PHRASE QUERYING FOR NEEDLE SEARCH ENGINE

By Priyadarshini Selvam

A Project
Submitted to the Faculty of Graduate School of the
University of Colorado at Colorado Springs
In Partial Fulfillment of the Requirements
For the Degree of Master of Science
Department of Computer Science
Fall 2008

Advisory Committee

Dr. Jugal Kalita

Dr. Edward Chow

Dr. Xiaobo Zhou
ACKNOWLEDGEMENTS

This project has been accomplished with great support and valuable suggestions of the following people.

First, I would like to sincerely thank my advisor Dr. Jugal Kalita for his continuous motivation and guidance.

I also thank Dr. Edward Chow and Dr. Xiaobo Zhou for their valuable suggestions.

Special thanks to -

Sunil Bhave, Sonali Patankar, Padmaja Adipudi and Yi Zhang who had worked on the Needle project.

Rob Garvie and Donovan Thorpe from UCCS IT Department for their help and suggestions that made this search engine Web accessible.

All my other friends who had attended research meetings and provided good feedback.

Last but not the least, I wish to thank my parents, husband Selvam and daughter Varsha for their immense support at all times.
ABSTRACT

In this report, I present the design and implementation of a phrase search component for the Needle search engine. The Needle search engine is a search engine being built by UCCS students. Each component of the search engine is developed independently applying recent and improved methodologies. The components are then integrated into the Needle search engine. The phrase search component will be a new feature added to the Needle search engine and complement the text and image search features. Three common approaches for phrase querying Inverted index, NextWord index, Combined Inverted and NextWord index have been studied, implemented and the results analyzed to determine the best approach for the phrase search feature of the Needle search engine.
Table of Contents

1. Introduction............................................................................................................. 5
2. Phrase search and underlying problems................................................................. 6
  2.1 Inverted Index .................................................................................................. 6
  2.2. Phrase Index .................................................................................................. 7
  2.3 NextWord Index............................................................................................... 8
  2.4 Combined Inverted and NextWord Index......................................................... 8
3. Problem Statement and Approach ........................................................................... 9
4. Architecture Overview............................................................................................. 10
5. Process Flow for a phrase query............................................................................. 11
6. Background processing........................................................................................... 12
  6.1 Repository ....................................................................................................... 12
  6.2 Parser ............................................................................................................... 13
  6.2.1 Inverted Index ............................................................................................. 13
  6.2.2 NextWord Index............................................................................................ 14
  6.2.3 Combined Inverted and NextWord Index.................................................... 14
  6.3 Database Design................................................................................................ 14
  6.4 Searching.......................................................................................................... 20
  6.4.1 Inverted Index Search ................................................................................. 20
  6.4.2 NextWord Index Search .............................................................................. 21
  6.4.3 Combined Inverted and NextWord Index.................................................... 22
7. Results..................................................................................................................... 23
  7.1 Data .................................................................................................................. 23
  7.2 Estimation of precision ..................................................................................... 27
  7.3 Response Time................................................................................................... 27
  7.4 Analysis of Results ............................................................................................ 28
    7.4.1 Comparison of response time between three approaches using Inverted Index........................... 29
    7.4.2 Analysis of NextWord Index ...................................................................... 30
    7.4.3 Comparison of response time between two approaches using Combined Index........................... 31
    7.4.4 Comparison of results obtained by each index type .................................... 32
    7.4.6 Comparison of Response Time between different phrase lengths and different document sizes. .................................................................................. 34
    7.4.7 Comparison of Response time with number of search results returned......... 34
    7.4.8 Comparison of Response time with phrase structure............................... 35
    7.4.9 Comparison of response time with PageRank ........................................... 36
8. Integration with Needle Search Engine ................................................................... 42
  8.1 Needle Crawler ................................................................................................. 42
  8.2 Page Ranking .................................................................................................... 42
9. Problems faced........................................................................................................ 42
  9.1 Different file types and sizes............................................................................. 42
  9.2 Poorly formatted web documents. ..................................................................... 42
  9.3 Disk Space ........................................................................................................ 43
  9.4 Get a fresh repository ....................................................................................... 43
  9.5 Compute PageRank .......................................................................................... 43
1. Introduction
The World Wide Web has grown significantly in the last decade. The need of search engines to adopt new and improved techniques to retrieve information from the massive Web data is also growing at a fast pace. The problems faced by search engines are to keep the information up-to-date and to return appropriate results for a user’s query almost instantly. A considerable portion of requests received by Web search engines are based on phrase queries such as New York, cheap airfares etc. The efficiency of phrase querying depends on how the data is indexed, reducing the number of disk accesses and processing overhead required for query processing. The Needle search engine is a search engine being developed by students at UCCS. The Needle search engine currently has a Crawler component and a text search feature, thanks to the work done by [Yang 2005], [Zhang, 2006], [Patankar, 2006], [Patwa, 2006], [Kodavanti, 2006]. To complement the text search feature, this project aims to add a phrase search feature for the Needle search engine. In addition to this three techniques most commonly used for phrase searches – Inverted index, NextWord index and Combined Inverted and Next word index are analyzed and compared.

2. Phrase search and underlying problems

Phrases are lists of words that indicate the word as well as their order. Examples of phrases are Apple Macintosh, New York etc. The length of most phrases searched is 2, and less than 1% of the phrases have a length longer than 6. Most phrases also include a common term such as and, to, the, etc. [Williams 2004].

2.1 Inverted Index

The most common approach used for phrase search is the Inverted Index [Witten 1999]. An inverted index is a posting list that has a vocabulary and the occurrences of each word in the documents. A posting is a triplet of the form

\[ \langle d, f_{d,t}, [o_{1}, ..., o_{f_{d,t}}] \rangle \]

where d is the document identifier containing the term t, f_{d,t} is the frequency of t in d and o are the positions in d at which t occurs [Witten 1999]. An example of a posting list is given below.

\[ <101, 4,[5, 11, 25, 77]> \]
Using an inverted index, processing time for a phrase query depends on the number of terms in the query, the frequency of each term in the phrase. Evaluating a phrase query requires obtaining the posting list for each word and combining the results.

2.2. Phrase Index

The second approach used for a phrase search is a phrase index. A phrase index is an inverted index where the vocabulary consists of phrases. For each phrase query the phrase vocabulary is searched. When an exact match is found, the posting list is retrieved. When an exact match is not found, the evaluation needs to be done by following other methods discussed in next section [Gutwin 1998].

The advantage of a full or partial phrase index is evaluating a phrase query from such an index is straightforward. The disadvantage is it is not possible to determine all the phrases users are likely to search beforehand and if a user searches for a phrase that is not
indexed, no results can be returned to the user. The index size increases as the number of terms in the indexed phrase.

### 2.3 NextWord Index

A next word index is a mapping from pairs of terms to lists. A next word index can be used in a straightforward way to evaluate phrase queries [Bahle 2001a].

![Diagram of NextWord Index](image)

**Figure 3. Example of NextWord index**

Figure 3 shows an example of a NextWord index. For the term hatful the NextWord of and the corresponding posting list is stored. The term of has multiple terms as NextWords and the posting list for each vocabulary-NextWord pair is stored. Phrase query evaluation using a NextWord index is simple compared to the other two methods mentioned above. However the index size of the NextWord word index increases considerably depending on the number of indexed terms.

### 2.4 Combined Inverted and NextWord Index

The fourth approach is a Combined Inverted Common-NextWord index. Each word has an inverted list and the common first words [15] alone have a NextWord index [Williams 2004]. This approach reduces the index size required for a complete NextWord index and provides a better query time compared to inverted indexes alone.
In Figure 4, the NextWords are only stored for common terms in, new. The posting list for each common word, NextWord pair is stored. All other terms only have the posting list corresponding to the word itself and not the NextWord. This approach combines the advantages of both the inverted and NextWord index.

3. Problem Statement and Approach

There are two major factors that influence the efficiency of a search engine.

1. Any search engine should be capable of locating the exact information within the domain or area of interest that user is looking for. The objective of this project is to develop a phrase search component for the Needle search engine. The Needle search engine currently focuses on the UCCS domain but is scalable to other domains as well.

2. Phrase search feature is an integral part of any search engine. Understanding the properties of phrases is crucial in the design of the search engine. Analysis of Web queries has shown that the average length of a user query is 2.5 words. Less than 1% of the queries have a length of 6 words or more [3]. A large number of the phrases also have a common term such as *to*, *the*, *in*, *at* etc., and the common term rarely occurs at the end of a query. Examples of such queries are ‘careers in computers’, ‘cost of gasoline’, ‘President of United States’.

These factors are taken into account for the phrase search component of the Needle search engine. In addition to providing a phrase search component to the Needle
search engine, this project also compares critical factors like precision, response time and storage overheads of the three approaches: Inverted Index, NextWord Index and Combined Inverted NextWord Index.

4. Architecture Overview

A high level overview of the Needle search engine is shown here. Details of the individual components will be discussed in further sections.

![Diagram of Needle Search Engine Components](image)

Figure 5. Components of the Needle Search Engine

The search engine components are implemented using Perl on a Linux platform. MySQL is used as the backend database.

The components are built in house by computer science students in UCCS. The Needle crawler has a versatile design with a restart flag and is capable of crawling millions of Web pages. Additional Web pages can be crawled incrementally to expand the scope of
the search engine. An individual component for text parser called CAT SEARCH was developed but is currently not integrated into the search engine. This project implements the phrase search component for the search engine. The phrase search component is capable of performing both word as well as phrase searches. Development of the image search component is currently in progress and will be integrated when the work is completed.

5. Process Flow for a phrase query

The Needle front end provides user options to search based on text/phrase or image. When a user selects a phrase search option and enters a phrase, the query processor determines if the phrase has a common word. If a common word occurs, the input phrase is broken into common word-NextWord pair. The words occurring before and after the common word-NextWord pair are searched in the inverted index. The common word-NextWord pair is searched in the combined index. The results obtained from both indexes are combined. If the input phrase does not have a common word, search is done on inverted index. The results obtained are further sorted based on the weights computed for the phrase. The weight of a phrase is computed based on the page rank of the document in which the phrase occurs and the location within the document in which the phrase is found. Figure 6 below shows the process flow for a phrase query.

![Figure 6. Process Flow for a Phrase Search.](image)

- **Needle Front End**
  - Select search type (Text/Phrase or Image)
  - Enter search text/phrase
- **Query Processor**
  - Check for a common word in search text.
  - When found, Identify common word-NextWord pair
o Identify URL’s matching words before and after the common word-NextWord pair using inverted index.

o Identify URL’s matching common word-NextWord pair using combined index.

o Retrieve the URL’s returned by both inverted and combined next word index and order them by page rank and location weight.

o If search text does not have a common word, return results from inverted index.

- Database Engine
  o Hosts tables used by the search engine.

6. Background processing

6.1 Repository

The Needle crawler was re-run on the UCCS domain since many Web pages that were downloaded previously had been modified or were removed and the URLs did not exist. The Crawler was run for about 10 days on different domains (uccs.edu, cudenver.edu, Colorado.edu, mines.edu, colostate.edu) and required to be restarted a couple of times. The file types that could be downloaded such as html, php etc., are specified in a config file. Crawled URL’s with matching document type defined in the config file gets downloaded. The number of URLs crawled are 1,160,000. The number of downloaded text documents is approximately 150,000.

<table>
<thead>
<tr>
<th>Domain</th>
<th># of documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uccs.edu</td>
<td>12000 (excluding calendar links)</td>
</tr>
<tr>
<td>Cudenver.edu</td>
<td>33009</td>
</tr>
<tr>
<td>Colorado.edu</td>
<td>38236</td>
</tr>
<tr>
<td>Mines.edu</td>
<td>15437</td>
</tr>
<tr>
<td>Coloate.edu</td>
<td>27235</td>
</tr>
</tbody>
</table>

Table 1: Number of documents from each domain.
6.2 Parser

The parser is an important component of a search engine. Efficient query processing depends on how the data is indexed and it is the role of the parser to read the stored document collection and build an index. Developing a configurable parser was the most important and time consuming part of the project. The parser needed to be able to process different document types. Different tags and levels of the tags that may occur in a document increased the complexity of the parser.

The parser program reads the downloaded text documents. The path where the documents are stored is specified in the config file. The parser opens each document and reads the document. The contents that occur within specified tags such as <meta>, <keyword>, <para> etc., are parsed and indexed. The tags that should be included are configurable and specified in a config file. The tag or the document position at which each word occurs should be saved to evaluate a phrase search. The parser saves three different indexes for each word read in the document, i.e., inverted index, NextWord index and combined inverted NextWord index.

6.2.1 Inverted Index

The parser to index the words in the stored collection parses the specified tags in the document. Stop words specified in the config file are excluded, all other words are indexed. For each indexed word the corresponding document identifier and the position or tag at which the word occurs are stored.

The inverted index for the stored document collection has 6,036,802 entries. This is less than the number of entries in the NextWord Index discussed below. Most commonly occurring words such as a, an, the are called stop words or common words. Many of
these stop words are function words that are used to join longer words [15]. The stop words may or may not be significant in a small phrase query but does not alter the meaning of a long phrase are sentence. For example a query ‘flights to New York’ and ‘flights from New York’ are two different queries in which the user expects to get exact results. However in a long query such as ‘What are the historic sites to visit in New York’, the meaning of the query is not altered when stop words are removed and the search phrase is ‘historic + sites + visit + New +York’. The stop words are identified in the config file and are excluded from the index.

6.2.2 NextWord Index
The parser reads every document in the stored collection, indexes every word-NextWord pair that occurs within the tags specified in the config file. All words are included in the index. The posting list corresponding to each word-NextWord pair, the position at which it occurs is also stored.
The NextWord index for the stored document collection has 11,076,099 entries. The index size is huge because all words (common words) and the next word are included in the index.

6.2.3 Combined Inverted and NextWord Index
In this approach, the parser indexes the word or the Common-NextWord pair based on each word in the document. If the word is a common word that is specified in the config file, the common-next word pair for the common word is stored. If the word is not a common word only the inverted file for the word is stored. The index size is reduced by this approach.
The Combined Inverted NextWord Index for the stored document collection has 3,398,803 entries. This is much less than the number of entries seen in Inverted and NextWord Indexes.

6.3 Database Design
The Needle search engine has shared as well as module specific tables. The tables required for the phrase search component has been added in addition to the existing tables that were previously created by the Crawler and PageRank modules. Figure 7 shows the ER diagram for the phrase query component. This diagram identifies the entities and the relationship among the entities.

Entities:

- **URL** - Used to store information related to the URL’s crawled and uniquely identified by url_id.
- **Dictionary** – Used to store all words identified by the parser from the contents of the downloaded documents and uniquely identified by column ‘id’.
- **recent_query** – Used to store the recently searched texts and uniquely identified by searched text.
- **Combined** – One of the index table used to store information about the common word, its next word and the location of the common word appearing in the URL.
Common words are defined in a config file. The same word may appear with in a URL at different locations. In that case multiple entries appear in this table. Entries in this table are uniquely identified by keyword_id, NextWord_id, location_id and url_id.

- **Inverted** – One of the index tables used to store information about all the keywords appearing in the parsed documents with the location of the word within the URL. Common words specified in the config files are not part of this index table. The same word may appear with in a URL at different locations. In that case multiple entries appear in this table. Entries in this table are uniquely identified by keyword_id, location_id, url_id.

- **NextWord** - One of the index tables used to store information about the keyword, its next word and the location of the word appearing in the URL. The same word may appear with in a URL at different locations. In that case multiple entries appear in this table. Entries in this table are uniquely identified by keyword_id, NextWord_id, location_id, url_id.

**Relationships:**

- For every instance of entity URL, there will be zero; one or more instances of the entities Combined, Inverted or NextWord.
- For each instances of entities Combined, Inverted or NextWord, there will be one and only one instance of the entity URL.
- For each instance of ‘id’ in entity Dictionary, there will be zero, one or more instances of keyword_id in the entity Combined, Inverted or NextWord.
- For each instances of keyword_id in entity Combined, Inverted or NextWord, there will be one and only one instance of ‘id’ in the entity Dictionary.
- For each instance of ‘id’ in entity Dictionary, there will be zero, one or more instances of NextWord_id in the entity Combined or NextWord.
- For each instances of NextWord_id in entity Combined or NextWord, there will be one and only one instance of ‘id’ in the entity Dictionary.
Figure 8. ER diagram for phrase search component
Details of the primary tables used by the parser and search utilities are given below.

**Table Name: URL**

The URL table has the details of the URLs crawled by the crawler. The URL_id field uniquely identifies every record in the URL table. The doc_type_id field stores if the document corresponding to the URL is a text or an image. This field will be used by the text and image parser components. The URL table also has additional fields that are used to compute the page rank.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url_id</td>
<td>int(10) unsigned</td>
<td>Primary</td>
<td>Autoincrement field to identify URL</td>
</tr>
<tr>
<td>url</td>
<td>varchar(255)</td>
<td></td>
<td>URL Text</td>
</tr>
<tr>
<td>doc_type_id</td>
<td>tinyint(3) unsigned</td>
<td></td>
<td>Type of URL text, image.</td>
</tr>
<tr>
<td>container_url</td>
<td>int(10) unsigned</td>
<td></td>
<td>The URL id of the container URL.</td>
</tr>
<tr>
<td>Title</td>
<td>varchar(255)</td>
<td></td>
<td>Title for the URL obtained from anchor text</td>
</tr>
<tr>
<td>cluster_id</td>
<td>int(10) unsigned</td>
<td></td>
<td>Used by Page Ranking</td>
</tr>
<tr>
<td>url_hash</td>
<td>varchar(32)</td>
<td></td>
<td>32 byte URL hash</td>
</tr>
<tr>
<td>Retry_attempt</td>
<td>int(10) unsigned</td>
<td></td>
<td>Number of retry attempts</td>
</tr>
<tr>
<td>Crawled</td>
<td>char(1)</td>
<td></td>
<td>A switch with values Y, N, E for Yes, No, Error values.</td>
</tr>
<tr>
<td>err_detail</td>
<td>char(2)</td>
<td></td>
<td>Error detail</td>
</tr>
<tr>
<td>udomain_id</td>
<td>int(11)</td>
<td></td>
<td>Domain id for url</td>
</tr>
</tbody>
</table>

**Table Name: Dictionary**

The Dictionary table stores a unique word id for every new word read by the parser. The inverted, NextWord and combined indexes reference only the word ids.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Int(10) unsigned</td>
<td>Primary</td>
<td>Autoincrement field that is a unique identifier for every new word.</td>
</tr>
<tr>
<td>Word</td>
<td>varchar(32)</td>
<td></td>
<td>The word itself</td>
</tr>
</tbody>
</table>
Table Name: NextWord

The NextWord table is the primary table used to store the NextWord index. The fields keyword_id and nextword_id are used to store the word-nextword pair. The position at which the word-NextWord pair occurs is also stored for each pair. This position is used in identifying matching phrases for phrases having more than two words.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword_id</td>
<td>Int(10) unsigned</td>
<td>Primary Key</td>
<td>First word id referencing Dictionary table</td>
</tr>
<tr>
<td>url_id</td>
<td>Int(10) unsigned</td>
<td>Primary Key</td>
<td>URL id referencing the Down_URL table</td>
</tr>
<tr>
<td>NextWord_id</td>
<td>Int(10) unsigned</td>
<td>Primary Key</td>
<td>Nextword Id referencing the Dictionary table</td>
</tr>
<tr>
<td>position</td>
<td>Int(10) unsigned</td>
<td></td>
<td>Position of the word, NextWord pair in the document</td>
</tr>
<tr>
<td>location_id</td>
<td>smallint(5) unsigned</td>
<td>Primary Key</td>
<td>Location of the word, NextWord pair in the document</td>
</tr>
<tr>
<td>update_date</td>
<td>Date</td>
<td></td>
<td>Date the document was parsed and indexed.</td>
</tr>
<tr>
<td>frequency</td>
<td>mediumint(8) unsigned</td>
<td></td>
<td>Number of occurrences of the word, NextWord pair</td>
</tr>
</tbody>
</table>

Table Name: Inverted

The Inverted table is the primary table used when query evaluation is done using inverted index. The index does not include the common words. The common words are specified in a config file. The fields keyword_id, url_id, position and location uniquely identifies occurrence of each word in a document. The location_id and position fields are used to identify multiple occurrences of a phrase in a document.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword_id</td>
<td>Int(10) unsigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>url_id</td>
<td>Int(10) unsigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>position</td>
<td>Int(10) unsigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>location_id</td>
<td>smallint(5) unsigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>update_date</td>
<td>Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frequency</td>
<td>mediumint(8) unsigned</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Combined table stores the Combined Inverted NextWord Index. The table structure is similar to NextWord but the index is different. The first word or keyword in the Combined index must be a common word specified in the config file. The word occurring after a common word is also stored and included in the index. The position at which the common word-nextword pair occurs is also stored similar to the Inverted and NextWord indexes.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword_id</td>
<td>Int(10) unsigned</td>
<td>Primary</td>
<td>First word id referencing Dictionary table</td>
</tr>
<tr>
<td>url_id</td>
<td>Int(10) unsigned</td>
<td>Primary</td>
<td>URL id referencing the Down_URL table</td>
</tr>
<tr>
<td>position</td>
<td>int(10) unsigned</td>
<td></td>
<td>Position of the word, NextWord pair in the document</td>
</tr>
</tbody>
</table>

### Field Details

- **keyword_id**: Int(10) unsigned, Primary Key, word id referencing Dictionary table.
- **url_id**: Int(10) unsigned, Primary Key, URL id referencing the Down_URL table.
- **NextWord_id**: Int(10) unsigned, Primary Key, Nextword Id referencing the Dictionary table.
- **position**: int(10) unsigned, Position of the word, NextWord pair in the document.
<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location_id</td>
<td>smallint(5)</td>
<td>Location of the word, NextWord pair in the document</td>
</tr>
<tr>
<td>update_date</td>
<td>Date</td>
<td>Date the document was parsed and indexed.</td>
</tr>
<tr>
<td>frequency</td>
<td>mediumint(8)</td>
<td>Number of occurrences of the word, NextWord pair</td>
</tr>
</tbody>
</table>

The above inverted, NextWord and combined index tables also have additional column to store the location weight for each word and nextword.

**6.4 Searching**

The Needle front end has features for a user to input a search text and select options for text, phrase or image search. When the phrase search option is selected, the input search text is processed and a database query statement is formulated. This database query is executed and the results returned to the screen.

**6.4.1 Inverted Index Search**

In this method, to evaluate a phrase query a greedy approach has been used. Common words in the input phrase are removed.

1. Break input search phrase into individual words.
2. Remove common words.
3. Retrieve the documents in which all words in the search phrase is found.

Query evaluation using Inverted Index

Major search engines such as Google, Yahoo provide different levels of search such as exact phrase match, all words in the phrase occurring anywhere in the document and one or more words in the phrase occurring in the document. To compare the precision and response times obtained by these three methods, the following three approaches were experimented.

1. Retrieve the documents in which any one word in the search term occurs.
2. Retrieve the documents in which all words in the search term occurs anywhere in the document.
3. Retrieve the documents in which the words in the search term occurs within a specific location (title, anchor text, paragraph). The words may occur in any order.

Example of Inverted Index Search:

Query Term – ‘Network and Security’

1. The search term is broken into individual words ‘network’, ‘and’ and ‘security’. The common word ‘and’ is removed.
2. Matching results for words ‘network’, ‘security’ are retrieved.
3. Based on the above three approaches, the retrieved posting lists are further processed and combined to obtain the final results.

In the first approach, all documents with either ‘network’ or ‘security’ occurring one or more times are retrieved.

In the second approach, documents that have both words ‘network’ and ‘security’ anywhere in the document are retrieved.

In the third approach, documents that have both words ‘network’ and ‘security’ occur in the same location of a document are retrieved.

As mentioned previously, the index size is fairly high although stop words are excluded. The search results return documents, which contain a word that occurs in the phrase, but not the exact phrase itself. However there is no complex processing required. This approach is simple, returns results faster but the precision of the results are not good.

6.4.2 NextWord Index Search

In this method, the given input phrase is broken into word pairs. The posting list for each word pair is fetched from the index and the results returned are combined. The NextWord index can perform exact phrase matching for up to 6 words in a phrase. The processing involved in this implementation is slightly complex but result obtained is accurate. The index size is however much higher than the inverted index.

1. Break input search phrase into word pairs.
2. Retrieve the matching document list for each word pair.
3. For each document list that has all the word pairs, check if the word pair occurs in same sequence as the input phrase by verifying the location and position of each word pair.
4. Return the document list that has a complete
Query evaluation using NextWord Index

Example of NextWord Index search:

Query term – ‘University of Colorado at Colorado Springs’

1. The search term is broken into word pairs ‘university of’, ‘colorado at’, ‘colorado springs’

2. The posting list for each word pair is retrieved.

3. The results are processed to obtain the documents in which all the word pairs occur in the same sequence as the query term.

The results obtained by this method have the highest precision. However the index size and complex processing result in a slow response time.

6.4.3 Combined Inverted and NextWord Index

To evaluate an input phrase, if there is any common word in the phrase, then the common word-next word is searched in the Combined Inverted NextWord index. If the input phrase does not have any common word, the phrase is evaluated similar to an inverted index. This is done to avoid the cost of processing and combining involved if an exact query processing is done using NextWord index. The combined inverted and NextWord index finds exact match for phrases containing up to 4 words. For phrases that are longer than 4 words and have a common word, extra words are removed based on where a common word occurs in the phrase. This has been done because less than 2% of phrase queries have more than six words [3]. For a long phrase or a sentence that has many common words, all common words are dropped and the first four words are used in formulating the query and matching results are retrieved.

1. Determine if input phrase has a common word.
2. If common word occurs, find the matching document that has the common word-next word pair from the combined index.
3. If the input phrase does not have a common phrase evaluate using inverted index.
Two variations of query processing using the Combined Inverted NextWord Index have been experimented.

1. When a common word occurs in the search term, retrieve the documents that have the common word – NextWord pair from Combined index. Documents that have the words occurring before and after the common word-NextWord within the same location (title, anchor text, heading) are retrieved.

2. When a common word occurs in the search term, retrieve the documents that have the common word – NextWord pair from Combined index. Documents that have the words occurring before and after the common word-NextWord within the same location (title, anchor text, heading) and in the same order as the input text are retrieved. This involves more processing that the above two approaches, but the results are more accurate.

The index size of the Combined Inverted NextWord index is much less than Inverted Index. The results obtained using this approach was found to be more precise when the query had a common word followed by the keyword. For search terms that have no common term, the performance was the same as Inverted index.

### 7. Results

#### 7.1 Data

The total number of URLs crawled in the UCCS, CU Denver, Colorado, Colorado State and college of mines domains is about 1,160,000. The number of documents in the stored collection is 125,000 excluding the calendar urls. The number of rows in each index is given in Table 1 below.

<table>
<thead>
<tr>
<th>Index Type</th>
<th>Number of Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverted Index</td>
<td>6036802</td>
</tr>
<tr>
<td>NextWord Index</td>
<td>11,076,099</td>
</tr>
<tr>
<td>Combined Inverted NextWord Index</td>
<td>3,398,803</td>
</tr>
</tbody>
</table>

**Table 2. Number of entries in each index type for the stored document collection.**

Table 3 below shows the data length and index length.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rows</th>
<th>Avg_row_length</th>
<th>Data_length</th>
<th>Max_data_length</th>
<th>Index_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>NextWord</td>
<td>11076099</td>
<td>25</td>
<td>276902475</td>
<td>107374182399</td>
<td>790872064</td>
</tr>
<tr>
<td>Inverted</td>
<td>6036802</td>
<td>29</td>
<td>150920050</td>
<td>124554051583</td>
<td>240161792</td>
</tr>
<tr>
<td>Combined</td>
<td>3398803</td>
<td>25</td>
<td>98565287</td>
<td>107374182399</td>
<td>197738496</td>
</tr>
</tbody>
</table>
Table 3. Data length and index length for each index type.

From the above table it can be seen that the average data length for each entry is the same for the NextWord and Combined Inverted NextWord index. The average row length for the NextWord and Combined index is slightly higher because the length of the common words in most cases is less than 5. The data length and the index length for the both Inverted and Combined inverted NextWord index are similar. However there is a significant improvement in the index length of the Combined Inverted NextWord index compared to the NextWord Index.

<table>
<thead>
<tr>
<th># of pages indexed</th>
<th>Rows in Inverted Index</th>
<th>Rows in NextWord Index</th>
<th>Rows in Combined Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>45000</td>
<td>701922</td>
<td>1279202</td>
<td>412585</td>
</tr>
<tr>
<td>75000</td>
<td>2143309</td>
<td>4000346</td>
<td>1288875</td>
</tr>
<tr>
<td>110000</td>
<td>4127594</td>
<td>7606690</td>
<td>2363113</td>
</tr>
<tr>
<td>125000</td>
<td>4631891</td>
<td>8520367</td>
<td>2624859</td>
</tr>
<tr>
<td>152000</td>
<td>6036802</td>
<td>11076099</td>
<td>3398803</td>
</tr>
</tbody>
</table>

Table 4. Number of entries in each index type.

Figure 9. Graph showing the increase in index size as the number of documents indexed increases.
Table 3 and Figure 8 show, the increase in number or rows for each index type as the number of pages parsed increase. It can be seen that, the rate at which inverted index increases is smaller than the rate at which NextWord and combined indexes increase as more pages are parsed.

<table>
<thead>
<tr>
<th># of pages</th>
<th>Rows in NextWord Index</th>
<th>Rows in Combined Inverted and NextWord Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>45000</td>
<td>1279202</td>
<td>1114507</td>
</tr>
<tr>
<td>75000</td>
<td>4000346</td>
<td>3432184</td>
</tr>
<tr>
<td>110000</td>
<td>7606690</td>
<td>6490707</td>
</tr>
<tr>
<td>125000</td>
<td>8520367</td>
<td>7256750</td>
</tr>
<tr>
<td>152000</td>
<td>11076099</td>
<td>9435650</td>
</tr>
</tbody>
</table>

Table 4. Comparison of number of entries in NextWord and combined index type as the number of indexed pages increases.

![Graph showing the increase in number of entries for NextWord and combined index as the indexed document increases.](image)

Figure 10. Graph showing the increase in number of entries for NextWord and combined index as the indexed document increases.
The index and data file sizes of the Inverted and Combined indexes are shown below in figure 10.

**Figure 11. Index file size for Nextword and Combined Index**

**Figure 12. Data file size for Nextword and Combined Index.**
Figure 13. Time taken by Parser for different sizes of document collection.

The time taken by the parser to build the three index types were recorded for as documents downloaded from each domain were indexed. It can be seen that the time taken is directly proportional to the number of documents that are indexed. The parser can be run as each domain is crawled or can also be run once after all domains are crawled and the documents are downloaded.

7.2 Estimation of precision

Precision is the fraction of a search output that is relevant for a particular query [13]. This requires finding how many relevant and irrelevant documents are returned by the search engine for a given query. In the context of this project, precision can be defined as the fraction of number of relevant documents with exact phrase match to the total documents returned.

The number of relevant documents is determined by the results obtained from NextWord search since this approach returns only exact matches. If there are no results returned by NextWord search, the results obtained from Combined or Inverted search with highest refinement is used.

7.3 Response Time

The response time from the time the user submits the search phrase and the results displayed were less than 20 seconds. In most scenarios the results were returned in few seconds. Pagination of matching results allows the user to navigate to different pages of
the results and also reduces the display time. Only 5 URLs are displayed in each page and the user can select any page to move forward or backward to view results. In addition Next and Previous links are also provided to enhance the usability and results navigation of the search engine. When a phrase having more than 3 words was input, the NextWord index was seen to return results slower than the other two approaches. This is expected since the processing involved using NextWord index is complex compared to Inverted and Combined inverted NextWord index. However the query computation and processing may take longer if the size of the stored collection gets large and if more documents are indexed.

### 7.4 Analysis of Results

To analyze and compare the results search strings of different length and characteristics given below have been used. The following is a sample of some phrases that were used.

1. network security
2. campus activities
3. financial aid
4. multicultural affairs
5. library information
6. political science
7. radio astronomy
8. distributed computing
9. computer network security
10. business school faculty
11. campus parking information
12. distributed computer networks
13. list of scholarships
14. declaration of independence
15. Arthur lakes library
16. modeling of digital systems
17. gifted and talented program
18. Denver health medical center
19. college of nursing and health sciences
20. information for international students
21. academic success and advising center
22. Colorado center for community development
23. continuing education and professional studies
24. Colorado school of mines
25. reduce greenhouse gas emissions
26. democratic national convention
7.4.1 Comparison of response time between three approaches using Inverted Index.

Three levels of phrase matching were experimented using the Inverted Index.
1. Inverted: AnyWord - Retrieve documents that contain either of the words in the search phrase. Common words are excluded.
2. Inverted: AllWords - Retrieve documents that contain all the words in the search phrase occurring anywhere within the document. Common words are excluded.
3. Inverted: AllWordsInOrder – Retrieve the documents that contain all the words occurring within a specified tag (title, header, paragraph etc.,).

It is obvious that approach 1 will have the least precision and approach 3 will have the highest precision among the three approaches. The response times of the above three approaches for search phrases of lengths 2, 3 and 6 are shown in Figure 13 below.

![Response Time vs # words for inverted index](image)

**Figure 14. Response time using Inverted Index on 50000 documents.**

In addition to precision, the response time is also proportional to the number of matching documents available.

Any word: All documents that have the word ‘network’ or ‘security’ are retrieved. Since no exact phrase matching is done, processing involved is less. However there were many numbers of documents that had these words and the response time was high due to the volume of the documents. It is obvious that the precision obtained will be considerably low. In most cases the precision was 0.28.
All words anywhere in document: All documents that had both the words ‘network’ and ‘security’ anywhere in the document are retrieved. There is no exact phrase matching done, but all documents retrieved will have both the words occurring in the document. This reduced the number of documents retrieved but the response time was also considerably reduced. The average precision by the approach was about 0.45.

All words in same location in a document: All documents that had words ‘network’ and ‘security’ occurring within specified location or tags are retrieved. This approach retrieved documents that had both words occurring close to each other as opposed to being dispersed in different sections of the document. The numbers of documents returned were further limited in this approach. This improved the response time for phrases of different lengths in spite of a little processing overhead. The precision obtained by this approach was higher than 0.72.

Thus the results showed by Inverted Index show that by taking a greedy approach, the response time can be reduced.

7.4.2 Analysis of NextWord Index

The response time of NextWord Index for search phrases of different lengths is shown in Figure 14 below.

![Nextword search graph](image)

**Figure 15. Response time for NextWord index on 50,000 documents**

The NextWord index looks for exact match of phrase. For phrases that have more than 2 words, complex processing to match the exact phrase is done by breaking the input phrase into word pairs, retrieve the result for each pair and process the results to display
only the exact match. For example if the phrase is ‘declaration of independence’, the phrase is broken into two word pairs – ‘declaration of’, ‘independence’. The results of each word pair are retrieved independently along with the url, location and position for each occurrence. Based on the location and position information the results are further filtered and only exactly matching results are returned.

In the case of 2 word phrases, no complex processing was required. The response time is still high because the index size was high compared to Inverted Index. The precision was a 100% in the resulting documents.

In the case of 3 word phrases, additional one step of processing was required which increased the response time.

Response time for a phrase of 5 words further increased due to the processing overhead.

In the context of this project, results obtained from Inverted and Combined Index approaches are compared with the NextWord index, since the precision obtained using this index was very high.

7.4.3 Comparison of response time between two approaches using Combined Index.

Combined:AllWords: In this approach documents that had all words in the selected phrase occurring within the documents were retrieved. The order of the words was not considered but all the words in the selected phrase (word1 + common word + word2 + word3) should exist in the document.

Combined:AllWordsInOrder In the second approach, the search criteria was to retrieve only the documents that had all the words occurring in same sequence as the input phrase. The distances between the words were allowed to be only 2 or 3 words apart. For example if the search phrase was ‘network security’, this distance allowed phrases ‘network security’, ‘network and security’ to be retrieved.
From Figure 15 above, it can be seen that both approaches had same response time but is still less than the response time of the NextWord index. However the number of matching results returned by using the Combined Index and the NextWord index were almost the same as the number of the words in the phrase increased. In addition, the precision of the results were also very close to the NextWord index. Figures 12 and 13 below show the graphs depicting these results.

**7.4.4 Comparison of results obtained by each index type**

Figure 16 below shows the average number of search results obtained for different phrase lengths by each index type. The number of results returned for 2 word phrases are more than the number of results obtained for phrases of length 3 or more. The number of results returned by each index type is however similar for the different phrase lengths.
The precision obtained by Nextword Index in this project is almost 100% and is used as a baseline to compare the precision obtained by other two indexes. The fraction of first 10 urls returned by inverted and combined index types that were the same as that returned by Nextword Index was used to compute precision of Inverted and Combined indexes.
7.4.6 Comparison of Response Time between different phrase lengths and different document sizes.

The response time obtained for different phrase lengths as the number of stored documents increased were recorded and are shown in figure 18 below.

![Response Times vs # documents](image)

**Figure 19. Comparison of response time Vs # of documents**

It can be seen that the response times for 2 and 3 word phrases are high compared to the response times of 4 and 5 word phrases. This shows that the phrase length itself does not affect the response time for a phrase query.

7.4.7 Comparison of Response time with number of search results returned.

The number of search results returned for different phrase searches made on Needle Search engine were recorded. Based on these results, the response times when number of results were less than 10, between 10 and 50, between 50 and 100, between 100 and 500 and above 500 were observed. The observations have been shown in figure below.
7.4.8 Comparison of Response time with phrase structure.

Based on these results it can be concluded that the relatively smaller index size and the precision obtained using the Combined Inverted and NextWord index is the best option to
be used for phrase search and this approach has been adopted for the phrase query component of the Needle search engine.

7.4.9 Comparison of response time with PageRank

In addition to the comparison of Inverted, Nextword and Combined Index, comparison of precision and response times using three levels of page ranking was also performed.

1. Results obtained without considering page rank.
In this method, matching results for a phrase are obtained. The matching results are not sorted based on page rank. Due to this, the results may be displayed with highly relevant pages located anywhere in the result set.

2. Results obtained considering only page rank.
In this method, the matching results for a query are sorted based on only page rank. Thus the displayed results have matching documents that may be more relevant before the less relevant documents.

3. Results obtained considering both page rank and the location weight of the phrase.
In the third method, the location at which a search string occurs along with the page rank of the document is used to determine the display order of the results. For example if there are two documents with same page rank that have a search string. If the search string is found in the title of the first document and the same search string occurs in a heading in the second document, the first document takes precedence over the second document in the resulting display order. This ensures the highly relevant documents are displayed first and the less relevant documents are displayed later in the result set. This improves user experience and satisfaction since the most desirable results are displayed first. The user does not have to scroll through the entire result set.
Figure 22. Response time for different levels of ranking for 50,000 documents.

Summary of top 5 results obtained for each ranking level for search string ‘financial aid’.

1. No Page Rank
http://innovation.vast.uccs.edu/FAQ.html
http://innovation.uccs.edu/FAQ.html
http://eas.uccs.edu/cs/mscs.shtml
http://finaidse.uccs.edu

2. Only Page Rank
http://innovation.vast.uccs.edu/FAQ.html
http://innovation.uccs.edu/FAQ.html
http://eas.uccs.edu/cs/mscs.shtml
http://finaidse.uccs.edu

3. Page Rank and Location weight
http://finaidse.uccs.edu
http://innovation.vast.uccs.edu/FAQ.html
http://innovation.uccs.edu/FAQ.html
http://eas.uccs.edu/cs/mscs.shtml
From the results of above three methods and the results returned by a major search engine it can be seen that the results fetched using PageRank and location weight have the most relevant documents listed ahead of other documents with matching search string.

Results obtained by other search engines.

Sample results for search phrases obtained from major search engines have been shown below. The top ten results obtained from each search engine were observed and the common or relevant results have been shown below.

Search results for ‘nursing school’ in uccs.edu domain

Google

www.uccs.edu/~bethel/
web.uccs.edu/gradschl/certificates.html
web.uccs.edu/gradschl/programs.html

Yahoo

web.uccs.edu/orientation/.../Current/NURt_online_over.htm
uccs.edu/~ur/media/mediawatch/view_article.php?...
www.uccs.edu/~bethel/bsn.htm

MSN

web.uccs.edu/gradschl/programs.html
web.uccs.edu/gradschl/certificates.html
www.uccs.edu/~bethel/bsn.htm

Ask.com

www.uccs.edu/~bethel/
web.uccs.edu/gradschl/certificates.html

Needle

http://web.uccs.edu/bethel
http://web.uccs.edu/squalls/currentprojects.html

Search results for search query ‘campus placement’ in colostate.edu domain

Google

www.math.colostate.edu/placement/placement.
business.cudenver.edu/Disciplines/HealthAdmin/MBAHealth...

Needle

http://business.cudenver.edu/Disciplines/Management/Faculty/Gifford.html
http://thunder1.cudenver.edu/campussafety/personal.html

Results for search string ‘programmed cell death’ in Colorado.edu domain

Google Time (0.23 secs)

mcdb.colorado.edu/courses/4426/Ding/Conradt%20and%20Xue%202005.pdf
mcdb.colorado.edu/labs1/xue/MCDB4426/Lecture%2004.ppt
mcdb.colorado.edu/faculty/researcherobjects/Ding-Xue

Yahoo (Time 0.7 secs)

mcdb.colorado.edu/labs1/xue/Research.htm
www.colorado.edu/UCB/AcademicAffairs/ArtsSciences/MCDB/MCDB4620/15.pdf
mcdb.colorado.edu/labs1/xue/Activation.htm

MSN

mcdb.colorado.edu/courses/4426/Ding/Conradt%20and%20Xue%202005.pdf
mcdb.colorado.edu/labs1/xue/Research.htm
mcdb.colorado.edu/faculty/researcherobjects/Ding-Xue

ASK.com

mcdb.colorado.edu/courses/4426/Ding/Conradt%20and%20Xue
mcdb.colorado.edu/labs1/xue/MCDB4426/Lecture%2004.ppt
mcdb.colorado.edu/faculty/facultypublications/XueDPublications

Needle (Time 14 secs)

http://mcdb.colorado.edu/faculty/researcherobjects/Ding-Xue
http://mcdb.colorado.edu/postdoctoral/researcherobjects/Christine%20Ondzighi-Assoume
http://mcdb.colorado.edu/mcdb/faculty/facultypublications/BoswellRPublications
The number of results, response time and precision obtained from each search engine is given in Table below. Precision was computed based on the most relevant documents that were displayed in top 10 results. 20 phrases of length 2-5 were searched in different search engines and results were obtained.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Response Time</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>0.95</td>
<td>0.2 secs</td>
<td>0.95</td>
</tr>
<tr>
<td>Yahoo</td>
<td>0.9</td>
<td>0.5 secs</td>
<td>0.95</td>
</tr>
<tr>
<td>MSN</td>
<td>0.75</td>
<td>Not displayed</td>
<td>0.8</td>
</tr>
<tr>
<td>Ask</td>
<td>0.75</td>
<td>Not displayed</td>
<td>0.85</td>
</tr>
<tr>
<td>Needle</td>
<td>0.85</td>
<td>22 secs</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table 5. Comparison of results obtained from other search engines.

The response time and precision obtained by major search engines cannot be compared with the results obtained by Needle search engine due to the magnitude of indexed documents, highly complex processing capabilities and distributed nature of such systems. However the results obtained by Needle search engine also had good precision as well as acceptable response time.

The following screen shot shows the Needle front end.

Figure 23. Screenshot of Needle search engine front end.
8. Integration with Needle Search Engine

8.1 Needle Crawler
The indexes required for the phrase search component have been built based on the documents downloaded by the Needle Crawler[9]. The same database tables required by the crawler have been used. The Needle crawler was run on uccs.edu, cudenver.edu, Colorado.edu, mines.edu and colostate.edu and a total of 1,160,000 urls were crawled.

8.2 Page Ranking
The page rank for the downloaded documents have been computed using the cluster rank algorithm[8]. Page Rank is a way to measure the importance of a Web page by counting how many other pages link to it, as well as how important those page themselves are. Finding the page rank for the Web pages is very crucial for any successful search engine. The ClusterRank algorithm was run for the crawled urls. Time taken for the pageRank program to compute the pagerank fro 1,160,000 documents was about 8 hours.

9. Problems faced

9.1 Different file types and sizes.
Web sites may contain several file types such as html, php, doc, txt, xml etc. For the phrase search component only text documents are needed. The crawler was configured to download text documents to a separate directory. The crawler uses LWP::UserAgent Object which dispatches a get request on a url. Thus all web pages that are crawled and that can be downloaded are converted to html documents. Therefore a HTML parser alone is sufficient to parse the downloaded documents and create the necessary indexes for the phrase search feature. If documents are obtained using other protocols such as ftp, then different parser modules corresponding to different file types will be needed. In some cases the crawled was unable to download a document due to high file size. In such cases the crawler had to be restarted to ignore the current url and start from the next url.

9.2 Poorly formatted web documents.
All web documents are not created in the same manner. There are several html tags. In addition each document also has a number of tags such as <meta>, <anchor text> etc. The tags are further embedded or nested within other tags. There are different ways in which a tag is specified such as <para>, <p>, <paragraph>. Extracting data from web pages is also difficult due to the fact that many web pages do not conform to valid HTML syntax. Thus a parser that reads the Web documents has to be robust enough to handle a variety of documents. The parser had to be able to handle a variety of possible errors like typos, abbreviations in html tags, nested tags, non-ASCII characters etc. A simple parser that can be configured to parse only contents within pre-defined tags was developed. HTML::TokeParser module is used get each html tag in a given document. HTML::Entities Perl module is used to get the content within each html tag that will be parsed. Each html tag is also assigned a weight that will be used to determine the display order of search results.
9.3 Disk Space

To compare the three different approaches for phrase querying, three different indexes had to be built on the same data. A single parser that can create Inverted, NextWord and Combined inverted NextWord index was built to save time on running three different parsers and create the different indexes at the same time. As the number of documents indexed increased the number of entries in Nextword index table was more than the number of entries in Inverted and Combined Nextword index tables. The Nextword Index had 11,076,099 entries. The disk space used by the data and index files for the NextWord Index table was 1.1 GB while the disk space used by the Combined Inverted Index data and index files was 700 MB. The

After obtaining results from using NextWord Index for comparison purposes, the NextWord Index table can be deleted. This will allow more documents to be indexed.

9.4 Get a fresh repository

The Crawler was built and run in 2006 and many Web pages has been removed or modified. Due to this the downloaded URLs had become stale. A fresh repository of documents had to be built and this required the crawler to be run on several domains. This was a time consuming process. To download documents from the different domains, the crawler was run for about 10 days, for one domain at a time. In addition there were also issues with the crawler crashing at some URLs. The crawler database uses tables that have fields with auto-increment feature, the stores a url, the number of urls linking to the document etc. If the crawler crashed due to some reason, the tables had to be dropped and recreated again before starting the crawler again. Thus for each domain the crawler had to be run separately and the results for each domain had to be merged with the existing URL table.

9.5 Compute PageRank

The PageRank for the crawled URLs were computed using the PageRankByCluster algorithm. As the number of urls crawled increased the time taken to compute PageRank also increased. The final run of the PageRankByCluster algorithm on all the crawled URLs took about 8 hours to compute the page rank.

9.6 Keeping the search engine up to date.

Web documents change very often and information is added constantly. Thus a common issue faced by any search engine is to maintain a fresh repository of indexes. This implies the crawler needs to be run periodically to obtain the latest Web pages, page rank and indexes for the new collection computed on the recent documents. Thus a process by
which these updates can be made periodically needs to be identified. This process may not be possible to be fully automated since the updates may be done on different domain at different time frames. Thus a person who maintains the search engine must be fully aware of the how each of the program i.e the crawler, parser and pagerank can be run and the database updates that need to be done if the updates are made for each domain at different time frames.

The sequence of the steps that needs to be done is documented at the end of this document.

9.7 Calendar URLs

The UCCS domain has several calendar links. Out of the 50,000 documents that were parsed, 37,143 were calendar links. Details of how popular search engines like Google and Yahoo solve this problem have not been published. Experimental results done on Google showed calendar URLs were also returned for some searches. The calendar links also had important information in some cases. For example a search for phrase ‘financial aid’ also returned the calendar URL that had the first and last dates a student may apply for financial aid. Therefore the calendar URLs could not be completely eliminated. However due to a large number of calendar urls available in the stored document collection the results of a search often had identical URLs.

A large number of calendar urls were found only in UCCS domain. Other domains did not have calendar urls. A database query for urls with string ‘calendar’ or ‘webcal’ yielded only 3 results in other domains.

9.8 Providing a synopsis of URLs that have matching results.

For the results of a phrase query, displaying the URL itself or a few words around the search phrase in the document did not provide any information on the what the web document about. It was also noticed that the title of a document in most cases was the same information seen the URL or very generic and did not convey any information to the user. In order to provide a synopsis for each URL, the content in the title, headers were combined and added to each entry in the URL table. Thus the results displayed both the URL and information on the web page itself. However storing a the synopsis for each url increased the data size of the URL table.

9.9 Results Comparison

In order to compare three approaches, three separate implementations were needed. Since the indexes were different, the results could not be compared as apples to apples but only in a broad fashion.
10. Conclusion

All the three approaches discussed Inverted index, NextWord index and Combined inverted NextWord index have their own advantages and disadvantages.

The Inverted index is particularly suitable for single word searches and searches that have multiple words separated by logical OR operator. For a given phrase, the resulting pages returned by inverted index may not have exactly same phrase, but the results are still usable for the user.

The NextWord index is suitable for searches where the user expects results that contain exact match for an input phrase. This index provides the highest precision for a phrase query compared to the other methods. The index size however inflates as the number and size of documents parsed become more and the processing overhead also increases as the index size increases.

The Combined Inverted NextWord Index combines the advantages of both the inverted and NextWord indexes. This approach is particularly suitable for phrase searches that involve a common word. If the search phrase does not have a common word then query evaluation is done similar to an inverted index.

The Needle Search Engine has a user friendly front end that provides options to the user to select text/phrase search and image search. When only one word is entered as search phrase, the inverted index is used to find the matching results. When a search text having 2 or more words is entered, the input search text is first parsed to determine if there is a common word in the search text. If the search text has a common word, both the inverted and combined inverted index are used to process the query and find matching results. If the input search text does not contain any common word, the inverted index alone is used. When a user enters a long phrase or a sentence, the common words in the input text are removed. Only the first 4 words are included in the search and the matching results are found and displayed to the user.

11. Future Work

The success of a good search engine depends on the number of the documents that are indexed as well as the precision of the documents that are returned. As the number of indexed documents gets larger, the indexes as well as query processing time will also increase proportionally. To reduce the processing and response time in such cases, distributed storage and processing methods need to be explored.

Today’s relational databases provide data warehouse version of their engine which has capabilities to distribute data using hashing methods based on the primary index of the table. These engines perform distributed query processing and fragment elimination.
techniques to bring in faster response time irrespective of the amount of data; in our case number of URL’s crawled. By installing data warehouse version of database engine & distributed data storage techniques, acceptable response time and performance objectives can be met without any change to the underlying application layer.

Other method will be to crawl and store documents in separate servers. When a user enters a search text, the search text will be communicated to each secondary server by the master server. The processing to find matching results is locally done by each secondary server. Finally all secondary servers send the results to the master server which combines and displays the results. This approach can be used when different domains are crawled and indexed. Each server can store the documents and tables corresponding to one domain. A second method would be to store different tables in different servers for example, the inverted index can be stored in one server and combined NextWord index stored in a different server. Based on the type of the search text (single word, phrase having common word, phrase that does not have a common word) the master server can send the query to the right server. This approach will be helpful when the search engine is widely used and receives many queries at a given time.

Keyword searches have a tough time distinguishing between words that are spelled the same way, but mean something different (i.e. hard cider, a hard stone, a hard exam, and the hard drive on your computer). This often results in hits that are completely irrelevant to your query. There are also issues that may arise due to stemming, singular and plural words, verb tenses. Search engines also cannot return hits on keywords that mean the same, but are not actually entered in your query. A query on heart disease would not return a document that used the word "cardiac" instead of "heart." “Intelligent” methods that can lookup a dictionary or thesaurus to cluster and retrieve all relevant documents will produce more satisfying results to the user.
12. References.


[9]. Patankar, Sonali: Needle Crawler: A Large Scale Crawler for University Domains, Master’s project, UCCS Computer Science Department, 2006.


Appendix A Database Scripts

All the tables required for the crawler and page rank component will have to be created first. The database scripts for the corresponding components have been mentioned in the corresponding project reports [8,9].

The tables required for the Inverted and Combined Index can be created using the following scripts.

```sql
CREATE TABLE `inverted` (
    `keyword_id` int(10) unsigned NOT NULL default '0',
    `url_id` int(10) unsigned NOT NULL default '0',
    `position` int(10) unsigned default NULL,
    `location_id` smallint(5) unsigned NOT NULL default '0',
    `update_date` date NOT NULL default '0000-00-00',
    `frequency` mediumint(8) unsigned default NULL,
    `location_weight` float unsigned zerofill default NULL,
    UNIQUE KEY `keyword_id` (`keyword_id` ,`url_id` ,`location_id`),
    KEY `url_id` (`url_id` ),
    KEY `location_id` (`location_id` )
)

CREATE TABLE `Combined` (
    `keyword_id` int(10) unsigned NOT NULL default '0',
    `url_id` int(10) unsigned NOT NULL default '0',
    `NextWord_id` int(10) unsigned default NULL,
    `position` int(10) unsigned default NULL,
    `location_id` smallint(5) unsigned NOT NULL default '0',
    `update_date` date NOT NULL default '0000-00-00',
    `frequency` mediumint(8) unsigned default NULL,
    `location_weight` float unsigned zerofill default NULL,
    UNIQUE KEY `keyword_id` (`keyword_id` ,`url_id` ,`location_id` ,`NextWord_id`),
    KEY `url_id` (`url_id` ),
    KEY `location_id` (`location_id` ),
    KEY `NextWord_id` (`NextWord_id` )
)
```

Run the following query to update the pagerank for each url in the URL table. This is done after the pagerank module is executed to compute the page rank and the page rank is available in PageRankByCluster table.
update URL, PageRankByCluster SET URL.c_prc = PageRankByCluster.c_prc WHERE URL.url_id = PageRankByCluster.url_id;

Run the following query to calculate the location weight for each entry in the Inverted and Combined Index after the page rank for each document is updated in the URL table.

update KeyWordWork_Combined, PageLocation, Down_URL SET KeyWordWork_Combined.location_weight = Down_URL.c_prc * PageLocation.weight WHERE KeyWordWork_Combined.url_id = Down_URL.url_id AND KeyWordWork_Combined.location_id = PageLocation.id;

update KeyWordWork_inverted, PageLocation, Down_URL SET KeyWordWork_inverted.location_weight = Down_URL.c_prc * PageLocation.weight WHERE KeyWordWork_inverted.url_id = Down_URL.url_id AND KeyWordWork_inverted.location_id = PageLocation.id;
Appendix B User Manual

1. Configuration:
Almost every aspect of the Needle search engine is controlled by the Config.xml, which has six major sections: DB, crawler, textparser, ImageProcessing, RankingSystem and FrontEnd. All settings are commented with notes to explain their purposes.

2. Create the database tables required for the crawler and page rank modules.
To run the crawler in the background on a linux machine the following command can be executed.

   nohup perl crawlenov08.pl &

3. Create the tables required for inverted, NextWord and Combined Indexes.
   Run the text parser using the command:
   nohup perl text_parser_v6_all.pl &

4. Run the page rank modules
   nohup perl clustering.pl &
   nohup perl clusterrank.pl &

5. The perl program for the Needle Front End is located in the following directory
~pselvam/cgi-bin/CombinedIndex/search_list_match1.pl
Appendix C Using the search engine

The Needle search engine can be accessed at the following URL.

http://128.198.144.19/~pselvam/cgi-bin/CombinedIndex/search.pl

This requires UCCS VPN connectivity to be established.

The Needle search engine is also available on the internet from URL

http://pikespeak.uccs.edu/~priya/cgi-bin/search.pl

The front end submits the search to a database on a different server at 128.198.144.19.
Appendix D

User Survey

The criteria for assessing the performance of information retrieval systems are:
1) Coverage, 2) Recall, 3) Precision, 4) Response time, 5) User effort, and 6) Form of output [16, 17]. It is difficult to compute Recall for a web search engine and hence it has been omitted from this evaluation. Other factors that were used to evaluate the search engine were search capability, retrieval performance, display format, user effort and robustness. Based on these factors, a survey questionnaire was distributed and results were collected from a sample of people representing diverse fields.

The sample questionnaire has been shown in Figure below.

Survey Questions to evaluate Needle Phrase Search Component

Name :

Gender :

Background :

Rating 1 - Very Good 2 - Good 3 -Meets 4 - Not Satisfactory 5 - Not Acceptable 6 - Not Applicable

<table>
<thead>
<tr>
<th>SI No</th>
<th>Search Text</th>
<th>Number of results</th>
<th>Result Accuracy</th>
<th>Response Time</th>
<th>Results satisfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Did you find the search engine easy to use? Yes/No

How would you rate the overall results? Relevant/Not Relevant

How would you rate overall performance? Very Good/Good/Poor

Did you like the format in which search results are displayed? Yes/No

What other features would you like to be integrated with this search engine?

How many times did the search engine return an exception?

Is the search engine design aesthetically appealing?

How many pages in the results did you navigate?

Suggestions/Comments

Each person was asked to perform 10 different searches and the results were collected. Based on the results obtained from 15 people average response, result accuracy, user satisfaction, ease of use have been computed and shown below.

How would you rate the overall results? Relevant

How would you rate overall performance? Meets

Overall Response Time Meets

Did you like the format in which search results are displayed? Yes

Did you find the search engine easy to use? Yes

Suggestions/Comments:

Survey Results:

• 155 phrases were searched
• Overall Results – 2.9
• Overall performance – 2.5
• Overall Response Time – 3.2
For very specific searches, there were limited/no results. Users who had very good domain knowledge and did specific searches suggested more [ages indexed so that more results are displayed.

Users who made generic searches such as library information, business school, computer science etc. were satisfied with the results.

The general idea was the search engine met the objective of the project.
Appendix E

Analysis of Results obtained for different phrase lengths.

Figure 24. Response Time Vs # Documents

Figure 24 shows the response time obtained for different phrase lengths as the number of documents increased. It can be seen that the response time reduced as the length of the phrase increased.

Findings:
There were some words appearing over 10,000 times in the documents parsed. These words were used in the two, three, four & five word search strings used during the tests used for building the above graph. A slow response time was noticed whenever one of the words which appear over 10,000 times in the parsed documents was used as part of the search string irrespective of whether it is two, three, four or 5 word searches. Since these words were used only in few 4 or 5 word search strings, better response time was noticed.

It was also observed that the response time for the same search phrase performed at different time of the day produced different results. This may be due to the load on the server or network when the search was performed.

Explanation:
Analysis of the phrases used for obtaining results showed the response times for 4 and 5 word phrases were less than the response time for phrases of length 2 and 3 words. It was
observed that there were 122 words in the Inverted Index that occurred 5000 or more times in the index. 33 words had more than 10,000 entries in the Inverted Index.

```sql
mysql> select B.word from KeyWordWork_inverted A, Dictionary B where A.keyword_id = B.id group by B.word having count(*) > 10000;
```

arts
business
calendar
center
college
colorado
course
denver
department
event
faculty
forums
graduate
how
if
information
may
more
new
other
page
program
research
school
science
sciences
state
student
students
uccs
university
vision
web

Search phrases that were used to obtain response times have been listed below.

2 word phrases

computer science
computer vision
management studies
global warming
environmental justice
fiber optics
distance education
soil formation
bursar office
faculty information
gene sequence
design patterns
computer architecture
campus activities
financial aid
political science
undergraduate courses
distributed computing
multicultural affairs
library information

3 word phrases

programmed cell death
condensed matter research
computer science department
computer network security
chemistry and geochemistry
declaration of independence
arthur lakes library
environmental studies program
list of scholarships
analysis and design
rocky mountain history
kraemer family library
student success center
democratic national convention
environmental studies program
project management course
business school faculty
campus parking information
energy conversion laboratory
beetle eradication program

4 word phrases

modeling of digital systems
gifted and talented program
denver health medical center
information for international students
colorado school of mines
reduce greenhouse gas emissions
object oriented analysis design
folk arts in colorado
history of colorado river
college of applied science
distributed systems architecture testbed
SERL configuration management papers
naturally occurring cellular process
national renewable energy resource
boulder environmental studies program
biological survey in colorado
leadership in environmental justice
colorado fuel cell center
colorado denver business school

5 word phrases

college of nursing and health
colorado center for community development
uccs college of applied science
anatomical therapeutic chemical classification system
testbed for autonomous agents in space
programmed cell death is regulated
website development practice and technologies
national oceanic and atmospheric administration
graduate studies in chemical physics
colorado japanese beetle eradication program
CU Boulder environmental studies program
liberal arts alumni and friends
identify career goals and interests
research solutions to global warming
consortium for agricultural soil mitigation
engines and energy conversion laboratory
energy conversion and energy distribution
rabies research at fort collins
object oriented analysis and design
mission of the student chapter
speech language and hearing science

9 of the 20, 2 word phrases had one of these words that occurred more than 5000 times in the index. The following table shows the number of phrases that had one of these words that occurred most number of times as the first word in the phrase.

<table>
<thead>
<tr>
<th>Phrase length</th>
<th># phrases that had a frequent word in 20 phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
When a phrase had one of these most frequently occurring words as the first word, the response time was seen to be high since there were many hits from the index. Since 2 and 3 word phrases had more number of phrases having a frequently occurring word as the first word, the response times were higher than the 4 and 5 word phrases.

A comparison of the actual time that was taken to execute the sql query and the time taken to fetch and display the results have been shown in the figure below. There was no significant difference between both the times.

Response Time for major search engines

A comparison of results obtained from other search engines is shown in Table 5. Table 6 shows the reported and actual response times obtained from Google and Yahoo. The actual response times were computed using the wget method of Perl.

<table>
<thead>
<tr>
<th></th>
<th>Google</th>
<th>Yahoo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reported</td>
<td>Actual</td>
</tr>
<tr>
<td>0.18</td>
<td>0.77</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Loosely formatted HTML Documents
Parser code used for this project references all commonly used HTML tags in a config file. This file is used by the HTML::TokeParser module to get each start tag from the parsed documents. It was noticed roughly 7% of the documents had one or more tags that were not defined in the config file. Documents that had start tags corresponding to source code in C, Java, stylesheets, java script code were not parsed. Examples of such tags are <mkdir>, <copy>, <fileset>, <remove>, <st1:place> etc. Such tags were ignored by the parser, since these tags did not really have any valid information.

**Analysis of phrases used by users in survey.**

There were a total of 155 search phrases used by different users. The phrases were analyzed and the observation has been recorded.

<table>
<thead>
<tr>
<th># words</th>
<th>1 word</th>
<th>2 words</th>
<th>3 and more words</th>
</tr>
</thead>
<tbody>
<tr>
<td># phrases</td>
<td>62</td>
<td>70</td>
<td>23</td>
</tr>
<tr>
<td>% phrase</td>
<td>40%</td>
<td>45%</td>
<td>15%</td>
</tr>
<tr>
<td># common words</td>
<td>0</td>
<td>0</td>
<td>18</td>
</tr>
</tbody>
</table>

40% of the search phrase had 1 word, 45% had 2 words and 15% had 3 or more words. 27/155 search phrases had the words that occurred most frequently in the inverted index such as ‘university’, ‘colorado’, ‘department’. 18/23 phrases of length 3 and more has a common word such as ‘of’, ‘in’ etc.