TinySearch: An "Intelligent" Search Engine Implementation in Perl

http://students.cse.unt.edu/~dta0022/searchengine/

David Adamo Jr.
Department of Computer Science and Engineering
3940 N. Elm Street
Denton, TX 76207

DavidAdamo@my.unt.edu

ABSTRACT
This paper is a project report for CSCE 5200: Information Retrieval and Web Search. The paper describes the practical steps taken toward the implementation of a simple web search engine that crawls, indexes and retrieves information from the University of North Texas' web domain. The search engine implementation described in this report includes a breadth first web crawler/indexer, a text preprocessor, query engine and a web-based graphical user interface. This search engine implementation also infuses some intelligence capabilities into the search process. This is achieved by carrying out query expansion with synonyms and emphasizing words in a query that are deemed important. This report also gives an evaluation of the search engine's performance.

Categories and Subject Descriptors
H.4 [**Information Retrieval, Web Search**]: search, term-weighting, intelligent search; D.2.8 **[Software Engineering]:** Information Retrieval- *measures*

General Terms
Web Search

Keywords
Information Retrieval, web search, term-weighting

1. INTRODUCTION
Retrieving relevant information from large collections of data poses an interesting challenge. It turns out that attempting to retrieve relevant and useful information from large collections of data is something we do more often than we realize. An indicator of this is the fact that the Internet has rapidly become an important and extremely useful tool for gathering information. This is hardly surprising especially considering that the Internet is probably the largest and most diverse source of data and information on the planet. Due to the diverse and constantly changing nature of the Internet, it is necessary to have systems for accurately and efficiently retrieving information that is relevant to specific needs. Search engines are one of such systems. Consequently, the design and implementation of efficient search engines continues to be an active area of research in both academia and industry.

Web search engine implementations typically include a text preprocessor (for eliminating markup tags and identifying useful words and entities in a collection of words), a web crawler, indexer and query engine which takes advantage of some sort of document/query representation model and term-weighting scheme to represent the indexed documents. A popular document representation model is the *vector space model*. This is the document/query representation model employed in the search engine implementation described in this report.

The goal of the project described in this report, is to demonstrate the fundamental concepts and techniques typically used in the design and implementation of search engines. This is achieved by designing and implementing a search engine that searches and retrieves relevant documents from a collection of 4500 web pages culled from the University of North Texas (UNT) web domain. Consequently, the concepts introduced over the duration of CSCE 5200: Information Retrieval and Web Search are brought together in an attempt to build an efficient and functional web search engine.

1.1 Project Scope
The scope of the project was to implement a simple web search engine (using the Perl programming language). The search engine was required to provide a web-based Graphical User Interface (GUI) that enables users to provide queries for which they require relevant information. Based on a user's query, the search engine was required to present relevant web pages from within the UNT web domain. The following specific requirements for the search engine also had to be met:

- The starting point of the crawler would be `http://www.unt.edu`
- Multiple URLs are crawled using a breadth-first strategy and their contents are indexed.
- No duplicate web pages are indexed and all web pages must be from within the UNT web domain.
- At least 3000 web pages from the UNT domain have to be indexed.

In an attempt to improve the efficiency and accuracy of the search engine, an intelligent component was added to the implementation. The intelligent component attempts to improve the user-supplied query by adding words with similar meanings (synonyms) to the query and performing a search based on the reformulated query. The intelligent component also attempts to identify terms in the query that are important and then place further emphasis on those terms.

This report provides an evaluation of the search engine's performance as well as a brief exposition of some of the challenges faced and observations made during its design and implementation.

The rest of this report adheres to the following structure. In section 2, the core elements of the search engine are described. The proposed intelligent component is described in section 3. An evaluation of the search engine's performance in response to a number of test queries, and an outline of observations made, are presented in section 4. In section 5, some related works are presented. Finally, section 6 concludes the report and hints at some possible areas that could benefit from future work.

2. THE WEB SEARCH ENGINE

This section describes the various components of the search engine. These components include:

- Web Crawler
- Text Preprocessor/Indexer
- Query Engine
- Web-based Graphical User Interface

Over the course of CSCE 5200: Information Retrieval and Web Search, these different components were separately built and tested. This project represents a combination of these separate components to form a fully functional search engine. To facilitate the task of combining these different components, special attention was paid to code reusability and modularization. As a result, quite a number of subroutines/modules were used in the development of the search engine. The general architecture of the web search engine is shown in Figure 1 [6].

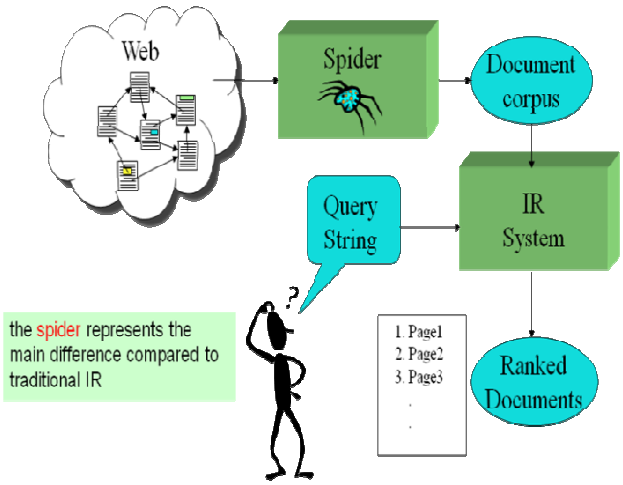

Figure 1: Web Search Engine Architecture

The search engine receives queries from users and uses two pre-constructed indexes to retrieve web pages that are considered relevant to the user-provided queries. The indexes were constructed using 4500 web pages gathered from the UNT web domain. The indexes are pre-constructed and stored persistently on disk before the search engine is ready to receive queries from users. The various components of the search engine are discussed in further details in the following subsections.

2.1 Web Crawler

The web spider starts out by gathering 6000 URLs from within the UNT web domain using a breadth-first strategy. The crawler was configured to start crawling from `http://www.unt.edu`. Page URLs were gathered using Perl's *HTML::LinkExtor* module. During the implementation of the web crawler, special attention was paid to preventing the crawler from gathering duplicate URLs, images, PDF documents, and other common non-text based file types. After gathering 6000 URLs, each of them was tested for validity by ensuring that the pages they link to were currently active. This was done to avoid crawling dead links. For each active link, the web page represented by that link was then pre-processed and indexed. Out of 6000 URLs gathered, 4500 were found to be valid and currently active. Hence, the final number of web pages indexed turned out to be 4500.

2.2 Text Preprocessor, Indexer and Vector Space Model

The text preprocessor is primarily responsible for stripping web pages of html tags, tokenization, removing stop words/punctuations and word stemming. Stemming is done to ensure that different words that have the same root form become identical to the search engine.

A request is made to each active link, and the web page contents returned are then preprocessed. While preprocessing the web pages, two indexes are built; a word-level index and a document level index. The word-level index is stored in a two-dimensional Perl hash and has the structure shown in Figure 2 [4].

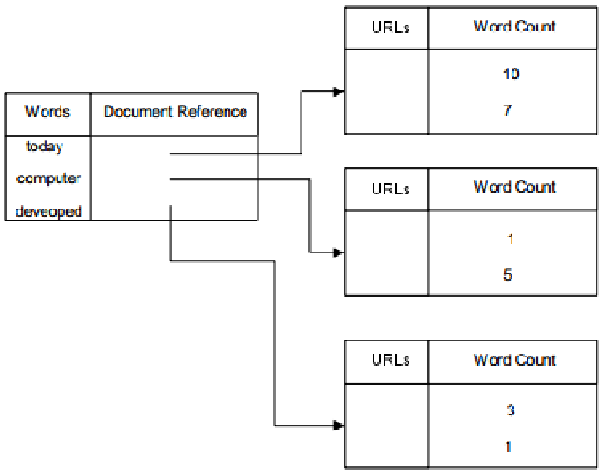

Figure 2: Word-level Inverted Index

The data structure shown in Figure 2 is created by the indexer as it processes each of the retrieved web pages. It is useful for keeping track of the *term frequency* and *document frequency* of each unique word in the entire corpus. *Term frequency* is the total number of times a particular term appears in a document. *Document frequency* refers to the total number of documents in which a term appears. A document-level index is also constructed by the indexer as it processes each web page. This document-level index holds a vector space model representation of each web page. The entries in the document-level index are eventually used to calculate similarity scores between queries and indexed documents. The structure of the document-level index is shown in Figure 3 [4].

The indexer generates web page (document) vectors by calculating weights for each term that appears in each document.

Using the word-level index illustrated in Figure 2, the term frequencies/word counts in a document are first normalized by the maximum term frequency within that particular document as follows.

$$tf_{norm} = \frac{tf}{\max tf}$$

Equation 1: Normalization of term frequencies

The normalized term frequency, tf_{norm} is then multiplied by the *inverse document frequency* of its corresponding term to get the final weight of the term. The *tf-idf* scheme employed in this implementation is a slightly modified version of the one described in [7]. Unlike [7], a *smoothing factor* of 0.5 is added to each term's weight. This was done to avoid having a zero query vector if the user decides to search for a word that appears in all web pages. This was considered important because a zero vector could eventually lead to a *division-by-zero* bug.

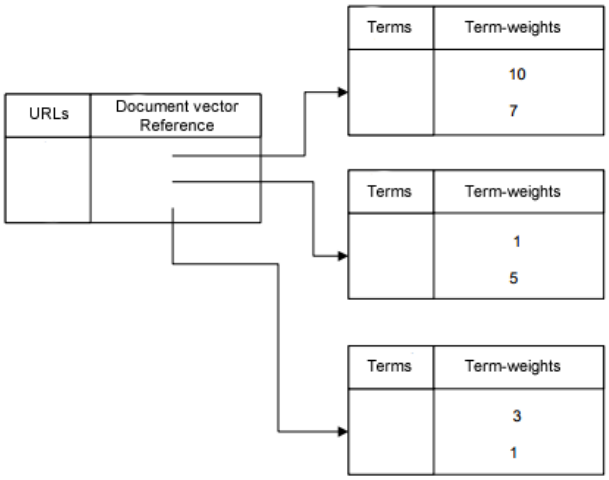

Figure 3: Document-level Index

The final weight of each term in each of the documents is calculated as shown in Equation 2 and the resulting document vectors are stored in the structure shown in Figure 3.

$$weight = tf_{norm} \times \log_{10}\frac{N}{n} + 0.5$$

Equation 2: tf-idf term weights

In equation 2, *N* represents the total number of indexed web pages while *n* is the number of documents in which a particular term appears (document frequency). The final implementation of the search engine described in this report uses the *tf-idf* term weighting scheme described in equation 2. Trials were also conducted using only the *term frequency* as a weighting scheme. The results showed very poor performance. The *tf-idf* weighting scheme clearly outperformed *term frequency*.

Numerous other weighting schemes that can be used in implementing a document retrieval system are presented in [7]. Just like the *tf-idf* weighting scheme used in this report, most of the other weighting schemes discussed in [7] also consider *N* and *n* as a means of gauging the importance (weight) of terms in a document. These weighting schemes are outlined in Figure 4 [7].

The generated indexes/document vectors are stored on disk using Perl's *Storable* module. Note that web pages are preprocessed and indexed as soon as a response to the URL request made to that page is returned. No actual web pages are ever stored on disk. Perl's *LWP::UserAgent* module was used to facilitate the process of retrieving HTML content from URLs.

2.3 Query Engine

The query engine is the component of the search engine responsible for transforming user queries into vectors and comparing these vectors with the indexed web pages. The query engine is also responsible for the query expansion/enhancement capabilities of the search engine. The query expansion/enhancement techniques used in this search engine implementation are discussed in Section 3.

The query engine handles query terms in a case-insensitive manner. The text preprocessor removes stop words from the query and *stems* the query terms. Query terms are then weighted using the same weighting scheme (*tf-idf*) discussed in Section 2.2. The vector representation of the query is then compared with the vectors representing each of the indexed web pages. As stated in subsection 2.2, the vectors for each indexed document are stored on disk and quickly retrieved at query-time. This is done in order to avoid the extra overhead of having to index all the documents all over again each time the user submits a query. The query engine uses a purely term occurrence-based relevance model. Therefore, all web pages that contain at least one of the terms in the query is considered to be relevant. The query engine compares the query vector with the vectors of all relevant web pages. The query engine does this by calculating the *cosine similarity* between the query vector and the web page (document) vectors. The *cosine similarity* between a query vector and a document vector is calculated as follows:

$$CosSim = \frac{dot\ product\ of\ query\ vector\ and\ document\ vector}{length\ of\ query\ vector\ \times length\ of\ document\ vector}$$

Equation 3: Cosine similarity between query vector and document vector

Each of the relevant documents is then ranked in descending order of *cosine similarity*. Therefore, a document with a *cosine similarity* value of 0.8 is considered to be more relevant to the user's query than one with a *cosine similarity* value of 0.75 and would appear higher in the search results ranking. After determining the relevance of the indexed relevant web pages to the user-supplied query, the query engine returns a list of URLs (search results) to the user via the user interface described in section 2.4.

2.4 Web-based Graphical User Interface

The final piece of the search engine implementation described in this report is the Graphical User Interface (GUI). A web-based GUI was constructed that enables a user to interact with the search engine via a web browser. The GUI essentially receives a query from the user and displays a list of web pages considered to be relevant to the query. The interaction between the Perl script and GUI was handled using HTML and Perl CGI. Snapshots of the web interface are shown in Figure 5.

Weighting System	Document term weight	Query Term weight
Best fully weighted system *tfc·nfx*	$\frac{\text{tf}\cdot\log\frac{N}{n}}{\sqrt{\sum_{vector}\left(\text{tf}_i\cdot\log\frac{N}{n_i}\right)^2}}$	$\left(0.5+\frac{0.5\ \text{tf}}{\max\ \text{tf}}\right)\cdot\log\frac{N}{n}$
Best weighted probabilistic weight *nxx·bpx*	$0.5+\frac{0.5\ \text{tf}}{\max\ \text{tf}}$	$\log\frac{N-n}{n}$
Classical idf weight *bfx·bfx*	$\log\frac{N}{n}$	$\log\frac{N}{n}$
Binary term independence *bxx·bpx*	1	$\log\frac{N-n}{n}$
Standard tf weight: *txc·txx*	$\frac{\text{tf}}{\sqrt{\sum_{vector}(\text{tf}_i)^2}}$	tf
Coordination level *bxx·bxx*	1	1

Figure 4: Typical term-weighting formulas

3. INTELLIGENT SEARCH COMPONENT

The search engine described in this report attempts to improve information retrieval efficiency and accuracy by systematically modifying the user's query. The general idea is to improve the relevance of the retrieved results by modifying the user's query in a manner that emphasizes important terms and adds other terms that are similar in meaning to the ones in the original query.

The intelligent component of the search engine emphasizes words in the user's query that are considered to be important. Emphasis is placed on a word by increasing the number of times it appears in the query. The search engine determines the importance of a word by considering its *document frequency*. The *document frequency* of a word is the number of documents that contain that word. Words with lower document frequencies are considered to be more important than words with higher document frequencies. Word emphasis is only carried out for queries that have more than one term. For example, the query *scholarship financial aid* could become *scholarship scholarship scholarship financial aid aid* under the informed assumption that the word *scholarship* is the most important word in the query and *aid* is next in line in order of importance. Only two words in a query are emphasized regardless of how long the query is.

The intelligent component of the search engine also considers words that have similar meanings to the words in a query. A thesaurus API from *BigHugeLabs* is used to achieve this [1]. The search engine sends POST requests to the *BigHugeLabs* Thesaurus API and the API returns a plain-text response containing synonyms for the terms in the query. The synonyms returned by the API are then *stemmed* and added to the original query.

Query expansion/enhancement techniques based on synonymy pose certain challenges. One of these major challenges is the possibility that the search results may actually contain an even greater number of irrelevant pages due to the words added to the query. This is mostly a result of the fact that certain synonyms added to the original query may not necessarily apply to the user's search context. For example, if a user enters a query *scholarship*, the intelligent component adds *erudition, learnedness* to the query. The actual intention might have been to find relevant information on *scholarship* within the context of *award, prize, aid*, etc. The intelligent component adds all synonyms regardless of context. In an attempt to reduce the amount of "noise" caused by modifying the user's original query, the weights of the added terms are reduced and made much lower than the weights of the terms in the original query.

Just like query expansion using synonyms, increasing the *term frequency* of certain words in a query is not a perfect technique either. There is absolutely no guarantee that the words considered to be important by the search engine are the exact words that are important to the user. Nevertheless, placing emphasis on certain words in a query was seen to improve the quality of search results in some cases. The search engine uses a combination of query expansion with synonyms and term emphasis for queries that have two or more terms. Table 1 and Table 2 present a comparison between search results generated using basic search, and search results using different aspects of the intelligent component. It can be seen that the intelligent component visibly influences the ranking of the top 5 results and sometimes even changes some URLs in the top results.

Table 1: Single term query expansion with synonyms

Query Term	*Scholarship*	
Rank	**Without synonyms**	**With synonyms**
1	http://financialaid.unt.edu/scholarships/external-scholarships/listing-external-scholarships	http://financialaid.unt.edu/scholarships
2	http://financialaid.unt.edu/scholarships	http://essc.unt.edu/finaid/scholarships.htm
3	http://essc.unt.edu/finaid/scholarships.htm	http://financialaid.unt.edu/scholarships/external-scholarships/listing-external-scholarships
4	http://financialaid.unt.edu/overview-scholarship-process	http://financialaid.unt.edu/overview-scholarship-process
5	http://www.opgf.unt.edu	http://financialaid.unt.edu/check-your-award-status-0

Table 2: Multiple term query expansion with synonyms and term emphasis

Query Term	*algorithm analysis*	
Rank	**Without synonyms and term emphasis**	**With synonyms and term emphasis**
1	http://www.unt.edu/behv	http://www.unt.edu/behv
2	http://pacs.unt.edu/behavior-analysis	http://pacs.unt.edu/behavior-analysis
3	http://cecera.unt.edu	http://www.cse.unt.edu/site/node/353
4	http://www.cse.unt.edu/site/node/353	http://www.unt.edu/untresearch/2012-2013/student-researchers.htm
5	http://pacs.unt.edu/behavior-analysis/content/north-texas-autism-project	http://cecera.unt.edu

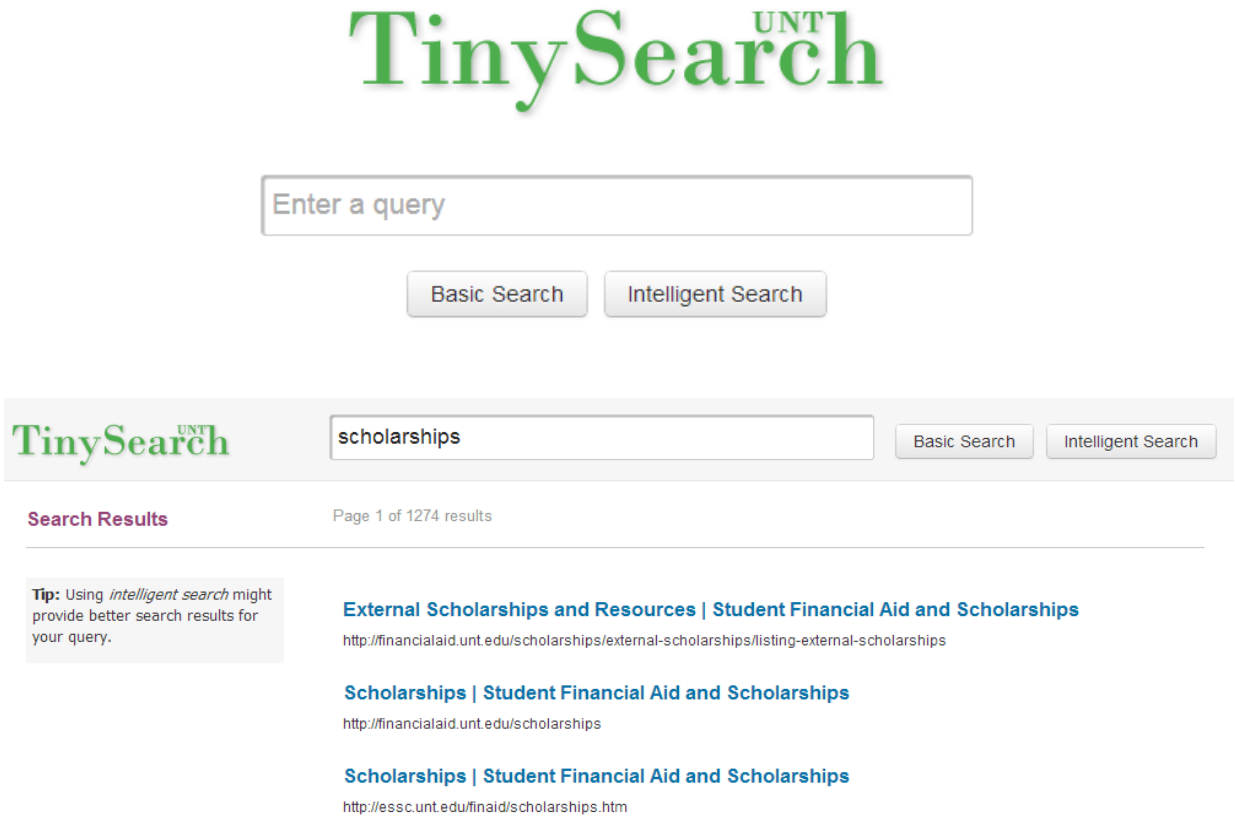

Figure 5: TinySearch Web User Interface

4. RESULTS AND ANALYSIS

This section discusses some of the challenges faced during the implementation of the search engine and also prevents an evaluation of the search engine's performance given a number of test queries.

4.1 Challenges Faced

One of the most crucial phases of the search engine's development was the implementation of the web crawler. Developing a smart and efficient web crawler proved to be quite a challenge. This is partly due to the inherent non-uniform nature of URLs and documents on the web. Most of the challenges centered on the nature of links found while crawling the UNT web domain. Many of these issues were discovered after generating the necessary indexes and required the index to be re-generated all over again. Unfortunately, generating indexes multiple times for 4500 documents proved to be a time-consuming process. The following are some of the typical issues that had to be tackled:

- **Duplicate Pages** – There were situations in which additional code had to be written to ensure that the crawler properly recognized pages that have slightly different URLs but link to the same document. An example of such a situation is the trailing slash '/' in some URLs e.g. `http://www.unt.edu` and `http://www.unt.edu/`. Other sources of duplicate pages that were encountered included anchor links to the same page and identical pages with different schemes (*http://* and *https://*). Special attention had to be paid to these instances, and consequently, additional code had to be written, to avoid redundant crawling of identical web pages. This helped increase the total number of unique documents indexed by the search engine.

- **Inconsistent file extensions:** In general, the search engine avoids file types that have non-text based content. This includes popular image formats and certain other proprietary file formats. It was discovered that web pages/documents on the UNT web domain did not adhere strictly to any rules concerning whether or not file extensions should be uppercase or lowercase. Therefore, there were situations where all .jpg file types were not crawled, but .JPG files appeared in the list of URLs crawled. To deal with this issue, additional code had to be written to deal with unwanted file types regardless of whether the file extensions were in uppercase or lowercase.

- **Storage Limitations:** As mentioned in Section 2, two indexes are built and stored on disk. The development server used during the implementation of the search engine had a per-user storage quota of 256MB. This placed tight constraints on the size of the generated indexes. Initially, the total size of the generated indexes was about 133MB. The bulk of this size was allocated to the word-level index. An effort was made to reduce the storage costs of the word-level index. Rather than store the index in the exact form shown in Figure 2, each of the hash references for each word was simply reduced to a single *inverse document frequency* value. The *inverse document frequency (idf)* of a word is given as:

$$idf = \log_{10} \frac{N}{n}$$

Equation 4: Inverse document frequency

N is the total number of indexed documents and *n* represents the total number of documents that contain the word. Mapping each word in the collection to a single *inverse document frequency* value rather than a reference to a hash of URLs and *term frequencies* drastically reduced the size of the word-level index, and brought the total storage requirements for both the document-level index and word-level index to about 35MB.

- **Memory Management:** An attempt was made to crawl up to 10000 links but it soon became apparent that the development server's per-user memory quota was not adequate for the task. As a result, the number of pages crawled was severely limited, and out of 6000 links gathered, 4500 of those links found to be valid and active were indexed.

- **Index file storage incompatibility between operating systems:** In an effort to gain access to more memory to enable crawling a lot more pages, an attempt was made to crawl and rebuild the index on a more powerful machine with the Windows Operating System installed. The *indexer* had absolutely no problems crawling 10000 or more pages. Nevertheless, there remained an unforeseen caveat. The development server which runs on the Unix Operating System was unable to read the indexes generated by the Windows machine due to file system incompatibilities. Consequently, the search engine had to settle for an index containing 4500 web pages due to the inadequate memory quota of the Unix-based development server.

4.2 Experiments and Evaluation

This section discusses the experiments used to test and evaluate the search engine's performance. The tests were conducted using the CSE machines at the Department of Computer Science and Engineering. A set of 10 carefully selected queries was used to test the information retrieval capabilities of the search engine. The top 10 search results for each query are then manually evaluated and discussed based on the author's relevance judgments. Also, the search results for each query are evaluated with and without the intelligent component. The ten test queries used are:

1. College of music
2. Computer science and engineering
3. Dance and theatre
4. Financial aid
5. School of graduate studies
6. Rada Milhacea
7. Health and recreation facilities
8. Student health insurance
9. College of engineering
10. Computer science research

The first query, *college of music*, returned search results with 90% precision in the top ten without the intelligent component. With the intelligent component, the same query returned top ten search results with 100% precision and the ranking of search results was slightly different from the ranking without the intelligent component. In the case of query 1, it can be seen that the intelligent component actually improves the retrieval efficiency of the search engine. Similar results are observed for test query 3 with precision values hitting the 90% mark.

Test query 5, *school of graduate studies*, returned top ten search results with a precision of 80% without the intelligent component. This was mostly due to pages containing *graduation* being considered to be relevant by the search engine, amongst other reasons. Intelligent search improves the results for test query 5 and gives a precision of 90%. Table 3 shows the (average) precision for the top ten search results returned for each of the test queries.

Table 3: Average precision for top ten search results for ten queries

Query	Precision (Basic search)	Precision (Intelligent search)
College of music	0.9	1
Computer science and engineering	1	1
Dance and theatre	0.9	0.9
Financial aid	1	1
School of graduate studies	0.8	0.9
Rada Milhacea	1	1
Health and recreational facilities	0.4	0.6
Student health insurance	0.8	0.7
College of engineering	0.7	0.7
Computer science research	0.7	0.7
Average Precision	**0.82**	**0.85**

For test query 6, *Rada Milhacea*, the top ten results with and without the intelligent component were exactly the same. Also, the search engine achieved 100% precision in the top ten results. This was expected as certain names tend to be quite unique and pages containing such names are more likely referring to the same person and are therefore absolutely relevant. This might not be the case for more common names though.

Searching for *health and recreational facilities* brings to light an interesting phenomenon. Due to the nature of the search engine, there is no consideration for inter-relation between words in a query. As a result, a search for *health and recreational facilities* without the intelligent component shows a precision of 40%. This happens because the search engine fails to understand the correlation between *health*, *recreational* and *facilities*. Rather than treat the search query as three related words, the search engine treats each individual word as a completely separate and independent entity. Using the intelligent component of the search actually improves the precision for test query 7, giving a precision value of 60%. The same lack of correlation between words was observed for the query *computer science research*. The search engine failed to understand the relationship between *computer*, *science* and *research* and instead treated each word as a separate entity. Treating each word in a query as a completely independent entity leads to a complete loss of context and although search results returned typically contained some or all of the words in the query, not all the results were particularly relevant to the search phrase within the desired context.

The query *student health insurance* leads to one of those situations where the intelligent component actually makes the search results worse. Without the intelligent component, the precision at 10 top results is 80%, and with the intelligent component, it becomes 70% as one more irrelevant document appears in the top 10. This same phenomenon was observed for the query *computer science research* as well.

Testing the query *college of engineering* with and without the intelligent component gives a precision of 70% for the top ten results. However, it was observed that without the intelligent document, the top three search results were not quite relevant. Using the intelligent component returns the same results but provides a better ranking with the irrelevant pages returned as the last three results in the top ten.

Although the intelligent component did not always improve the precision of the top ten documents, it always improved the total number of documents retrieved and hence, the *recall* of the search engine. Also, it was seen that using the intelligent component always affected the ranking of search results. Table 4 shows the top ten results for five queries using basic search and intelligent search.

5. RELATED WORK

Quite a number of attempts have been made to build more efficient search engines using various techniques and methods.

Google is probably the most famous and most successful attempt at building an excellent search engine. *Google* is a large-scale search engine which makes heavy use of the structure present in hypertext as a means of ranking search results [2]. Therefore, rather than using only cosine similarities and other similar measures for ranking search results, *Google* prioritizes a pages (document) using an importance score fundamentally measured by the number of other pages linking to that particular page and the number of other pages linked to by that page. Interesting insights into methods that greatly improve the scalability of a search engine are also presented in [2].

An attempt build a domain specific search engine using machine learning techniques is discussed in [5]. This was motivated by the belief that domain-specific search engines provide increased accuracy and extra features not possible with the general, web-wide search engines [5]. The approach in [5] proposes the use of machine learning techniques to automate the creation and maintenance of domain-specific search engines.

Rather than index only textual content, [3] uses image content in addition to associated text to index images, and presents the user with a selection that is potentially relevant to the user's query.

Table 5: Top ten results for five queries

Query	Rank	Basic search	Intelligent Search
College of music	1.	http://www.unt.edu/majors/umusi.htm	http://www.unt.edu/majors/umusi.htm
	2.	http://music.unt.edu/admissions	http://www.music.unt.edu/admissions
	3.	http://www.music.unt.edu/admissions	http://music.unt.edu/admissions
	4.	http://music.unt.edu/about/divisions-and-centers	http://music.unt.edu/about/divisions-and-centers
	5.	http://www.unt.edu/majors/umusic.htm	http://www.music.unt.edu/tcmm
	6.	http://music.unt.edu/advising/international.php	http://music.unt.edu/advising/international.php
	7.	http://music.unt.edu/advising/prospective.php	http://www.unt.edu/majors/umusic.htm
	8.	http://news.unt.edu/experts/category	http://music.unt.edu/advising/prospective.php
	9.	http://www.music.unt.edu/tcmm	http://music.unt.edu/about/divisions-and-centers/18
	10.	http://www.music.unt.edu/prospective-students	http://music.unt.edu/about/divisions-and-centers/18
Computer science and engineering	1.	http://www.unt.edu/majors/ucsci.htm	http://www.unt.edu/majors/ucsci.htm
	2.	http://www.unt.edu/pais/grad/gc_engineering.htm	http://www.unt.edu/majors/uelen.htm
	3.	http://www.unt.edu/majors/uelen.htm	http://www.unt.edu/pais/grad/gc_engineering.htm
	4.	http://www.cse.unt.edu/site/node/9	http://www.unt.edu/pais/grad/getec.htm
	5.	http://www.cics.unt.edu/education.html	http://www.unt.edu/majors/umeen.htm
	6.	http://www.cse.unt.edu/site/from_the_chair	http://engineering.unt.edu/undergrad
	7.	http://www.cse.unt.edu/site/node/449	http://www.cse.unt.edu/site/node/9
	8.	http://engineering.unt.edu/undergrad	http://engineering.unt.edu/content/faq
	9.	http://www.unt.edu/pais/grad/getec.htm	http://www.cse.unt.edu/site/node/449
	10.	http://www.cse.unt.edu/site/about	http://www.cics.unt.edu/education.html
Dance and theatre	1.	http://news.unt.edu/news-releases/unt-holds-faculty-dance-concert-student-alumni-and-acclaimed-guest-dance-educator	http://www.danceandtheatre.unt.edu/current-students
	2.	http://www.danceandtheatre.unt.edu/current-students	http://www.danceandtheatre.unt.edu/future-students
	3.	http://www.danceandtheatre.unt.edu/future-students	http://news.unt.edu/news-releases/unt-holds-faculty-dance-concert-student-alumni-and-acclaimed-guest-dance-educator
	4.	http://www.cas.unt.edu/events	http://cas.unt.edu/events
	5.	http://cas.unt.edu/events	http://www.cas.unt.edu/events
	6.	http://www.cas.unt.edu/news/spring-dance-theater-season-includes-thought-provoking-performances	http://news.unt.edu/news-releases/guest-lecturer-speak-latino-theater-april-9
	7.	http://news.unt.edu/news-releases/guest-lecturer-speak-latino-theater-april-9	http://www.music.unt.edu/calendar/library/caldetail.php?height=375&width=400&lechifre=41112&detail=128030
	8.	http://northtexan.unt.edu/content/cultural-impact	http://www.cas.unt.edu/news/spring-dance-theater-season-includes-thought-provoking-performances
	9.	http://www.danceandtheatre.unt.edu/events/2013/apr/celebrating-power-dance-faculty-dance-concert	http://northtexan.unt.edu/content/cultural-impact
	10.	http://www.danceandtheatre.unt.edu/productions-ticket-purchasing/past-productions-photos	http://www.danceandtheatre.unt.edu/events/2013/mar/advisingregistration-summer-fall-2013
Financial aid	1.	http://financialaid.unt.edu/overview-financial-aid-process	http://financialaid.unt.edu/overview-financial-aid-process
	2.	http://financialaid.unt.edu/financial-aid-forms	http://financialaid.unt.edu/financial-aid-forms
	3.	http://financialaid.unt.edu/your-rights-and-responsibilities	http://financialaid.unt.edu/your-rights-and-responsibilities
	4.	http://financialaid.unt.edu/finaidapply	http://financialaid.unt.edu/basics
	5.	http://financialaid.unt.edu/how-check-your-financial-aid-status	http://financialaid.unt.edu/basics/overview-financial-aid-process
	6.	http://financialaid.unt.edu/financial-aid-policies	http://financialaid.unt.edu/finaidapply
	7.	http://financialaid.unt.edu/basics	http://financialaid.unt.edu/how-check-your-financial-aid-status
	8.	http://financialaid.unt.edu/basics/overview-financial-aid-process	http://financialaid.unt.edu/financial-aid-policies
	9.	http://financialaid.unt.edu/staying-eligible-financial-aid	http://financialaid.unt.edu/staying-eligible-financial-aid
	10.	http://financialaid.unt.edu/satisfactory-academic-progress-requirements	http://financialaid.unt.edu/satisfactory-academic-progress-requirements

School of graduate studies	1.	http://www.unt.edu/pais/grad	http://www.unt.edu/pais/grad/gs_journalism.htm
	2.	http://www.unt.edu/gradmajors	http://www.unt.edu/pais/grad
	3.	http://www.unt.edu/pais/grad/index.htm	http://www.unt.edu/gradmajors
	4.	http://www.unt.edu/pais/grad/gs_journalism.htm	http://www.unt.edu/pais/grad/index.htm
	5.	http://essc.unt.edu/registrar/graduation.html	http://essc.unt.edu/registrar/schedule/major.html
	6.	http://essc.unt.edu/registrar/schedule/major.html	http://www.unt.edu/majors/usabr.htm
	7.	http://www.unt.edu/pais/grad/gs_visualarts.htm	http://www.unt.edu/pais/grad/gc_music.htm
	8.	http://www.unt.edu/pais/grad/gc_music.htm	http://www.unt.edu/pais/grad/gs_visualarts.htm
	9.	http://learningcenter.unt.edu/graduatestudentservices	http://learningcenter.unt.edu/graduatestudentservices
	10.	http://learningcenter.unt.edu/graduatestudentservices	http://www.opgf.unt.edu/postgrad.htm

6. CONCLUSIONS AND FUTURE WORK

This report discussed the practical steps taken in building a fully-functional but limited search engine using the vector space model. Evaluation results seem to indicate that the search engine is fairly effective since it always tended to provide quite a number of web pages in the top 10 search results that were relevant to user-supplied queries. The intelligent component of the search engine also turned out to be more effective than initially anticipated.

Possible future work centers on improving the intelligent aspects of the search engine. There are many techniques that could be used to achieve this including topic-sensitive *PageRank*, spell check, relevance feedback and many other added capabilities. However, what seems most cogent to the search implementation described in this report is improving the ability of the search engine to understand queries especially as regards the relationship between multiple words in the same query. This could be achieved using an efficient term proximity search method as described in [8] and [9]. This would greatly improve the search engine's performance especially in situations where it fails to understand queries that have implicit relationships between multiple terms.

7. ACKNOWLEDGEMENTS

The search engine described in this report was implemented under the excellent tutelage of Dr. Rada Milhacea at the Department of Computer Science and Engineering, University of North Texas.

8. REFERENCES

[1] BigHugeLabs Thesaurus API. http://words.bighugelabs.com/api.php

[2] Brin, S. and Page, L. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Computer Networks and ISDN Systems* 30 (1998), 107-117.

[3] Frankel, C., Swain M. J., and Athitsos, V. 1996. *WebSeer: An Image Search Engine for the World Wide Web*. Technical Report. The University of Chicago, Computer Science Department.

[4] Greenwood, M. 2001. *Implementing a Vector Space Document Retrieval System*. Department of Computer Science, The University of Sheffield, UK.

[5] McCallum, A., Nigam, K., Rennie, J., and Seymore, K. 1999. *Building Domain-Specific Search Engines with Machine Learning Techniques*.

[6] Milhacea, R. CSCE 5200 Information Retrieval and Web Search Lecture Notes, 2012.

[7] Salton, G and Buckley, C. 1988. Term-Weighting Approaches in Automatic Text Retrieval. In *Information Processing & Management* 24, 5 (1988), 513-523.

[8] Schenkel, R., Broschart, A., Hwang, S., Theobald, M., and Weikum, G. 2007. Efficient Text Proximity Search. In *Proceedings of the 14th International Conference on String Processing and Information Retrieval* (2007), 287-299.

[9] Yan, H., Shi S., Zhang, F., Suel, T., and Wen, J. 2010. Efficient Term Proximity Search with Term-Pair Indexes. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management* (2010), 1229-1238.