.

Appendix A

Term Weighting System and Ranking Algorithm

The following term weight schemes are some of the selection from which we based our ranking algorithm:

1. **Term Frequency (TF)** – is based on the notion that constructs (words, phrases, word groups) that frequently occur in the test of documents have some bearing on the content of the text. The weight of term k in document i, $WEIGHT_{ik}$, might be set equal to the frequency of the occurrence of word construct k in document I

   $WEIGHT_{ik} = FREQ_{ik}$

   The term frequency system makes no distinction between terms that occur in every document of a collection, and those that occur in only a few items. Since, we are not going to keep large database, as what commercial search engines do, we basically have no collection of documents to consider.

2. **Inverse Document Frequency** – is based on the notion that the usefulness of a term for content representation increases with the frequency of the term in the document, but decreases with the number of documents (DOCFREQk) to which the term is assigned

   WEIGHTik = FREQik /DOCFREQk

   The inverse document frequency is useless when we consider cases of collection of documents. This weighting scheme is left aside in our ranking algorithm scheme as we have no collection of documents to consider.

3. **Term Descrimination Theory** – depends on the degree to which the assignment of a term to the documents of a collection is capable of decreasing the density of the document space (the average distance between documents). The discrimination value of term k (DISCVALUEk) is the difference between two measurements of documents space density, corresponding to the densities before and after assignment of term k. The weighting function for term k in document i is

   WEIGHTik = FREQik * DISCVALUEk

   This scheme is not in the nature of the hit documents we consider, so we dropped it off from consideration.

4. **Probabilistic Indexing Theory** – states that the best index terms are those that tend to occur in the relevant documents with respect to some query. When the terms are assigned to the documents independently of each other, a measure of term value is obtained from the term relevance, TERMRELk. This is the ratio of the proportion of the relevant items in which term k occurs to the proportion of non-relevant items in which the terms occur.

   WEIGHTik = FREQik – TERMRELk

   This scheme is not in the nature of the hit documents we consider, so we dropped it off from consideration.

Ranking Algorithm

We have formulated our ranking algorithm based on some information given below:

Definition of Terms:

Term = keyword

Document = resource

M = the number of query terms

Q_j = the jth query term, for $1 < j < M$

N = the number of documents in the database

D_i = the ith document in the database, for $1 < i < N$

$R_{i,q}$ = the relevance score of document D_i with respect to query q

$Li_{i,k}$ = the occurrence of an incoming hyperlink from document D_k to D_i

= 1 if such a hyperlink exist,

= 0 if otherwise

$Lo_{i,k}$ = the occurrence of an outgoing hyperlink from document D_i to document D_k

= 1 if such a hyperlink exists

= 0 if otherwise

C_{ij} = occurrence of query term Q_j in document D_i

= 1 if document D_i contains Q_j

= 0 if otherwise

TF x IDF (Term Frequency – Inverse Document Frequency) – based on the notion that constructs (words, phrases, word groups) that frequently occur in the test of documents have some bearing on the content of the text (TF) and that the usefulness of a term for content representation increases with the frequency of the term in document but decreases with the number of documents to which the term is assigned.

WEIGHT ik = FREQik/DOCFREQik

where

WEIGHTik = weight of term k in document i

DOCFREQk = number of documents containing term k

And the relevance score of document D_i with respect to the query q is

$$R_{i,q} = \Sigma_{\text{term } j \in q}(0.5 + 0.5\ TF_{i,j} / TF_{i,max})\ IDF_j$$

where $TF_{i,j}$ = term frequency of query term Q_j in document D_i

$TF_{i,max}$ = the maximum term frequency of a keyword in document D_i

$IDF_j = \log (N/\Sigma_{I=1}{}^{N} C_{ij})$

where C_{ij} = occurrence of query term Q_j in document D_i

= 1 if document D_i contains query term Q_j

= 0 if otherwise

N = number of documents in the database.

Hence, our ranking algorithm is

$$\mathbf{R_{i,q} = \Sigma_{term\ j \in q}(0.5 + 0.5\ TF_{i,j} / TF_{i,max}) * 0.8 + 0.2\ (Li_{i,k})}$$

Relevance score = 80% based on term frequency

+ 20% based on hyperink/popularity

The summation of every query term is obtained by the logical operation involved in the query.

Appendix B

Keywords and Methods Used in Categorizing Documents by Academic Relevance

1. **Course Notes**

1.1 if "*Course Note(s)*" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as course notes. If it is not found in the title, proceed to step1.2.

1.2 If "*Course Notes*" appear in at least one heading of the document (via HTML tag pairs <H1> and </H1>, up to <H6> and </H6>), the document is categorized as "*Course Notes*". If it is not found in the heading, proceed to step 1.3.

1.3 We treat "*course note*" like a user's query. Using our ranking algorithm we find the score of the query "*Course Notes*". If the document's score is 60% or more, the document is considered a course notes material. If it is not, proceed to step 1.4.

1.4 The frequency of each of the following keywords are added together and compared against the keyword that appears the most in the document. If the ratio of the frequency of all the keywords below against the frequency of the dominant keyword in the document is 60% or more, the document is considered a "*Course Notes*" material.

Here are the keywords that denote that the document is a "*Course Notes*" material:

course, notes, fall, winter, summer, term, semester, trimester, university, college, academy, institute, school, professor, instructor, teacher, subject, reference, textbook, chapter

Figure B.1 Database keywords that denote "Course Notes"

2. Frequently Asked Questions (FAQ)

2.1 if "Frequently Asked Questions" or "FAQ" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as FAQ. If it is not found in the title, proceed to step 2.2.

2.2 If "*Frequently Asked Questions*" or "*FAQ*" appear in at least one heading of the document (via HTML tag pairs <H1> and </H1>, up to <H6> and </H6>), the document is categorized as "FAQ". If it is not found in the heading, proceed to step 2.3.

2.3 We treat "*Frequently Asked Questions*" or "*FAQ*" like a user's query. Using our ranking algorithm we find the score of the query "*Frequently Asked Questions*" or "*FAQ*". If the document's score is 60% or more, the document is considered a course notes material. If it is not, proceed to step 2.4.

2.4 The frequency of each of the following keywords are added together and compared against the keyword that appears the most in the document. If the ratio of the frequency of all the keywords below against the frequency of the dominant keyword in the document is 60% or more, the document is considered a "*FAQ*" material.

Here are the keywords that denote that the document is a "*FAQ*" material:

Frequently, asked, questions, faq, FAQ, question, answer, Q:, A: q:, a:

Figure B.2 Database keywords that denote "Frequently Asked Questions"

3. Research Paper

3.1 if "*Research Paper*" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as a research paper. If it is not found in the title, proceed to step 3.2.

3.2 Check if the title of the document appears in an abstract (List of papers published in different journals). (*This is not implemented nor tested in this thesis*) If it does, then it is a research paper. If not, it is not a research paper.

4. Technical Report

4.1 if "*Technical Report*" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as a technical report.

5. Thesis

5.1 if "*Thesis*" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as a thesis. If it is not found in the title, proceed to step 5.2.

5.2 The frequency of each of the following keywords are added together and compared against the keyword that appears the most in the document. If the ratio of the frequency of all the keywords below against the frequency of the dominant keyword in the document is 60% or more, the document is considered a "*thesis*" material.

Here are the keywords that denote that the document is a "thesis" document:

thesis, degree, bachelor, master, doctor, university, college, academy, institute, school, department, faculty, dean, examining committee, chair, examiner, supervisor

Figure B.3 Database keywords that denote "Thesis"

6. Tutorial

6.1 if "*Tutorial*" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as Tutorial. If it is not found in the title, proceed to step 6.2.

6.2 If "*Tutorial*" appears in at least one heading of the document (via HTML tag pairs <H1> and </H1>, up to <H6> and </H6>), the document is categorized as "Tutorial" document.

7. Review

7.1 if "*Review*" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as Review. If it is not found in the title, proceed to step 7.2.

7.2 If "*Review*" appears in at least one heading of the document (via HTML tag pairs <H1> and </H1>, up to <H6> and </H6>), the document is categorized as "*Review*" document.

8. Research Paper/Technical Report

8.1 if "*Research Paper*" or "*Technical Report*" (case insensitive) appear in the title (via HTML tag pair <title> and </title>), then the document is considered as a research paper/technical report. If it is not found in the title, proceed to step 8.2.

8.2 The frequency of each of the following keywords are added together and compared against the keyword that appears the most in the document. If the ratio of the frequency of all the keywords below against the frequency of the dominant keyword in the document is 60% or more, the document is considered a "*Research Paper/Technical Report*" material.

Here are the keywords that denote that the document is a "*research paper /technical report*" document:

introduction, abstract, methodology, validation, summary, future work, references

Figure B.4 Database keywords that denote "Research Paper/Technical Report"

Appendix C

Stop Words, Stop Symbols, and HTML Tags

A. Database of Keywords Considered Stop Words or Non-Important Words:

The following words or terms are considered unessential in determining the content of document:

Part of Speech Category	Stop Words or Negative Words
1. Preposition	*about, above, across, after, against, along, among, around, at, before, behind, below, beneath, beside, between, beyond, but, by, despite, down, during, except, for, from, in, inside, into, like, near, of, off, on, onto, out, outside, over, past, since, through, throughout, till, to, toward, under, underneath, until, up, upon, with, within, without*
2. Pronoun	*I, you, she, he, it, we, you, they, me, you, her, him, it, us, them, mine, yours, hers, his, its, ours, theirs, this, that, these, those, who, whom, which, what, whoever, whomever, whichever, whatever, all, another, any, anybody, anyone, anything, each, everybody, everyone, everything, few, many, nobody, none, one, several, some, somebody, someone, myself, yourself, herself, himself, itself, ourselves, yourselves, themselves*
3. Conjunction	*and, but, or, nor, for, so, yet, after, although, as, because, before, how, if, once, since, than, that, though, till, until, when, where, whether, while*
4. Adverb	*also, consequently, finally, furthermore, hence, however, incidentally, indeed, instead, likewise, meanwhile, nevertheless, next, nonetheless, otherwise, still, then, therefore, thus, now, today, tomorrow*
5. Verb	*is, am, are, was, were, has, have, will, would, could, should, been, must*
6. Others	*a, an, the, hello, hi, goodbye, bye, sir*

Table C.1 Stop Words (also called Negative Dictionary)

B. Stop Symbols

The following symbols are considered unessential in determining the content of document:

#	@	!	,	;	&	<	>	=	
$	'		"	:	*	?	[	]	.

Table C.2 Stop Symbols

C. Essential HTML Tags

The following HTML tags are considered important and so are the terms attached to them:

<*title*> and </*title*>	to determine the title of the document
<*h1*> and </*h1*>	to determine the headings
<*h2*> and </*h2*>	
<*h3*> and </*h3*>	
<*h4*> and </*h4*>	
<*h5*> and </*h5*>	
<*h6*> and </*h6*>	
<*p*> and </*p*>	to determine the extent of the paragraph
<*a href=" "*> and </*a*>	to determine the hypertext link

Table C.3 Essential HTML Tags

Appendix D

Detailed Experiment Results on Ambiguous Queries

1. Ambiguous query word: **orange**

A. WordNet Senses/Definitions:

(a) orange and citrus
(b) citrus fruit
(c) citrous fruit
(d) chromatic color
(e) chromatic colour
(f) spectral color
(g) spectral colour
(h) citrus tree
(i) pigment
(j) river

B. Original Ambiguous Query:	orange
Excite, Hotbot and Google Results:	30 documents
Orange as fruit :	15/30
Orange as county:	7/30
Orange as browser/website:	4/30
Orange as electronic company:	2/30
Orange as bicycle store:	1/30
Dead link:	1/30

C. Modified Query:

1. Orange and fruit:

Precision :	30/30 = 100%

2. Orange and color:

Precision:	30/30 = 100%

2. Ambiguous query word: **bat**

A. WordNet Senses/Definitions:

(a) placental
(b) placental mammal
(c) eutherian
(d) eutherian mammal
(e) turn
(f) play
(g) racket
(h) racquet
(i) cricket equipment
(j) club

B. Original ambiguous query: bat

Excite, Hotbot and Google Results:	30 documents
Bat as an animal:	16/30
Bat as a baseball equipment:	10/30
Bat as a softball equipment:	2/30
Bat as an email system:	2/30
Bat as a tobacco group:	2/30
Bat as an Israeli women group:	1/30

C. Modified Query:

1. Bat and mammal:
   Precision: 30/30 = 100%
2. Bat and "cricket equipment"
   Precision: 30/30 = 100%

3. Ambiguous Query word: **washington**

A. WordNet Senses/Definitions:

(a) national capital
(b) American state
(c) federal government
(d) general
(e) full general
(f) President of the United States
(g) President
(h) Chief Executive
(i) Educator
(j) Pedagogue

B. Original ambiguous query: washington

Excite, Hotbot, and Google Results:	30 documents
Washington as American state:	12/30
Washington as a city in District of Columbia, USA:	11/30
Washington as actor:	1/30
Washington as magazine:	1/30
Washington as a mountain:	2/30
Washington as a university	2/30
Washington as a brewery:	1/30
Washington as a basketball league (NBA) member:	1/30

C. Modified Query:

1. Washington and "President of the United States"

Precision:	24/30 = 80%

2. Washington and "national capital"

Precision:	25/30 = 83%

4. Ambiguous Query word: **china**

A. WordNet Senses/Definitions:

(a) Asian country
(b) Asian nation
(c) Porcelain
(d) Island
(e) Crockery
(f) Dishware

B. Original ambiguous query:	china
Excite, Hotbot, and Google Results:	30 documents
China as an Asian country:	29/30
China as a dishware:	1/30

C. Modified Query:

1. China and dishware:

Precision:	27/30 = 90%

2. China and "asian nation"

Precision:	29/30 = 97%

5. Ambiguous Query word: **turkey**

A. WordNet Senses/Definitions:

(a) domestic fowl
(b) fowl
(c) poultry
(d) country
(e) state
(f) land
(g) unpleasant person
(h) disagreeable person
(i) flop
(j) bust
(k) gallinaceous bird
(l) gallinaceon

B Original ambiguous query:	turkey
Excite, Hotbot, and Google Results:	30 documents
Turkey as animal/poultry:	13/30
Turkey as a country:	16/30
Irrelevant hit:	1/30

C. Modified Query:

1. Turkey and poultry:	
Precision:	24/30
2. Turkey and country:	
Precision:	27/30

6. Ambiguous Query word:	**quebec**

A. WordNet Senses/Definitions:
(a) provincial capital
(b) Canadian province

B. Original ambiguous query:	quebec
Excite, Hotbot, and Google Results:	30 documents
Quebec as a Canadian province:	23/30
Quebec as a city:	6/30
Irrelevant hit:	1/30

C. Modified query:

1. Quebec and province:	
Precision:	30/30 = 100%
2. Quebec and city	
Precision:	26/30 = 87%

7. Ambiguous Query word:	**mercury**

A. WordNet Senses/Definitions:
(a) metallic element
(b) metal
(c) Roman deity
(d) inferior planet
(e) temperature

B. Original ambiguous query:	mercury
Excite, Hotbot, and Google Results:	30 documents
mercury as an insurance business:	1/30
mercury as a car dealer:	2/30
mercury as car financing agent:	1/30
mercury as a car:	3/30
mercury as a hotel:	1/30
mercury as mortgage network:	1/30
mercury as a boat engine:	3/30
mercury as an Australian newspaper:	3/30
mercury as Washington news bureau:	1/30
mercury as a software company:	2/30
mercury as Sacramento news bureau:	1/30
mercury as Kansas newspaper:	1/30

mercury as San Jose newspaper:	2/30
mercury as music group band:	1/30
mercury as women's basketball team:	2/30
mercury as US record label:	1/30
mercury as a programming language:	1/30
mercury as a planet:	1/30
mercury as a magazine:	1/30
irrelevant:	1/30

C. Modified query:

1. Mercury and planet
   Precision: 30/30 = 100%
2. Mercury and metal
   Precision: 29/30 = 97%

8. Ambiguous Query word: **mouse**

A. WordNet Senses/Definitions:

(a) rodent
(b) gnawer
(c) gnawing animal
(d) electronic device

B. Original query word:	mouse
Excite, Hotbot, and Google Results:	30 documents
Mouse as a computer peripheral:	11/30
Mouse as an animal:	11/30
Mouse as a music group:	2/30
Mouse as an internet service provider company:	1/30
Mouse as a Spanish newspaper:	1/30
Mouse as a computer learning school:	1/30
Irrelevant:	3/30

C. Modified query:

1. Mouse and rodent
   Precision: 27/30 = 90%
2. Mouse and computer
   Precision: 29/30 = 97%

9. Ambiguous Query word: **java**

A. WordNet Senses/Definitions:
- (a) island
- (b) beverage
- (c) drink
- (d) drinkable
- (e) potable
- (f) object-oriented programming language

B. Original query word:	java
Excite, Hotbot, and Google Results:	30 documents
Java as a computer programming tool/language:	30/30

C. Modified query	
1. Java and Indonesia	
Precision:	29/30 = 97%
2. Java and drink:	
Precision:	29/30 = 97%

10. Ambiguous Query word: **georgia**

A. WordNet Senses/Definitions:
- (a) American state
- (b) Colony
- (c) Asian country
- (d) Asian nation

B. Original query word:	georgia
Excite, Hotbot, and Google Results:	30 documents
Georgia as a US state:	24/30
Georgia as an Asian country:	2/30
Georgia as a search engine:	2/30
Georgia as a paper manufacturer:	2/30

C. Modified query:	
1. Georgia and USA	
Precision:	28/30 = 93%
2. Georgia and republic	
Precision:	27/30 = 90%

Appendix E

Detailed Experiment Results on Focused Queries

1. Focused Query on "Who?"

A. Original Queries:

1. Who is Mahatma Gandhi?
2. Who killed John F. Kennedy?
3. Who is the richest man on earth?
4. Who is the father of modern chemistry?
5. Who are the Acadians?

B. Corresponding Modified Queries:

1. Mahatma and Gandhi
2. Killed and John and F. and Kennedy
3. richest and man and earth
4. father and modern and chemistry
5. Acadians

C. Relevance of Hit Documents

1. Excite: 10/10, Hotbot: 10/10, Google: 10 /10
   Relevant Documents: 30/30 = 100%
   Irrelevant Documents: 0/30 = 0%

2. Excite: 10/10, Hotbot: 10/10, Google : 10/10
   Relevant Documents: 30/30 = 100%
   Irrelevant Documents: 0/30 = 0%

3. Excite: 4/10, Hotbot: 6/10, Google :8/10
   Relevant Documents: 18/30 = 60%
   Irrelevant Documents: 12/30 = 40%

4. Excite: 9/10, Hotbot: 7/10, Google : 7/10
   Relevant Documents: 23/30 = 77%
   Irrelevant Documents: 7/30 = 23%

5. Excite: 10/10, Hotbot: 9/10, Google : 10/10
   Relevant Documents: 29/30 = 97%
   Irrelevant Documents: 1/30 = 3%

2. Focused Query on "What?"

A. Original Queries:

1. What is the atomic number of oxygen?
2. What elements compose water?
3. What is the capital of Greenland?
4. What is an alchemist?
5. What science deals with the study of plants and animals?

B. Corresponding Modified Queries:

1. atomic and number and oxygen
2. elements and compose and water
3. capital and Greenland
4. alchemist
5. science and study and plants and animals

C. Relevance of Hit Documents

1. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 22/30 = 73%
   Irrelevant Documents: 8/30 = 27%

2. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 25/30 = 83%*
   Irrelevant Documents: 5/30 = 17%

   (If the question "what elements compose water?" is asked inside the classroom, the likely answer would be hydrogen and oxygen, in this case only 2 documents out of 25 are relevant.*

   If the question would cover mineral water and other commercially sold water, the answer would be the added elements in the water. In such case, the other 23 documents are relevant as well)

3. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 29/30 = 97%
   Irrelevant Documents: 1/30 = 3%

4. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 27/30 = 90%
   Irrelevant Documents: 3/30 = 10%

5. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 28/30 = 93% * (see next page)
   Irrelevant Documents: 2/30 = 7%

(The relevance measure here is deceiving in the sense that majority of the documents are considered relevance due to the presence of keywords in the same sentence or in the same paragraph, but only 7/28 contains biology or ecology which is the right answer if the question is asked inside the classroom)*

3. Focused Query on "Why?"

A. Original Queries:

1. Why is smoking bad to our health?
2. Why was JFK assassinated?
3. Why did George W. Bush reject the Kyoto Protocol?
4. Why is the ocean salty?
5. Why is the color of the sky blue?

B. Corresponding Modified Queries:

1. smoking and bad and health
2. JFK and assassinated
3. George and W. and Bush and rejected and Kyoto and Protocol
4. ocean and salty
5. color and sky and blue

C. Relevance of Hit Documents

1. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 26/30 = 87%
   Irrelevant Documents: 4/30 = 13%

   (Our front-end tool is able to obtain 26 out 30 documents from the hits of search engines, and after subjecting it to our test 26/26 are found to be all relevant)

2. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents from search engine: 25/30 = 83%*
   Relevant Documents from front-end: 3/25
   Irrelevant Documents: 5/30 = 17%

   (There are many Internet documents about President Kennedy's assassination but there are very little documents that answers the question why he was assassinated. Because of our extra keywords requirement, most of the returned documents 22/25 are rejected because of their failure to answer the question. After obtaining 3/25 selected documents, we read the full documents and found out that they did answer the question.*

   There are 2/25 documents though that answers the question but not in straightforward manner. Since the documents do not have the right keywords, they were rejected as irrelevant document, although we admit that their contents are really relevant)

3. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 29/30 = 97%
      Relevant document considered by front-end tool : 6/29
      Relevant document rejected by front-end tool: 12/29
   Irrelevant Documents: 1/30 = 3%

   (It is interesting to note that 12 out of 29 documents that explain why Pres. George W. Bush rejected the Kyoto Protocol was rejected by our front-end tool even if they are relevant. The reason is that they lack some extra keywords that we were looking for. Also, we note that most documents presented the reasons without saying the word "because", "reason", "in order to", etc. We attribute this failure of our front-end system to detect these relevant documents to the richness of natural language – English in this case – in presenting data/information without necessarily following a pattern.)

4. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 29/30 = 97%
      Relevant documents considered by front-end tool : 28/29
   Irrelevant Documents: 1/30 = 10%

5. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 27/30 = 90% *
      Relevant documents reported by front-end tool : 17/27
   Irrelevant Documents: 3/30 = 10%

4. Focused Query on "When?"

A. Original Queries:

1. When is McGill University founded?
2. When was JFK assassinated?
3. When was the Eiffel Tower constructed?
4. When was the Montreal Summer Olympic Games?
5. When did Apollo 11 landed on the moon?

B. Corresponding Modified Queries:

1. McGill and University and founded
2. JFK and assassinated
3. Eiffel and Tower and constructed
4. Montreal and Summer and Olympic and Games
5. Apollo and 11 and landed and moon

C. Relevance of Hit Documents

1. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 26/30 = 87%
      Relevant document according to front-end tool : 23/26
   Irrelevant Documents: 4/30

2. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents from search engine: 30/30
      Relevant Documents from front-end: 23/30
   Irrelevant Documents: 0/30

3. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 27/30 = 90%
      Relevant document considered by front-end tool : 23/27
      Relevant document rejected by front-end tool: 0/27
   Irrelevant Documents: 1/30 = 3%
   Dead link: 2/30 = 7%

4. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 28/30 = 93%
      Relevant documents considered by front-end tool : 28/28
   Irrelevant Documents: 2/30 = 7%

5. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 30/30 = 100%
      Relevant documents reported by front-end tool : 30/30
   Irrelevant Documents: 0/30

5. Focused Query on "Where?"

A. Original Queries:
1. Where is Taj Mahal located?
2. Where can I find the smallest fish in the world?
3. Where is the office of the President of Russia?
4. Where can I buy the cheapest helicopter in Canada?
5. Where is the largest diamond deposit in Africa?

B. Corresponding Modified Queries:
1. Taj and Mahal and located
2. find and smallest and fish and world
3. office and President and Russia
4. buy and cheapest and helicopter and Canada
5. largest and diamond and deposit and Africa

C. Relevance of Hit Documents
1. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 27/30 = 90%*
      Relevant document of front-end tool : 26/27
      Relavnt document missed by fron-end tool : 1/27
   Dead link : 3/30 = 10%

(There are 27 out of 30 documents are considered relevant, however, 3/27 are actually about Taj Mahal hotel, 1/27 is about Trump Taj Mahal, and 2/27 are about Taj Mahal Casino. However, following our front-end tool, the documents are actually stating where the Taj Mahak casino is located, and so with taj mahal hotel and restaurant)

2. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents from search engine: 26/30
      Relevant Documents found by front-end tool: 10/26
      Relevant document found by front-end tool that are irrelevant: 3/10
   Irrelevant Documents: 4/30

3. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 23/30 = 77%
      Relevant document relevant considered by front-end tool : 0/23
      Relevant document rejected by front-end tool: 0/23
      Irrelebvant document found by front-end tool: 23/23
   Irrelevant Documents: 7/30 = 23%

4. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 20/30 = 67%
      Relevant documents considered by front-end tool : 4/20
      Relevant document considered irrelevant by front-end tool: 1/20
   Irrelevant Documents: 1/030 = 33%

5. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 24/30 = 80% *
      Relevant documents reported by front-end tool : 18/24
      Relevant document considered irrelevant by front-end tool: 0/18
   Irrelevant Documents: 1/30 = 3%
   Dead link : 5/30 = 17%

6. Focused Query on "How?"

A. Original Queries:

1. How does the operating system work?
2. How does alcohol affect our judgment?
3. How does the ship float in water?
4. How to invest in stock market?
5. How to bake a croissant?

B. Corresponding Modified Queries:

1. operating and system and work
2. alcohol and affect and judgment
3. ship and float and water
4. invest and stock and market
5. bake and croissant

C. Relevance of Hit Documents

1. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 28/30 = 93%*
      Relevant document of front-end tool : 28/28
      Relavnt document missed by fron-end tool : 0/28
   Dead link : 2/30 = 10%

2. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents from search engine: 29/30
      Relevant Documents found by front-end tool: 29/29
      Relevant document found by front-end tool that are irrelevant: 0/29
   Dead link : 1/30

3. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 30/30 = 77%
      Relevant document relevant considered by front-end tool : 30/30
      Relevant document rejected by front-end tool: 0/30
      Irrelevant document found by front-end tool: 0/30
   Irrelevant Documents: 0/30 = 0%

4. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 20/30 = 67%
      Relevant documents considered by front-end tool : 4/20
      Relevant document considered irrelevant by front-end tool: 1/20
   Irrelevant Documents: 1/030 = 33%

5. Excite, Hotbot, and Google Results: 30 documents
   Relevant Documents: 28/30 = 93% *
      Relevant documents reported by front-end tool : 28/28
      Relevant document considered irrelevant by front-end tool: 0/28
   Irrelevant Documents: 2/30 = 7%*

   (* *Documents are considered irrelevant because they are written in languages other than English, and words like bake does not exist, although croissant is a general international word which means a kind of bread.)*

Appendix F

Algorithms

Appendix F.1

Algorithm for determining an ambiguous query

```
//Prompt for user to enter a query, and the user query is stored as a string
//*********************************************************
display "Enter a query: "
read (query)

//Count the number of distinct keywords in the query string.
//If distinct keyword count is more than one, there is no need to check for its ambiguity
//*******************************************************************
tokenCount ← number of distinct keyword;
if (tokenCount > 1 )
        exit
else
    begin
        queryString ← Result of (WorldNet query (query));

        //if there are 2 or more definitions for the keyword, the keyword is ambiguous
        //***************************************************************
        if ( "Sense 2" is a substring of QueryString)

           begin
                count ← number of distinct senses in queryString
                Parse(QueryString)
                index ← 0
                array_of_string  strArray[20] ← null string
                n ← 1;
                while (n is less than or equal to count)
                begin
                        token = n^th sense of queryString
                        n ← n +1
                        strArray[index] ← token
                        index ← index + 1
                end while

           end if

           // display a message saying that the query is ambiguous and the keywords
           // which the user may wish to add to the query
           //**********************************************************
           display "Your query is ambiguous. It has more than one meaning:"
           display "Please choose one (or more) of the following keywords that you wish"
           display "to be added to your query"
           n ← 0
```

```
          while (n is less than or equal to index)
          begin
               display strArray[n]
               n ← n + 1
          end while
end if
```

Appendix F.2

Algorithm for Obtaining Hit Documents from Search Engines Based on User's Query

```
Begin Algorithm
//display a prompt for user's query
//***************************
display "Please enter a query: "
read (query)

//Parse the query to determine keywords and Boolean operation present in the query
//*******************************************************************
string query ← the query entered by the user

string operation ← NULL

//determine the Boolean operation (if any) in the query
//******************************************
if " (double quote) is a substring of query
  operation ← "Phrase"

integer keywordCount ← number of keywords in the query
if (keywordCount is equal to 1)
  operation  ← "none"

if ("AND" is a substring of query) OR
   ("&" is a substring of query)
   operation ← "AND"

if ("OR" is a substring of query) OR
  ("+" is a substring of query)
   operation ← "OR"

if (keywordCount > 1) AND ("&" is not a substring of query) AND
   ("+" is not a substring of query)
  operation ← "OR"


//delete stop words in the query
//***********************
array_of_string StopWords[] ← the stop words dictionary as entered by the programmer
(see Appendix C for the list of stop words)

integer min ← 0
integer max ← size of array StopWords

//perform binary search to determine if every keyword in the query is a stop word
//*****************************************************************
string tempQuery ← query
integer n ← 1

while (n is less than or equal to keywordCount)
begin
    string keyword ← nth keyword in tempQuery

    //this is the binary search part to check if keyword is a stop word
    //if a keyword is a stop word, it is deleted from the query
    //**************************************************
    integer mid ← (min + max) /2
    while (min is less than or equal to max)
    begin
        if (keyword  is same as StopWords[mid])
         begin
            query ← remove keyword from  query
            break
         end
      else
         if keyword is greater than StopWords[mid]
             max ← mid -1
         else
             min ← mid + 1
         end if
      end if

      mid ← (min + max) / 2
   end while

  n ← n + 1

end while

//after deleting stop words, the number of distinct keywords in the query is reduced
//all query keywords are stored in an array of string called queryVector
//*****************************************************************
keywordCount ← number of keywords in query
array_of_string queryVector[keywordCount] ← NULL
integer n ← 1
integer indx ← 0

while (n is less than or equal to keywordCount)
```

```
 begin
    queryVector[indx] ← $n^{th}$ keyword in query
    indx ← indx + 1
   n ← n + 1
 end while


//create URL of the search engines on the user query
//determine the complete URL to be passed to the search engine
//only 3 search engines (default) are presented, but it can be modified to include
//as many search engines as possible, with the algorithm almost the same
//*******************************************************************
//Case#1: Excite Search Engine
//*************************


string exciteSearchURL ← NULL
string temp1 ← "http://search.excite.com/search.gw?search="
string temp2 ← NULL
string tempQuery ← NULL

//the URL address is dependent on the keywords on the query and the operation involved
//*********************************************************************
if (operation is "NONE")
begin
   temp2 ← "&trace=1&src=nsl&sorig=netscape"
   else
      if (operation is "OR")
      begin
         temp2 ← "&tsug=-1&csug=10&sorig=netscape"
         tempQuery ← queryVector[0] + "%2B" + queryVector[1]
      end
      else
        if (operation is "AND")
        begin
           temp2 ← "&sorig=netscape"
           tempQuery ← queryVector[0] + "%26" + queryVector[1]
        end
        else
           if (operation is "PHRASE")
           begin
              temp1 ← temp1 + "%22"
              temp2 ← "%22&sorig=netscape"
              tempQuery ← queryVector[0] + "+" + queryvector[1]
           end
end if
```

```
exciteSearchURL ← temp1 + tempQuery + temp2



//determine the complete URL to be passed to the search engine
//************************************************
//Case #2: Hotbot Search Engine
//*************************

string hotbotSearchURL ← NULL
string temp1 ← "http://www.hotbot.com/?"
string temp2 ← NULL
string tempQuery ← NULL

//the URL address is dependent on the keywords on the query and the operation involved
//*******************************************************************
if (operation is "NONE")
begin
   temp1 ← temp1 + "comefrom=nspanel-search&MT="
   else
      if (operation is "OR")
      begin
         temp1 ← temp1 + "MT="
         temp2 ← "&_v=2&OPs=MDRTP";
         tempQuery = queryVector[0] + "%2B" + queryVector[1]
      else
        if (operation is "AND")
        begin
           temp1 ← temp1 + "MT="
           temp2 ← "&_v=2&OPs=MDRTP"
           tempQuery ← queryVector[0] + "%26" +queryVector[1]
        end
        else
           if (operation is "PHRASE")
           begin
              temp1 = temp1 + "=%22"
              temp2 = "%22&_v=2&OPs=MDRTP"
              tempQuery = queryVector[0] + "+" + queryVector[1]
           end
end if

hotbotSearchURL ← temp1 + tempQuery + temp2;
```

```
//determine the complete URL to be passed to the search engine
//***********************************************
//Case #3: Lycos Search Engine
//**************************

string hotbotSearchURL ← NULL
string temp1 ← "http://www.lycos.com/cgi-bin/pursuit?"
string temp2 ← NULL
string tempQuery ← NULL

//the URL address is dependent on the keywords on the query and the operation involved
//*********************************************************************
if (operation is "NONE")
begin
    temp1 ← temp1 + "query="
    temp2 ← "&cat=dir&maxhits=10"
    else
        if (operation is "OR")
        begin
            temp1  ← temp1 + "cat=dir&maxhits=10&query="
            temp2 ← "&npl="
            tempQuery ← queryVector[0] + "%2B" + queryVector[1]
        else
           if (operation is "AND")
           begin
              temp1 ← temp1 + "cat=dir&maxhits=10&query"
              temp2 ← "+&npl="
              tempQuery ← queryVector[0] + "%26" + queryVector[1]

           end
           else
               if (operation is "PHRASE")
               begin
                   temp1 ← temp1 + "cat=dir&maxhits=10&query=%22"
                   temp2 ← "%22&npl="
                   tempQuery ← queryVector[0] + "+" + queryvector[1]
               end
end if

lycosSearchURL ← temp1 + tempQuery + temp2;




array_of_string engineAddress[] = {exciteSearchURL, hotbotSearchURL,
                                        lycosSearchURL}
 integer size ← size of array engineAddress
 array_of_string searchEngineDoc[size] ← NULL


 //open each of the URL and save the HTML document on the disk
 //****************************************************
 integer indx ← 0
 while (n is less than size)
 begin
    //the following are basically the way to open a URL in Java
    //and read the HTML text file of the document online
    //************************************************
    URL testURL ← engineAddress[indx]
    BufferedReader in ← InputStreamReader(testURL.openStream())
    string  text ← NULL
    string line ← NULL
     line ← in.readLine()

    keep reading one line at a time until end of file
    //************************************
     while (line is not NULL)
     begin
        text  ← text + line
         line ← in.readLine()
     end while

     searchEngineDoc[indx] ← text

    indx ← indx + 1
end while


//out of the string stored in searchEngineDoc[] (one each for every search engine)
//each has to be parsed in order to obtain the hit results of each search engine
//*****************************************************************

//Case  #1: Excite Search Engine
//the Excite result is usually (if not always) 10 hit documents,
//each of which begins with "http://search.excite.com"
//************************************************

string text ← searchEngineDoc[0]
integer textLength ← number of words in string text
array_of_string exciteHitResults[10] ← NULL
integer indx ← 0
```

```
 integer n ← 1
 while (n is less than or equal to textLength)
 begin
   string testString ← n^th word in string text
   bool valid ← false
   valid ← ((“href” is a substring of testString) AND
            (“ http://search.excite.com” is a  substring of testString))

   if (valid is true)
   begin
      integer x ← index of “href” in testString
      integer y ← index of “http://search.excite.com” in testString

      //the URL is always enclosed in a pair of double quotes
      //*********************************************
      integer start ← index of “(double quote) in testString starting at position x
      integer last ← index of “ (double quote) in testString starting at position y

     string hitdocument ← substring of testString from position (start+1) up to
                          position (last-1)
      exciteHitResults[indx] ← hitdocument
   end

   n ← n + 1
 end



//Case  #2: Hotbot Search Engine
//the Hotbot result is usually (if not always) 10 hit documents,
//each of which begins with "http://www.hotbot.com"
//***********************************************

string text ← searchEngineDoc[1]
integer textLength ← number of words in string text
array_of_string hotbotHitResults[10] ← NULL
integer indx ← 0
integer n ← 1
while (n is less than or equal to textLength)
begin
  string testString ← n^th word in string text
  bool valid ← false
  valid ← ((“href” is a substring of testString) AND
           (“ http://www.hotbot.com” is a substring of testString)
           AND (“/director.asp” is NOT a substring of testString))
  if (valid is true)
  begin
    integer x ← index of "href" in testString
    integer y ← index of "http://www.hotbot.com" in testString

    //the URL is always enclosed in a pair of double quotes
    //*******************************************
    integer start ← index of "(double quote) in testString starting at position x
    integer last ← index of " (double quote) in testString starting at position y

   string hitdocument ← substring of testString from position (start+1) up to
                         position (last-1)
    hitResults[indx] ← hitdocument
  end

  n ← n + 1
end


End Algorithm
```

Appendix F.3

Algorithm for Parsing an HTML Document to obtain document's title, headings and referenced website

```
Begin Algorithm
//This algorithm opens an HTML document online and parse the resulting text document
//to obtain the document's title, headings and anchors or referenced document
//************************************************************************
//declare and initialize some variables
//*****************************
string filename ← "test2.html"                //or any HTML document for that matter
BufferedReader buff ←
        new BufferedReader(new InputStreamReader (new FileInputStream(filename))

//keep reading one line of text from the file
//********************************
string text ← NULL

string line ← buff.readline()
while (line is not NULL)
begin
    text ← text + line
    line ← buff.readline()
end while

//get the title of the document by finding the keywords
//delimited by <TITLE> and </TITLE>
//**************************************
integer positionStartTitle ← index of "<TITLE>" in the string text
integer positionEndTitle ← index of "</TITLE>" in the sting text

string testString ← substring of text from positionStartTitle to positionEndTitle
integer n ← position of ">" in testString
string Title ← substring of testString from position n+1

text ← delete substring "<TITLE>" from text
text ← delete substring "</TITLE>" from text
text ← delete substring Title from text

Perform Categorization of Document based on its Title
(see algorithm for Categorization of Document based on Title)
```

```
//get the first heading of the document by finding the keywords
//delimited by <H1> and </H1>.
//************************************************
integer positionStartHeading ← index of "<H1>" in the string text
integer positionEndHeading ← index of "</H1>" in the sting text

string testString ← substring of text from positionStartHeading to positionEndHeading
integer n ← position of ">" in testString
string Heading1 ← substring of testString from position n+1

text ← delete substring "<H1>" from text
text ← delete substring "</H1>" from text
text ← delete substring Heading1 from text

Perform Categorization of Document based on its Headings
(see algorithm for Categorization of Document based on Headings)

//get the headings of the document by finding the keywords
//delimited by <H2> and </H2>
//**********************************************
integer positionStartHeading ← index of "<H2>" in the string text
integer positionEndHeading ← index of "</H2>" in the sting text

string testString ← substring of text from positionStartHeading to positionEndHeading
integer n ← position of ">" in testString
string Heading2 ← substring of testString from position n+1

text ← delete substring "<H2>" from text
text ← delete substring "</H2>" from text
text ← delete substring Heading2 from text

Perform Categorization of Document based on its Headings
(see algorithm for Categorization of Document based on Headings)

//get the headings of the document by finding the keywords
//delimited by <H3> and </H3>
//**********************************************
integer positionStartHeading ← index of "<H3>" in the string text
integer positionEndHeading ← index of "</H3>" in the sting text

string testString ← substring of text from positionStartHeading to positionEndHeading
integer n ← position of ">" in testString
string Heading3 ← substring of testString from position n+1
```

```
text ← delete substring "<H3>" from text
text ← delete substring "</H3>" from text
text ← delete substring Heading3 from text

Perform Categorization of Document based on its Headings
(see algorithm for Categorization of Document based on Headings)

//get the headings of the document by finding the keywords
//delimited by <H4> and </H4>
//***********************************************
integer positionStartHeading ← index of "<H4>" in the string text
integer positionEndHeading ← index of "</H4>" in the sting text

string testString ← substring of text from positionStartHeading to positionEndHeading
integer n ← position of ">" in testString
string Heading4 ← substring of testString from position n+1

text ← delete substring "<H4>" from text
text ← delete substring "</H4>" from text
text ← delete substring Heading4 from text

Perform Categorization of Document based on its Headings
(see algorithm for Categorization of Document based on Headings)

//get the headings of the document by finding the keywords
//delimited by <H5> and </H5>
//***********************************************
integer positionStartHeading ← index of "<H5>" in the string text
integer positionEndHeading ← index of "</H5>" in the sting text

string testString ← substring of text from positionStartHeading to positionEndHeading
integer n ← position of ">" in testString
string Heading5 ← substring of testString from position n+1

text ← delete substring "<H5>" from text
text ← delete substring "</H5>" from text
text ← delete substring Heading2 from text

Perform Categorization of Document based on its Headings
(see algorithm for Categorization of Document based on Headings)
```

```
//get the headings of the document by finding the keywords
//delimited by <H6> and </H6>
//**********************************************
integer positionStartHeading ← index of "<H6>" in the string text
integer positionEndHeading ← index of "</H6>" in the sting text

string testString ← substring of text from positionStartHeading to positionEndHeading
integer n ← position of ">" in testString
string Heading6 ← substring of testString from position n+1

text ← delete substring "<H6>" from text
text ← delete substring "</H6>" from text
text ← delete substring Heading6 from text

Perform Categorization of Document based on its Headings
(see algorithm for Categorization of Document based on Headings)

//get the anchor of the document to determine if it is referencing other related documents
//in the hit list for the purpose of adding popularity score to the referenced document. It is
//delimited by <a> and </a>, but its URL address is inside <a> tag, in parameter "href".
//*****************************************************************
array_of_string URLAdresses[50] ← NULL
integer indx ← 0
integer positionStartAnchor ← index of "<a>" in the string text

while (positionStartAnchor is not -1)
begin
        integer n ← position of ">" in testString
        string testString ← substring of text from positionStartAnchor to n-1

        string addressString ← 1st substring in testString
        if (href is a substring of addressString)
        begin
                integer m ← index of 1st " (double quote) in addressString
                integer p ← index of 2nd " (double quote) in addressString
                string URLaddress ←substring of addressString
                                        from index (m+1) to (p-1)
                URLAddresses[indx] ← URLaddress
                indx ← indx + 1
        end
    text ← delete substring "<a>" from text
    text ← delete substring "</a>" from text

    integer positionStartAnchor ← index of "<a>" in the string text
end while
End Algorithm
```

Appendix F.4

Algorithm for Categorization of Document Based on its Title

```
Begin Algorithm
 string Title ← title of document obtained from performing parsing of HTML document
 string Category ← "Unclassified"

 //determine if it is a "Course Notes" document
 //****************************************
 bool test ← false

 //determine if "Course Notes" or its derivative exists in the title of the document
 //**************************************************************
 if ("Course Notes" is a substring of Title) OR
   ("Course Note" is a substring of Title) OR
   ("Lecture Notes" is a substring of Title)
 begin
        //determine if preposition "ON" does exist in the title
        //without it, the document is automatically a "Course Notes" document
        //*****************************************************
        integer n ← index of ("ON") in Title
        if (n is less than zero)
          test ← true
        else
                begin
                  //determine if "ON" appears after "Course Notes" or its derivatives
                  //**************************************************
                  integer x ← index of "Course Notes" in the Title
                  integer y ← index of "Course Note" in the Title
                  integer z ← index of "Lecture Notes" in the Title
                        if (x less than n) OR (y is less than n) OR (z is less than n)
                          test ← true
                end else
        end if
end if

if (test is equal to true)
 Category ← "Course Notes"
end if
```

```
//determine if it is an FAQ (Frequently Asked Questions) document
//***********************************************************
bool test ← false

//determine if "F.A.Q" or its derivative exists in the title of the document
//***********************************************************
if ("FAQ" is a substring of Title) OR
  ("Frequently Asked Questions" is a substring of Title) OR
  ("F.A.Q." is a substring of Title)
begin
      //determine if preposition "ON" does exist in the title
      //without it, the document is automatically an "FAQ" document
      //***********************************************
      integer n ← index of ("ON") in Title
      if (n is less than zero)
        test ← true
      else
              begin
                //determine if "ON" appears after "FAQ" or its derivatives
                //*********************************************
                integer x ← index of "FAQ" in the Title
                integer y ← index of "Frequently Asked Questions" in the Title
                integer z ← index of "F.A.Q." in the Title
                       if (x less than n) OR (y is less than n) OR (z is less than n)
                         test ← true
              end else
      end if
end if

if (test is equal to true)
 if (Category is "Unclassified")
   Category ← "FAQ (Frequently Asked Questions)"
 else
    Category ← Category  + ", FAQ (Frequently Asked Questions)"
 end if
end if
```

```
 //determine if it is a "Research Paper" document
//******************************************
bool test ← false

//determine if "Research Paper" or its derivative exists in the title of the document
//****************************************************************
if ("Research Paper" is a substring of Title) OR
  ("Research Papers" is a substring of Title) OR
begin
       //determine if preposition "ON" does exist in the title
       //without it, the document is automatically a "Research Paper" document
       //*******************************************************
       integer n ← index of ("ON") in Title
       if (n is less than zero)
         test ← true
       else
               begin
                 //determine if "ON" appears after "Research Paper" or its derivatives
                 //***************************************************
                 integer x ← index of "Research Paper" in the Title
                 integer y ← index of "Research Papers" in the Title
                 if (x less than n) OR (y is less than n)
                         test ← true
               end else
       end if
end if

if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Research Paper"
  else
     Category ← Category  + ", Research Paper"
  end if
end if


//determine if it is a "Technical Report" document
//******************************************
bool test ← false

//determine if "Technical Report" or its derivative exists in the title of the document
//****************************************************************
if ("Technical Report" is a substring of Title) OR
  ("Technical Reports" is a substring of Title) OR
  ("Tech. Rep." is a substring of Title)
begin
```

```
        //determine if preposition "ON" does exist in the title
        //without it, the document is automatically a "Technical Report" document
        //**********************************************************
        integer n ← index of ("ON") in Title
        if (n is less than zero)
          test ← true
        else
                begin
                 //determine if "ON" appears after "Technical Report" or its derivatives
                 //**********************************************************
                 integer x ← index of "Technical Report" in the Title
                 integer y ← index of "Technical Report" in the Title
                 integer z ← index of "Tech. Rep." In the Title
                 if (x less than n) OR (y is less than n) OR (z is less than n)
                         test ← true
                end else
        end if
end if

if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Technical Report"
  else
    Category ← Category  + ", Technical Report"
  end if
end if
```

//determine if it is a "Tutorial" document

```
//*************************************
bool test ← false

//determine if "Tutorial" or its derivative exists in the title of the document
//**********************************************************
if ("Tutorial" is a substring of Title) OR
  ("Tutorials" is a substring of Title) OR
begin
       //determine if preposition "ON" does exist in the title
       //without it, the document is automatically a "Tutorial" document
       //****************************************************
       integer n ← index of ("ON") in Title
       if (n is less than zero)
         test ← true
       else
```

```
                begin
                  //determine if "ON" appears after "Tutorial" or its derivatives
                  //**********************************************
                  integer x ← index of "Tutorial" in the Title
                  integer y ← index of "Tutorials" in the Title
                  if (x less than n) OR (y is less than n)
                          test ← true
                end else
        end if
end if

if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Tutorial"
  else
    Category ← Category  + ", Tutorial"
  end if
end if


//determine if it is a "Thesis" document
//**********************************
bool test ← false

//determine if "Thesis" or its derivative exists in the title of the document
//*******************************************************
if ("Thesis" is a substring of Title)
begin
        //determine if preposition "ON" does exist in the title
        //without it, the document is automatically a "Thesis" document
        //************************************************
        integer n ← index of ("ON") in Title
        if (n is less than zero)
          test ← true
        else
                begin
                 //determine if "ON" appears after "Thesis" or its derivatives
                 //**********************************************
                 integer x ← index of "Thesis" in the Title
                 if (x less than n)
                    test ← true
                end else
        end if
end if
```

```
if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Thesis"
  else
    Category ← Category + ", Thesis"
  end if
end if
```

//determine if it is a "Review" document

```
//*************************************
bool test ← false

//determine if "Review" or its derivative exists in the title of the document
//*************************************************************
if ("Review" is a substring of Title) OR
  ("Revue" is a substring of Title) OR
begin
        //determine if preposition "ON" does exist in the title
        //without it, the document is automatically a "Review" document
        //****************************************************
        integer n ← index of ("ON") in Title
        if (n is less than zero)
          test ← true
        else
                begin
                  //determine if "ON" appears after "Review" or its derivatives
                  //************************************************
                  integer x ← index of "Review" in the Title
                  integer y ← index of "Revue" in the Title
                  if (x less than n) OR (y is less than n)
                          test ← true
                end else
        end if
end if

if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Review"
  else
    Category ← Category + ", Review"
  end if
end if
```

```
//determine if it is a "Research Paper/Technical Report" document
 //*********************************************************
 bool test ← false

 //determine if "Technical Report" or its derivative exists in the title of the document
 //*****************************************************************
 if ("Technical Report" is a substring of Title) OR
   ("Technical Reports" is a substring of Title) OR
   ("Tech. Rep." is a substring of Title) OR
   ("Research Paper" is a substring of Title)
 begin
      //determine if preposition "ON" does exist in the title, without it, the document is
      //automatically a "Research Paper/Technical Report" document
      //*****************************************************************
      integer n ← index of ("ON") in Title
      if (n is less than zero)
        test ← true
      else
              begin
                //determine if "ON" appears after
                //"Research Paper/Technical Report" or its derivatives
                //*******************************************
                integer w ← index of "Technical Report" in the Title
                integer x ← index of "Technical Report" in the Title
                integer y ← index of "Tech. Rep." in the Title
                integer z ← index of "Research Paper" in the Title
                if  ((w is less than n) OR (x less than n) OR
                    (y is less than n) OR (z is less than n))
                      test ← true
              end else
      end if
 end if

 if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Research Paper/Technical Report"
  else
    Category ← Category  + ", Research Paper/Technical Report"
  end if
 end if


End Algorithm
```

Appendix F.5

Algorithm for Categorization of Document Based on its Headings

```
//This algorithm is performed only after categorization of document based on its title
//failed to classify the document
//**********************************************************************
Begin Algorithm

 string Category ← category of document after performing Categorization of Document
                    based on its Title
 string Heading1 ← first heading of document obtained from performing
                    parsing of HTML document
 string Heading2 ← second heading of document obtained from performing
                    parsing of HTML document
 string Heading3 ← third heading of document obtained from performing
                    parsing of HTML document
 string Heading4 ← fourth heading of document obtained from performing
                    parsing of HTML document
 string Heading5 ← fifth heading of document obtained from performing
                    parsing of HTML document
 string Heading6 ← sixth heading of document obtained from performing
                    parsing of HTML document
 array_of_string Headings[6] = {Heading1, Heading2, Heading3,
                               Heading4, Heading5, Heading6}

//do all these only if document is still "Unclassified"
//*****************************************
if (Category is not "Unclassified")
  exit
else
 //determine if it is a "Course Notes" document
 //*****************************************
 bool test ← false
 integer indx ← 0
 //determine if "Course Notes" or its derivative exists in the heading of the document
 //**********************************************************************
 while (indx is less than 6)
 begin
  if ("Course Notes" is a substring of Headings[ind]) OR
    ("Course Note" is a substring of Headings[indx]) OR
    ("Lecture Notes" is a substring of Headings[indx])
   begin
     //determine if preposition "ON" does exist in the heading
     //without it, the document is automatically a "Course Notes" document
     //*******************************************************
```

```
        integer n ← index of ("ON") in Headings[indx]
        if (n is less than zero)
          begin
              test ← true
              break
           end
        else
                begin
                  //determine if "ON" appears after "Course Notes" or its derivatives
                  //****************************************************
                  integer x ← index of "Course Notes" in Headings[indx]
                  integer y ← index of "Course Note" in Headings[indx]
                  integer z ← index of "Lecture Notes" in Headings[indx]
                   if (x less than n) OR (y is less than n) OR (z is less than n)
                     begin
                           test ← true
                           break
                     end
                end else
        end if
   end if

   indx ← indx + 1
end while

if (test is equal to true)
  Category ← "Course Notes"
end if
```

//determine if it is an "FAQ (Frequently Asked Questions)" document

```
//************************************************************
bool test ← false
integer indx  ← 0
//determine if "FAQ" or its derivative exists in the heading of the document
//************************************************************
while (indx is less than 6)
begin
  if ("FAQ" is a substring of Headings[ind]) OR
    ("Frequently Asked Questions" is a substring of Headings[indx]) OR
    ("F.A.Q." is a substring of Headings[indx])
  begin
       //determine if preposition "ON" does exist in the heading
       //without it, the document is automatically an "FAQ" document
       //************************************************
```

```
          integer n ← index of ("ON") in Headings[indx]
          if (n is less than zero)
            begin
                test ← true
                break
             end
          else
                  begin
                   //determine if "ON" appears after "FAQ" or its derivatives
                   //*********************************************
                   integer x ← index of "FAQ" in Headings[indx]
                   integer y ← index of "Frequently Asked Questions" in Headings[indx]
                   integer z ← index of "F.A.Q." in Headings[indx]
                    if (x less than n) OR (y is less than n) OR (z is less than n)
                      begin
                            test ← true
                            break
                      end
                  end else
          end if
     end if

     indx ← indx + 1
  end while

 if (test is equal to true)
   if (Category is "Unclassified")
     Category ← "FAQ (Frequently Asked Questions)"
   else
      Category ← Category  + ", FAQ (Frequently Asked Questions)"
   end if
 end if
```

//determine if it is a "Research Paper" document

```
 //*****************************************
 bool test ← false
 integer indx  ← 0
 //determine if "Research Paper" or its derivative exists in the heading of the document
 //*****************************************************************
 while (indx is less than 6)
 begin
  if ("Research Peper" is a substring of Headings[ind]) OR
    ("Research Papers" is a substring of Headings[indx])
```

```
        begin
          //determine if preposition "ON" does exist in the heading
          //without it, the document is automatically a "Research Paper" document
          //**********************************************************
          integer n ← index of ("ON") in Headings[indx]
          if (n is less than zero)
            begin
               test ← true
               break
            end
          else
                 begin
                   //determine if "ON" appears after "Research Paper" or its derivatives
                   //******************************************************
                   integer x ← index of "Research Paper" in Headings[indx]
                   integer y ← index of "Research Papers" in Headings[indx]
                    if (x less than n) OR (y is less than n)
                      begin
                           test ← true
                           break
                      end
                 end else
          end if
      end if

      indx ← indx + 1
    end while

   if (test is equal to true)
     if (Category is "Unclassified")
       Category ← "Research Paper"
     else
       Category ← Category  + ", Research Paper"
     end if
   end if
```

//determine if it is a "Technical Report" document

```
  //******************************************
  bool test ← false
  integer indx  ← 0
  //determine if "Technical Report" or its derivative exists in the heading of the document
  //*************************************************************
  while (indx is less than 6)
  begin
```

```
    if ("Technical Report" is a substring of Headings[ind]) OR
      ("Technical Reports" is a substring of Headings[indx]) OR
      ("Tech. Rep." is a substring of Headings[indx])
    begin
         //determine if preposition "ON" does exist in the heading
         //without it, the document is automatically a "Technical Report" document
         //**********************************************************
         integer n ← index of ("ON") in Headings[indx]
         if (n is less than zero)
           begin
               test ← true
               break
            end
         else
                 begin
                   //determine if "ON" appears after "Technical Report" or its derivatives
                   //*****************************************************
                   integer x ← index of "Technical Report" in Headings[indx]
                   integer y ← index of "Technical Reports" in Headings[indx]
                   integer z ← index of "Tech. Rep." in Headings[indx]
                    if (x less than n) OR (y is less than n) OR (z is less than n)
                      begin
                            test ← true
                            break
                      end
                 end else
         end if
    end if

    indx ← indx + 1
  end while

 if (test is equal to true)
   if (Category is "Unclassified")
     Category ← "Technical Report"
   else
      Category ← Category  + ", Technical Report"
   end if
 end if
```

```
//determine if it is a "Tutorial" document
 //***********************************
 bool test ← false
 integer indx  ← 0
 //determine if "Tutorial" or its derivative exists in the heading of the document
 //************************************************************
 while (indx is less than 6)
 begin
   if ("Tutorial" is a substring of Headings[ind]) OR
     ("Tutorials" is a substring of Headings[indx])
     begin
        //determine if preposition "ON" does exist in the heading
        //without it, the document is automatically a "Tutorial" document
        //*************************************************
        integer n ← index of ("ON") in Headings[indx]
        if (n is less than zero)
          begin
              test ← true
              break
           end
        else
                begin
                  //determine if "ON" appears after "Tutorial" or its derivatives
                  //*********************************************
                  integer x ← index of "Tutorial" in Headings[indx]
                  integer y ← index of "Tutorials" in Headings[indx]
                if (x less than n) OR (y is less than n)
                    begin
                          test ← true
                          break
                    end
                end else
        end if
   end if

   indx ← indx + 1
 end while

if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Tutorial"
  else
    Category ← Category  + ", Tutorial"
  end if
 end if
```

```
//determine if it is a "Thesis" document
//************************************
bool test ← false
integer indx  ← 0
//determine if "Thesis" or its derivative exists in the heading of the document
//**********************************************************
while (indx is less than 6)
begin
 if ("Thesis" is a substring of Headings[ind])
   begin
      //determine if preposition "ON" does exist in the heading
      //without it, the document is automatically a "Thesis" document
      //*****************************************************
      integer n ← index of ("ON") in Headings[indx]
      if (n is less than zero)
        begin
            test ← true
            break
        end
      else
              begin
                //determine if "ON" appears after "Thesis" or its derivatives
                //*********************************************
                integer x ← index of "Thesis" in Headings[indx]
                if (x less than n)
                   begin
                          test ← true
                          break
                   end
              end else
      end if
   end if

   indx ← indx + 1
end while

if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Thesis"
  else
    Category ← Category  + ", Thesis"
  end if
end if
```

```
//determine if it is a "Review" document
 //**********************************
 bool test ← false
 integer indx  ← 0
 //determine if "Review" or its derivative exists in the heading of the document
 //**************************************************************
 while (indx is less than 6)
 begin
   if ("Review" is a substring of Headings[ind]) OR
      ("Revue" is a substring of Headings[indx])
      begin
          //determine if preposition "ON" does exist in the heading
          //without it, the document is automatically a "Review" document
          //****************************************************
          integer n ← index of ("ON") in Headings[indx]
          if (n is less than zero)
            begin
                test ← true
                break
            end
          else
                begin
                  //determine if "ON" appears after "Review" or its derivatives
                  //**********************************************
                  integer x ← index of "Review" in Headings[indx]
                  integer y ← index of "Revue" in Headings[indx]
                if (x less than n) OR (y is less than n)
                    begin
                          test ← true
                          break
                    end
                end else
          end if
    end if

    indx ← indx + 1
 end while

if (test is equal to true)
  if (Category is "Unclassified")
    Category ← "Review"
  else
     Category ← Category  + ", Review"
  end if
 end if
```

//determine if it is a "Research Paper/Technical Report" document

```
//**********************************************************
bool test ← false
integer indx  ← 0
//determine if "Research Paper/Technical Report"
//or its derivative exists in the heading of the document
//***************************************
while (indx is less than 6)
begin
  if ("Technical Report" is a substring of Headings[ind]) OR
    ("Technical Reports" is a substring of Headings[indx]) OR
    ("Tech. Rep." is a substring of Headings[ind]) OR
    ("Research Paper" is a substring of Headings[indx])

    begin
      //determine if preposition "ON" does exist in the heading without it, the document
      //is automatically a "Research Paper/Technical Report" document
      //****************************************************************
      integer n ← index of ("ON") in Headings[indx]
      if (n is less than zero)
        begin
            test ← true
            break
        end
      else
              begin
                //determine if "ON" appears after
                //"Research Paper/Technical Report" or its derivatives
                //******************************************
                integer w ← index of "Technical Report" in Headings[indx]
                integer x ← index of "Technical Reports" in Headings[indx]
                integer y ← index of "Tech. Rep." In Headings[indx]
                integer z ← index of "Research Paper" in Headings[indx]
              if (w is less than n) OR (x is less than n) OR
                (y is less than n) OR (z is less than n)
                  begin
                        test ← true
                        break
                  end
              end else
      end if
  end if

  indx ← indx + 1
end while
```

```
 if (test is equal to true)
   if (Category is "Unclassified")
     Category ← "Research Paper/Technical Report"
   else
     Category ← Category + ", Research Paper/Technical Report"
   end if
 end if

end if

End Algorithm
```

Appendix F.6

Algorithm for Categorization of Document Based on its Contents

```
//This algorithm is performed only after categorization of document based on its headings
//failed to classify the document
//*********************************************************************
Begin Algorithm

  string Category ← category of document after performing Categorization of Document
                    based on its Headings

array_of_string CourseNotesKeywords[] = { "course", "notes", "fall", "winter",
          "summer", "term", "semester", "university", "college", "academy", "institute",
          "school", "professor", "instructor", "teacher", "subject", "reference",
         "textbook", "chapter"}
array_of_string FAQKeywords[] = { "frequently", "asked", "questions", "question",
          "answer", "answers", "FAQ", "F.A.Q.", "Q:", "A:", "q:", "a:"}

array_of_string ThesisKeywords[] = { "thesis", "degree", "bachelor", "doctor", "master",
          "PhD", "Ph.D.", "university", "college", "academy", "institute", "school",
         "faculty", "department", "dean", "examining", "committee", "chair", "examiner",
         "supervisor"}

array_of_string ResearchPaper_TechReportKeywords[] = { "introduction", "abstract",
"methodology", "validation", "summary", "future", "works", "references"}

  array_of_string Headings[6] = {Heading1, Heading2, Heading3,
                                 Heading4, Heading5, Heading6}

array_of_string textKeywords[] ← array containing keywords in the document text
                    obtained from performing the algorithm to calculate document
                    relevance score

array_of_string textKeywordCount[] ← array containing keywords frequency count in
                    the document text obtained from performing the algorithm to
                    calculate document relevance score

integer TFimax ← the frequency of the dominant keyword in string text, obtained from
                    performing the algorithm to calculate document relevance score
 //do all these only if document is still "Unclassified"
 //****************************************
 if (Category is not "Unclassified")
   exit
 else
```

```
//determine if it is a "Course Notes" document
//****************************************
bool test ← false
integer keywordSize  ← size of array CourseNotesKeywords[]
integer indx ← 0
integer score ← 0

//determine if the frequency score of each keyword in array CourseNotesKeywords[]
//*****************************************************************
while (indx is less than keywordSize)
begin
  integer min ← 0
  integer max ← size of array textKeywords[]

 //this is the binary search part to check if a "Course Notes" keyword is
 //in the array textKeywords[], if so then determine its frequency score
 //******************************************************
  boolean found  ← false
  integer mid ← (min + max) /2
  while (min is less than or equal to max)
  begin
      if (CourseNotesKeywords[indx]  is same as textKeywords[mid])
       begin
            found ← true
            integer position ← mid
            break
       end
    else
       if (CourseNotesKeywords[indx] is greater than textKeywords[mid]
            max ← mid -1
       else
            min ← mid + 1
       end if
    end if

   mid ← (min + max) / 2
  end while

  //determine the frequency of the keyword if it is found
  //*****************************************
  if (found  is true)
    score ←  score + textKeywordCount[position]
  end if

 indx ← indx + 1
end while
```

```
float relevanceScore ← score/TFimax

if (relevanceScore is greater than or equal to 0.6)
  Category ← "Course Notes"
end if
```

//determine if it is a "FAQ (frequently Asked Questions)" document

```
  //*********************************************************
  bool test ← false
  integer keywordSize  ← size of array FAQKeywords[]
  integer indx ← 0
  integer score ← 0

  //determine if the frequency score of each keyword in array FAQKeywords[]
  //************************************************************
  while (indx is less than keywordSize)
  begin

     integer min ← 0
     integer max ← size of array textKeywords[]

    //this is the binary search part to check if a "FAQ" keyword is
    //in the array textKeywords[], if so then determine its frequency score
    //*****************************************************
     boolean found  ← false
     integer mid ← (min + max) /2
     while (min is less than or equal to max)
     begin
          if (FAQKeywords[indx]  is same as textKeywords[mid])
           begin
                found ← true
                integer position ← mid
                break
          end
       else
           if (FAQKeywords[indx] is greater than textKeywords[mid]
                max ← mid -1
           else
                min ← mid + 1
           end if
       end if

      mid ← (min + max) / 2
    end while
```

```
        //determine the frequency of the keyword if it is found
        //*******************************************
        if (found  is true)
          score ← score + textKeywordCount[position]
        end if

       indx ← indx + 1

end while

float relevanceScore ← score/TFimax

if (relevanceScore is greater than or equal to 0.6)
  if (Category is "Unclassified")
    Category ← "FAQ"
  else
     Category ← Category + ", FAQ"
end if
```

//determine if it is a "Thesis" document

```
//************************************
   bool test ← false
   integer keywordSize  ← size of array ThesisKeywords[]
   integer indx ← 0
   integer score ← 0

   //determine if the frequency score of each keyword in array ThesisKeywords[]
   //************************************************************
   while (indx is less than keywordSize)
   begin
      integer min ← 0
      integer max ← size of array textKeywords[]

     //this is the binary search part to check if a "Thesis" keyword is
     //in the array textKeywords[], if so then determine its frequency score
     //******************************************************
      boolean found  ← false
      integer mid ← (min + max) /2
      while (min is less than or equal to max)
      begin
            if (ThesisKeywords[indx]  is same as textKeywords[mid])
             begin
                   found ← true
                   integer position ← mid
                   break
             end
```

```
          else
              if (ThesisKeywords[indx] is greater than textKeywords[mid]
                    max ← mid - 1
              else
                    min ← mid + 1
              end if
          end if

        mid ← (min + max) / 2
      end while

      //determine the frequency of the keyword if it is found
      //*******************************************
      if (found  is true)
        score ←  score + textKeywordCount[position]
      end if

     indx ← indx + 1

end while

float relevanceScore ← score/TFimax

if (relevanceScore is greater than or equal to 0.6)
  if (Category is "Unclassified")
    Category ← "Thesis"
  else
     Category ← Category + ", Thesis"
end if
```

//determine if it is a "Research Paper/Technical Report" document

```
   //*******************************************************
   bool test ← false
   integer keywordSize  ← size of array ResearchPaper_TechReportKeywords[]
   integer indx ← 0
   integer score ← 0

   //determine the frequency score of each keyword in array
   // ResearchPaper_TechReportKeywords[]
   //********************************************
   while (indx is less than keywordSize)
   begin

      integer min ← 0
      integer max ← size of array textKeywords[]
```

```
    //this is the binary search part to check if a "Research Paper/Technical Report"
    //keyword is in the array textKeywords[], if so then determine its frequency score
    //************************************************************
     boolean found ← false
     integer mid ← (min + max) /2
     while (min is less than or equal to max)
     begin
        if (ResearchPaper_TechReportKeywords[indx] is same as textKeywords[mid])
         begin
             found ← true
             integer position ← mid
             break
         end
       else
         if (ResearchPaper_TechReportKeywords[indx] is greater than
             textKeywords[mid])
              max ← mid -1
         else
              min ← mid + 1
         end if
       end if

      mid ← (min + max) / 2
     end while

    //determine the frequency of the keyword if it is found
    //*****************************************
    if (found is true)
      score ← score + textKeywordCount[position]
    end if

    indx ← indx + 1

end while

float relevanceScore ← score/TFimax

if (relevanceScore is greater than or equal to 0.6)
  if (Category is "Unclassified")
    Category ← "Research Paper/Technical Report"
  else
    Category ← Category + ", Research Paper/Technical Report"
end if

End Algorithm
```

Appendix F.7

Algorithm to Calculate Document Relevance Score

```
//This algorithm calculates the document's relevance score based on the user's query
//*******************************************************************

Begin Algorithm
//declare and initialize some variables
//*******************************
string text ← the resulting string from doing algorithm for parsing HTML document to
              obtain document's title, headings and referenced website
string query ← the query entered by the user

string operation ← NULL

//determine the Boolean operation (if any) in the query
//*******************************************
if " (double quote) is a substring of query
  operation ← "Phrase"

integer keywordCount ← number of keywords in the query
if (keywordCount is equal to 1)
  operation ← "none"

if ("AND" is a substring of query) OR
   ("&" is a substring of query)
   operation ← "AND"

if ("OR" is a substring of query) OR
  ("+" is a substring of query)
   operation ← "OR"

if (keywordCount > 1) AND ("&" is not a substring of query) AND
   ("+" is not a substring of query)
  operation ← "OR"


//delete stop words in the query
//************************
array_of_string StopWords[] ← the stop words dictionary as entered by the programmer
(see Appendix C for the list of stop words)

integer min ← 0
integer max ← size of array StopWords
```

```
//perform binary search to determine if every keyword in the query is a stop word
//*****************************************************************
string tempQuery ← query
integer n ← 1

while (n is less than or equal to keywordCount)
begin
      string keyword ← nth keyword in tempQuery

      //this is the binary search part to check if keyword is a stop word
      //if a keyword is a stop word, it is deleted from the query
      //***************************************************
      integer mid ← (min + max) /2
      while (min is less than or equal to max)
      begin
           if (keyword  is same as StopWords[mid])
            begin
                  query ← remove keyword from  query
                  break
            end
         else
             if keyword is greater than StopWords[mid]
                   max ← mid -1
             else
                   min ← mid + 1
             end if
         end if

        mid ← (min + max) / 2
    end while

  n ← n + 1

end while

//after deleting stop words, the number of distinct keywords in the query is reduced
//*****************************************************************
keywordCount ← number of keywords in query


//delete unnecessary tags in text
//************************
string token ← token in text, each token is separated by whitespace
```

```
while (token is not NULL)
begin
      if ("<" is a substring of token)
      begin
          integer m ← index of "<" in text
          integer n ← index of ">" in text

         text ← remove substring (from m to n) in text
      end if
     token ← next token in string text
end while


//Special Case: Query is a Phrase
//**************************
//Note: the stop words in the text are untouched because a query that is a phrase may
//contain some stop words. If stop words are deleted from the text, a phrase that originally
//occurred may not exit anymore
// *********************************************************************

//initially the frequency count is zero
//*****************************
integer TFij ← 0
if (operation is "Phrase")
begin
   integer queryLength ← length (number of characters) of query
   integer textLength ← length  (number of characters) of string text
   integer start ← 0
   boolean  present ← false

   //determine if the query (a phrase) does exist  (a substring) in string text
   //keep counting its frequency until no more occurrence is encountered
   //******************************************************
   present ← query is a substring from index (start to textLength) of string text
   while (present is true)
   begin
      TFij ← TFij + 1;
      integer position ← index of query in string text
     start ← position + queryLength
     present ← query is a substring from index (start to textLength) of string text
   end while

end if
```

```
//remove all stop words in the document text
//***********************************
string tempText ← text
 wordsCount ← length (number of words) in string  text
integer n ← 1
integer min ← 0
integer max ← size of array StopWords

while (n is less than or equal to wordsCount)
begin
        //perform binary search to determine if every keyword in the query is a stop word
        //****************************************************************

        string keyword ← nth keyword in tempText

        //this is the binary search part to check if keyword is a stop word
        //if a keyword is a stop word, it is deleted from the query
         //************************************************
         integer mid ← (min + max) /2
        while (min is less than or equal to max)
        begin
            if (keyword  is same as StopWords[mid])
            begin
                text ← delete keyword from text
                break
            end
           else
              if keyword is greater than StopWords[mid]
                      max ← mid -1
             else
                      min ← mid + 1
              end if
           end if
            mid ← (min + max) / 2
        end while

        n ← n + 1
end while

//after removing the stop words in string text, the string text is leaner
//all the keywords frequency in text will be counted
//**************************************************
integer wordsCount ← number of words in string text
array_of_string textKeywords[wordsCount] ← NULL
array_of_integer textKeywordCount[wordsCount] ← all zero's
```

```
integer indx ← 0
integer n ← 1

while (n is less than or equal to wordsCount)
begin
    string keyword ← nth keyword in string text
    integer min ← 0
    integer max ← size of array textKeywords

   //find if the keyword is already in the array
   //if it is not there, put it and its count is 1
   //if it is already there, do nothing, just increment its count by 1
   //**************************************************

   //this is the binary search part to check if keyword is already in the array
    //********************************************************
     boolean found  ← false
     integer mid ← (min + max) /2
     while (min is less than or equal to max)
     begin
          if (keyword  is same as textKeywords[mid])
           begin
                 found ← true
                 integer position ← mid
                 break
           end
       else
           if keyword is greater than textKeyowrds[mid]
                 max ← mid -1
           else
                 min ← mid + 1
           end if
       end if

      mid ← (min + max) / 2
    end while

    //increment frequency if keyword is already in the array
    //*******************************************
    if (found  is true)
      textKeywordCount[position] ← textKeywordCount[position] + 1
    else
       begin
```

```
            //keyword not in the array, create a new entry, its count is 1
            //**********************************************
            textKeywords[indx] ← keyword
            textKeywordCount[indx] ← 1
            indx ← indx + 1
          end
        end if

    n ← n + 1
end while


//find the dominant keyword, the keyword that has the highest frequency
//the highest frequency is stored in TFimax
//*******************************************************
integer indx ← 0
integer TFimax ← 0
integer position ← 0
while (textKeywordCount[indx] is greater than zero)
begin
    if (textKeywordCount[indx] is greater than TFimax)
    begin
        TFimax ← textKeywordCount[indx]
        position ← indx
    end

    indx ← indx + 1
end


//get all keywords in the query, put them in array, each keyword frequency is initially 0
//******************************************************************
integer keywordCount ← number of keywords in string query (with no stop words)
array_of_string queryKeywords[keywordCount] ← NULL
array_of_integer queryKeywordsFrequency[keywordCount] ← all zero's

integer n ← 1
integer indx ← 0
while (n is less than or equal to keywordCount)
begin
    string keyword ← nth word in string query
    queryKeyword[indx] ← keyword
    queryKeywordFrequency[indx] ← 0
    indx ← indx +1
    n ← n + 1
end while
```

```
//determine the frequency of every keyword in the query in the string text
//this is done by finding first if the keyword is in the list of distinct keywords in text
//if the keyword is found, its location is enough to determine its frequency
//**********************************************************************
integer keywordCount ← number of keywords in string query
integer n ← 1
integer indx ← 0
while (n is less than or equal to keywordCount)
begin
    integer min ← 0
    integer max ← size of array textKeywords[]

   //this is the binary search part to check if keyword is already in the array
   //************************************************************
    boolean found ← false
    integer mid ← (min + max) /2
    while (min is less than or equal to max)
    begin
        if (queryKeyword[indx]  is same as textKeywords[mid])
         begin
             found ← true
             integer position ← mid
             break
         end
      else
         if keyword is greater than textKeywords[mid]
             max ← mid -1
         else
             min ← mid + 1
         end if
      end if

     mid ← (min + max) / 2
    end while

    //determine the frequency of the keyword if it is found
    //*******************************************
    if (found  is true)
      queryKeywordFrequency[indx] ← textKeywordCount[position]
    end if

   indx ← indx + 1

  n ← n + 1
end while
```

```
//calculate the relevance score by determining TFij relative to TFimax
//the score of TFij is also dependent on the Boolean operation involve in the query
//*****************************************************************
integer keywordCount ← number of keywords in string query
integer n ← 1
integer indx ← 0
integer minScore ← queryKeywordFrequency[indx]
integer maxScore ← queryKeywordFrequency[indx]

while (n is less than or equal to keywordCount)
begin

   //find the keyword that has the highest and lowest frequency scores
   //****************************************************
   if (queryKeywordFrequency[indx] is greater than maxScore)
     maxScore ← queryKeywordFrequency[indx]
   end if

  if (queryKeywordFrequency[indx] is less than minScore)
     minScore ← queryKeywordFrequency[indx]
  end if
  indx ← indx +1
  n ← n + 1
end while


if (operation is "AND")
  TFij ← minScore
else
   if (operation is "OR")
      TFij ← maxScore
   else
       TFij ← queryKeywordFrequency[0]
   end if
end if

//calculate relevance score based on term frequency
//****************************************
float relevanceScore  ← 0.5 + 0.5 * TFij/TFimax

End Algorithm
```