**CIS6905 - Individual Study Report**

**TOPIC: Information Retrieval**

**Supervisor: Dr. José A.B. Fortes**

**Lakshmanan Velusamy**

**UFID: 481010206**

**ACKNOWLEDGEMENTS**

Thanks to **Dr. José A.B. Fortes** for recommending this interesting topic and also mentoring this course work.

Thanks to **Dr. Andrea Matsunaga** and **Dr. Mauricio Tsugawa** of ACIS lab for their guidance, time and support regarding this course work and also for their help in setting up data sets for the course projects and experiments.

Thanks to fellow graduate students **Kyuho Jeong** and **Sarfaraz Ahmed Soomro** for their presentations on various topics and tools regarding this course.

1. **INTRODUCTION / MOTIVATION**

In this 21st century most of us are getting information from internet through a computing device rather than from the people. Gone are the times when people used human travel agents to book their travel. During the late 1990’s and early 2000s, relentless optimization of information retrieval effectiveness has driven web search engines to new quality levels where most people are satisﬁed most of the time, and web search has become a standard and often preferred source of information ﬁnding. For example, the 2004 Pew Internet Survey (Fallows 2004) found that “92% of Internet users say the Internet is a good place to go for getting everyday information.” Information retrieval is the science behind answering our day to day information needs such as finding the shortest route to our destinations, current traffic data in a particular highway, the arrival time of the next bus at the nearest bus stop, tomorrow’s weather forecast etc. Most of the commercial and open source search engines are able to provide high-quality results within sub-second response times for hundreds of millions of searches a day over billions of web pages.

These reasons motivated me to study, understand and learn the concepts, ideas, data structures and algorithms applied behind the scenes to perform efficient information retrieval under the supervision of Dr. José A.B. Fortes at Advanced Computing and Information Systems laboratory, University of Florida.

This course work consists of four different milestones. The course work is organized as follows:

1. Learning the concepts of information retrieval and understanding the components of an information retrieval system, validating / evaluating the effectiveness of an information retrieval system (from the text book [1]).
2. Understanding the internals of existing popular information retrieval systems such as [Lucene](http://lucene.apache.org/), [Solr](http://lucene.apache.org/solr/), [OpenSearchServer](http://www.open-search-server.com/) and [Sphinx](http://sphinxsearch.com/) and comparing the ideas applied in them with the concepts learned in the text.
3. Implementing an Information Retrieval system for the searching of High Performance Computing 2011 conference proceedings collection using Lucene Information retrieval library, Solr and Sphinx search server utilities.
4. Index the biodiversity data collection (which us represented using [Darwin Core Archives](http://www.gbif.org/informatics/standards-and-tools/publishing-data/data-standards/darwin-core-archives/) format) on search engine based information retrieval solutions, feed the data set into a Data Base based information retrieval solution and compare the results in terms of qualitative and quantitative metrics such as precision, recall and response time for the following kind of queries:
   1. String query / Free Text queries
   2. Range queries
   3. Boolean (AND, OR & NOT) queries
   4. Missing information / IS NULL queries

A brief summary of each of these activities and key results are documented in the upcoming sections.

1. **INFORMATION RETRIEVAL CONCEPTS**

The following concepts have been learned with the help of the text book [1] and also from World Wide Web.

**2.1 Introduction to information retrieval**

The term Information retrieval could be loosely defined as “Finding material (usually documents) of an unstructured nature (usually text) that satisﬁes information need from within large collections (usually stored on computers)”. The essential components of an Information retrieval system are as shown the figure 1.

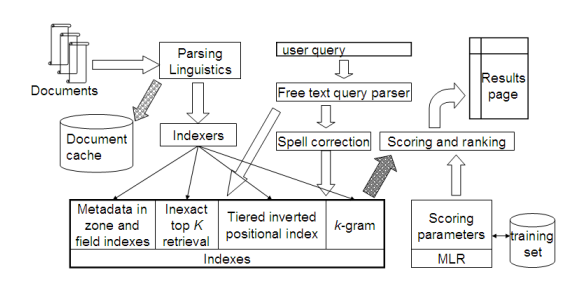


Fig.1. A complete Information Retrieval System and its components

**2.2 Information Retrieval models**

The information need is expressed in terms of query. A query is nothing but a collection of terms. Documents and term relationship can be represented using different models such as Boolean retrieval model and inverted index model. Boolean retrieval model captures the presence / absence of terms in documents in a Boolean matrix, which is sparse. Inverted index maintains a dictionary of terms and per term postings list which is a linked list of document IDs that contain the particular term. Posting lists can be effectively traversed in sub-linear time using the ideas such as skip pointers.

**2.3 Issues related to parsing the documents and query**

A search system shall be given the document set / data source it may crawl the document sources such as World Wide Web to index them. The document shall be represented in different formats such as RTF, PDF, and XML etc. Documents shall be written in different languages. Linguistics brings us lot of issues in information retrieval. How to break the document into terms? How to find a word when query term is represented in caps letter and the document contains that word in small letter? Can we convert everything to small letters? If so, How to find the appropriate documents if the user is searching for CAT (Common Admission Test) by avoiding documents related to cat?. How to process a query synonymous query terms even though the document collection contains the information need but expressed in different (synonymous) terms? How to handle typos? Each search engine uses language specific heuristics to handle all the special cases and exceptions in each language.

**2.4 Compressing the Index**

Avoiding the common words (also called as ‘stop words’), prepositions, and articles shall end up compressing the indexes in a great deal. There are several other techniques to represent dictionary and to encode posting list such as Variable byte codes, γ codes etc to achieve compression.

**2.5 Handling larger query loads and document using distributed indexes and query processing**

Index size is directly proportional to the size of the data collection. When more users start accessing a search engine a single machine and single index becomes a bottleneck. The concept of distributed indexes and map-reduce programming paradigm provides methods to parallely build distributed indexes and also facilitate distributed and parallel query processing to achieve faster searching.

**2.6 Scoring models to measure the relevance of documents to the query**

On top of retrieving the documents which contain the query terms, the users would be interested in getting the top K results and the results be sorted in terms of relevance. A search engine should be capable of scoring documents based on query.

**2.6.1 Statistical model**

A straight forward approach would be to use the statistics of terms and rank the documents which have more occurrences of query terms and most number of query terms and the documents which contain the query terms in a closer proximity. This is the idea behind the statistic measures such as [tf-idf](http://en.wikipedia.org/wiki/Tf*idf) (term frequency – Inverse documents frequency).

**2.6.2 Vector Space Model**

A more advanced approach is to treat documents as a vector in N dimensional space and also represent the query terms as a vector and compute the relevance between documents and query in terms of the angle between vectors. Computing the cosine score between all the documents and the query at the query time could result in slower response time. An information retrieval system should employ several heuristics to figure out the champion lists which would be potential candidates for top K documents.

**2.7 Query processing and User Interface**

A query should also be processed and spell correction has to be applied for possible typos for an improved user experience. The algorithmic techniques such as edit distance along with k-grams, phonetic based spell correction algorithms such as *soundex* are employed to correct spell mistakes. A search engine should support various querying methods such as prefix queries, range queries, wild card queries, etc. The underlying index should be designed to accommodate these query types. Crucial decisions in designing the data structures can make or break the system.

It is important to have an intuitive, effective and user friendly User interface to receive the query from the user and to effectively display the results to the user. The system should cover the spectrum of user types from naïve users (who are least familiar with computer who mostly prefers free text queries) to more advanced law firm search agents (who can be efficient in querying using various operators).

**2.7.1 Highlighting the results (Static summary / Dynamic Excerpts)**

The user would like to know why a particular document is returned for his query and why it is rated high over others. It would be nice to show an excerpt of the documents, which could be statically generated summary or it could be a dynamically generated that conveys the presence of query terms in a closer proximity.

**2.8 Testing and Evaluation of Information retrieval system**

It is crucial to evaluate the information retrieval system to find its effectiveness. Rate of repeated users is an indirect metric to measure the user happiness. There are several test sets available such as Reuters Corpus Volume (RCV), TREC (Text REtrieval Conference) collection with the sample queries and expected results and their relevance.

**2.8.1 Precision, Recall, Relevance**

There are standard metrics defined in literature to measure the search results. Precision and relevance measure the accuracy of the results i.e., % of documents retrieved which are relevant. Recall measures the % of correct documents retrieved from the collection.

**2.8.2 A/B testing**

A/B testing is a systematic technique to measure the effectiveness of parameter / configuration changes to measure the impact. The idea is to expose randomly expose only a small set of users to the modified system and changing only one parameter to measure its effect. For example, if the ranking function is changed how many users are clicking the top results etc?

With this background about science of information retrieval, existing information retrieval tools have been studied, information retrieval systems have been developed using those tools and also experiments have been conducted to compare the search engine based information retrieval and data base based information retrieval.

1. **STUDY OF EXISTING INFORMATION RETRIEVAL SYSTEMS**

The following information retrieval utilities have been studied. A brief summary of them is included in this report. For a detailed presentation on this please visit the course [forum page](https://www.acis.ufl.edu/index.php?session=c74e54b6da75dafdea1dd48f26004a82&threads=yes&archive=no&viewdocs=no&farchive=no&myforums=yes&l=44&forumid=736). Please feel free to [contact me](mailto:lakshmanan.v@ufl.edu) if you can’t access it.

**3.1 Lucene**

Lucene is a powerful and popular Java Information Retrieval Library. It makes it easy to add full-text search capability to an application or website. Internally it follows inverted index representation. Lucene has been ported to many other programming languages, including Perl, Python, C++, and .NET. Lucene's powerful APIs focus mainly on text indexing and searching. It can be used to build search capabilities for applications such as e-mail clients, mailing lists, Web searches, database search, etc. Web sites like Wikipedia, jGuru, and LinkedIn have been powered by Lucene.

The following block diagram in Figure 2 shows the components involved in typical Information retrieval systems and the components covered by lucene (shaded entities). Document crawlers are not built-into lucene by default. The developers can add crawling feature to their applications from other open source projects such as [Nutch](http://nutch.apache.org/), [Grub](http://www.grub.org), [and Aperture](http://aperture.sourceforge.net) etc. Lucene can support free text, Boolean, range, wildcard, phrase, prefix and fuzzy queries. Lucene has filters, processors and analyzers build to process documents from different languages, thanks to the open source community. It also supports both statistical (based on tf-idf) and vector space scoring models.

Lucene has many features. It:

* Has a powerful, accurate, and efficient search algorithm.
* Calculates a score for each document that matches a given query and returns the most relevant documents ranked by the scores.
* Supports many powerful query types, such as PhraseQuery, WildcardQuery, RangeQuery, FuzzyQuery, BooleanQuery, and more.
* Supports parsing of human-entered rich query expressions.
* Allows users to extend the searching behavior using custom sorting, filtering, and query expression parsing.
* Uses a file-based locking mechanism to prevent concurrent index modifications.
* Allows searching and indexing simultaneously.

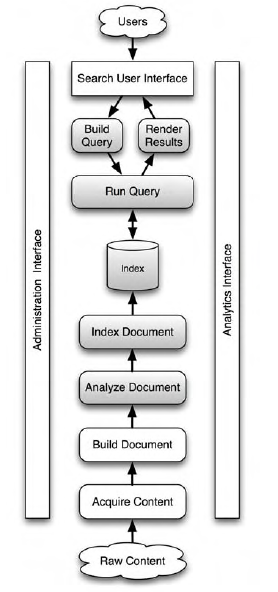


Fig.2. Lucene components

**3.1.1 Lucene Index analysis with Luke**

Tools used: Java, Luke lucene index analyzer V0.9.9, Java Lucene library, Eclipse, TPC-H data generator.

A sample data set is generated using data base benchmark standard [tpc-h](http://www.tpc.org/tpch/) data generator (lineitem relation, which has 6 Million plus records) and indexed. The index structure has been analyzed using [Luke](http://code.google.com/p/luke/) – lucene index analyzer toolbox. The indexing is as shown in the Figure 3 and the analysis of lucene indexes using Luke is as shown in figure 4 & 5.

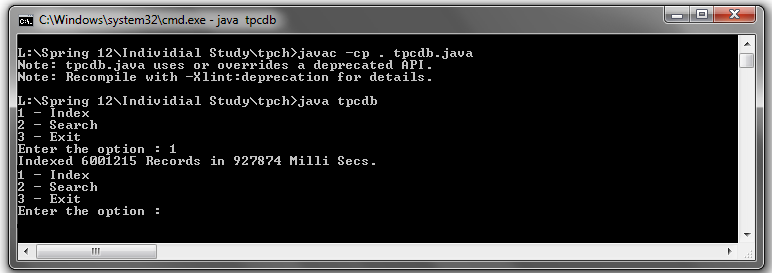


Fig.3. Index creation on TPC-H data set to analyze lucene index structure

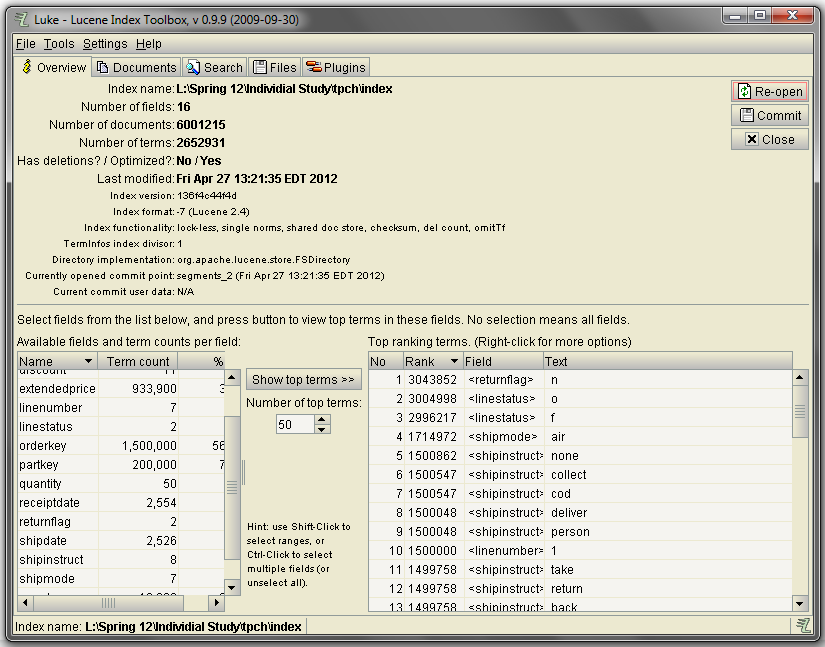


Fig.4. Index analysis using Luke (Lucene Index analyzer), Document structure and field statistics.

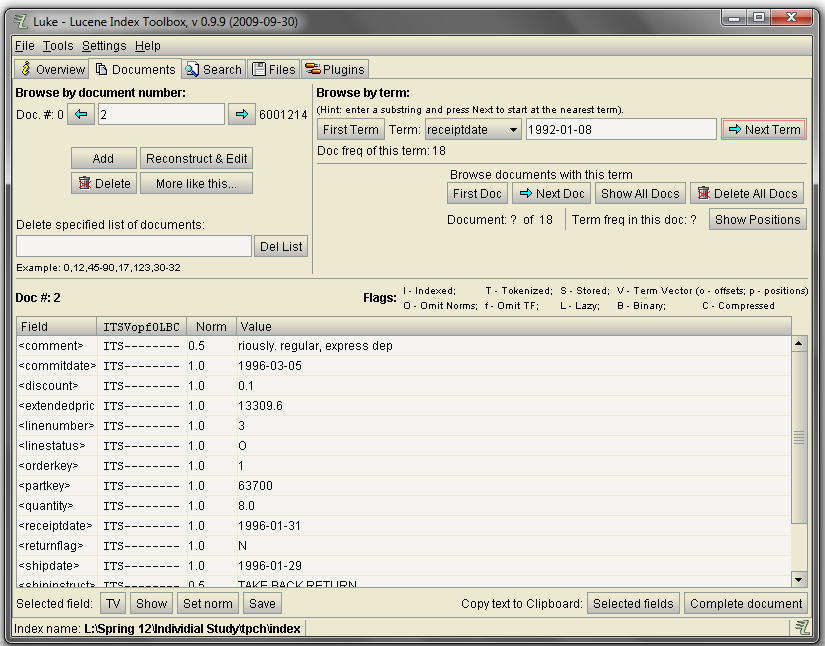


Fig.5. Index analysis using Luke (Lucene Index analyzer), View of an Indexed document.

**3.2 Sphinx**

Sphinx is a full-text search engine, publicly distributed under GPL version 2. Technically, Sphinx is a standalone software package provides fast and relevant full-text search functionality to client applications. It was specially designed to integrate well with SQL databases storing the data, and to be easily accessed by scripting languages. However, Sphinx does not depend on nor require any specific database to function. Sphinx also uses inverted index representation to represent indexes.

Sphinx has a stack of features, including:

* It can index any data that can be represented as a string.
* It can index the same data in different ways. With multiple indexes, each tuned for a specific purpose, the developers can choose the most appropriate index to optimize search results.
* It can associate attributes with each piece of indexed data. Users can then use one or more of the attributes to further filter search results.
* It supports morphology, so a search for the word "cats" also finds the root word "cat."
* Distributed Sphinx can index among many machines, providing failover.
* It can create indexes of word prefixes of arbitrary length and indexes of infix substrings of varying lengths. For instance, a part number may be 10 characters wide. The prefix index would match against all possible substrings anchored at the start of the string. The infix index would match substrings anywhere within the string.
* It can be easily integrated with MYSQL and the data can be very easily fed from data bases directly and indexed. It also SQL select like querying.
* It supports advanced indexing and querying tools, several different ranking models (phrase proximity ranker, traditional statistics based BM25 ranker, etc).
* It supports advanced result set post-processing (SELECT with expressions, WHERE, ORDER BY, GROUP BY etc over text search results).

Sphinx has three components: an index generator, a search engine, and a command-line search utility:

**3.2.1 Indexer:** The index generator is called indexer. It queries your database, indexes each column in each row of the result, and ties each index entry to the row's primary key.

**3.2.2 Searchd:** The search engine is a daemon called searchd. The daemon receives search terms and other parameters, scours one or more indices, and returns a result. If a match is made, searchd returns an array of primary keys. Given those keys, an application can run a query against the associated database to find the complete records that comprise the match. Searchd communicates to applications through a socket connection.

**3.2.3 Search:** The handy search utility lets you conduct searches from the command line without writing code. If searchd returns a match, search queries the database and displays the rows in the match set. The search utility is useful for debugging your Sphinx configuration and performing offhand searches.

Sphinx also supports UTF-8 character set, exact phrase matches, Boolean, range, phrase queries etc. Sphinx also provides APIs for PHP, Perl, Ruby, C/C++ and many other programming languages. It powers popular web applications such as Craigslist, DailyMotion, NetLog, etc

* 1. **Solr**

Solr is an open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites such as Netflix, Source forge, digg, The Guardian, CiteseerX etc.

Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Apache Lucene and Apache Solr are both produced by the same Apache Software Foundation development team.

The following figure 7 shows the sample solr index analysis of a sample index created using IEEE HPCC 2011 conference proceedings data collection. More details about this course project to create of Information retrieval system on this data collection using different Information retrieval utilities will be discussed in the upcoming section. Figure 8 shows how the token fortes is analyzed and kept in the index. If you look at it carefully, the porter-stemmer filter removes the word suffix and keeps the root word ‘fort’. Figure 9 shows the verbose output of this analysis.

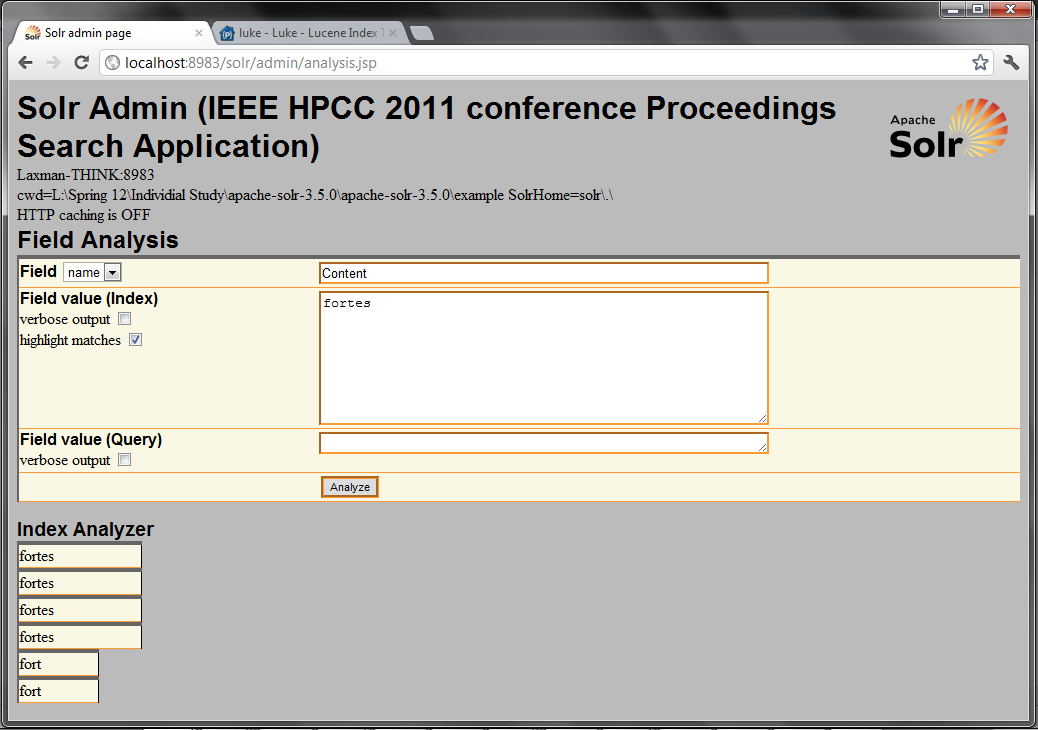


Fig.6. Index analysis in Solr

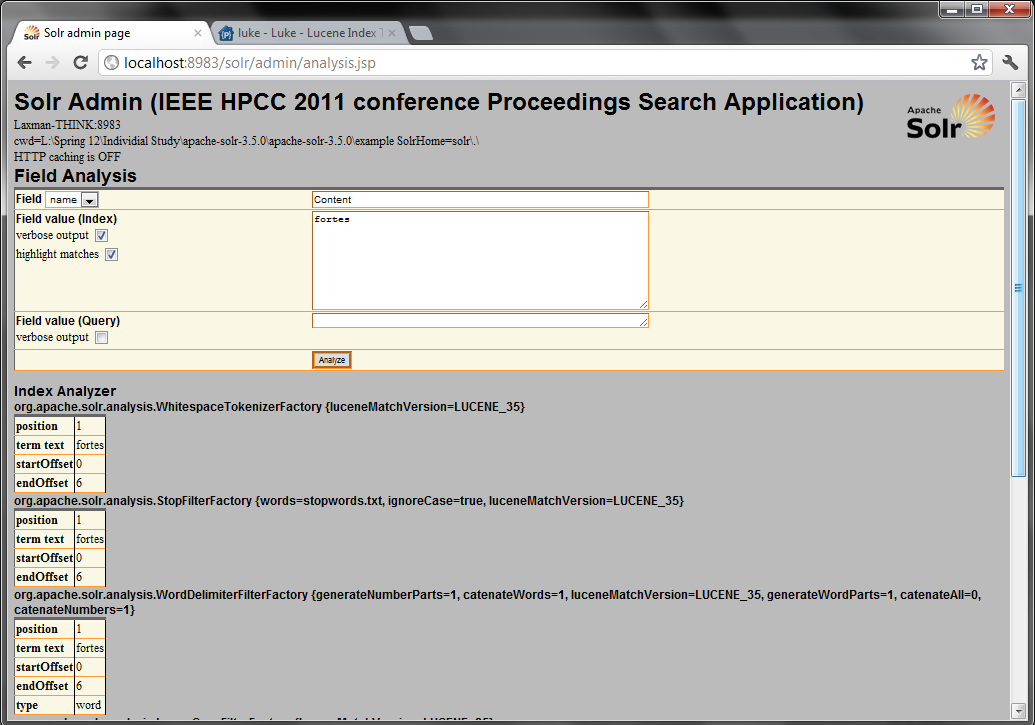


Fig.7. Index analysis in Solr (Verbose view)

1. **COURSE PROJECT – IEEE HPCC 2011 CONFERENCE PROCEEDINGS SEARCH SYSTEM**

The conference proceedings of High Performance computing Conference (HPCC) 2011 have been considered as data collection for building the information retrieval system as this course project which would facilitate free text searching on the proceedings.

Document format:All the conference proceedings are in the pdf format.

Three information retrieval systems were developed using the technologies/tools which we discussed in the previous section. The design, implementation and results are as presented below:

**4.1 Lucene Based Search engine CLI**

**4.1.1 Development tools:**

* Java, Eclipse, [Java Lucne package](http://www.apache.org/dyn/closer.cgi/lucene/java/3.6.0).

A CLI-based index/search application is developed in jave using lucene search library.

**4.1.2 Design**

As discussed above, by default, lucene doesn’t contain document parsers. But the developers can use open source [Tika](http://tika.apache.org/) framework which is a collection of parsers that supports various document formats including PDF, visio, RTF, outlook, word etc. The pdf files are pre-processed using the [Apache PDFBox](http://pdfbox.apache.org/) (a Java PDF library).

The parsed documents are converted to Lucene document with author’s name, title, abstract, content as fields and added the documents the lucene’s IndexWriter to index the documents. The searching part of this application receives free text query as input from the user and calls IndexSearcher, which intern opens the index and performs the search and returns the top matching documents.

User can run the program and select his option to index and search the data collection as shown below:

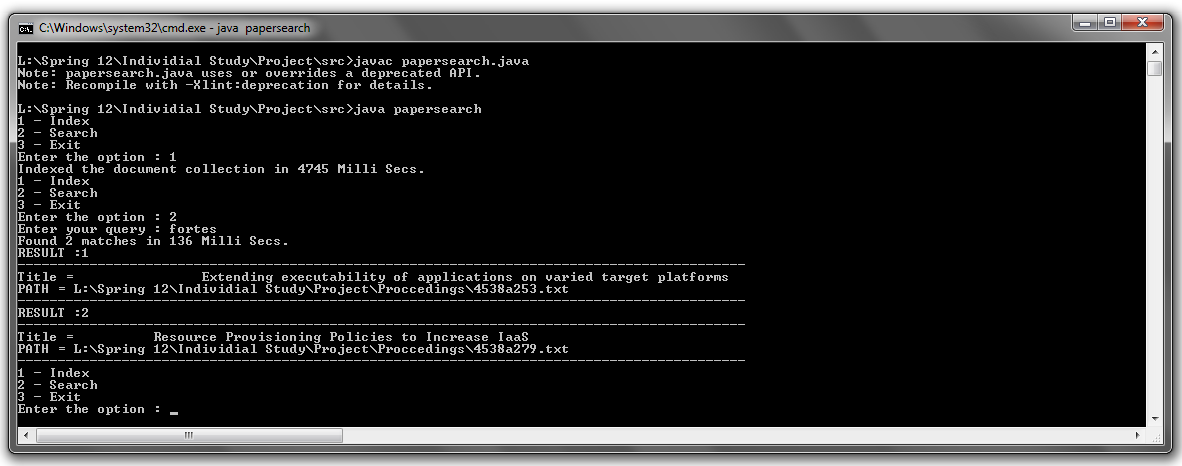


Fig.8. Lucene based HPCC 2011 conference proceedings search System - Command Line Interface

**4.1.3 Statistics:**

Size of the data collection: 35.8 MB

Index size: 4.26 MB

Indexing time: 4745 Milli Secs (on Intel Core i3- 210M CPU @ 2.10 GHz, Windows 7 Operating System and Indexing is done on NTFS file system).

Searching time: ~ 100 Milli Secs.

The complete source code can be found here.

**4.2 Lucene Based Search engine GUI**

A free text search engine web application has been developed with a User interface.

**4.2.1 Development tools**

* Java, Java Servlets, JSP, Eclipse Java EE IDE for Web Developers, [Java Lucne package](http://www.apache.org/dyn/closer.cgi/lucene/java/3.6.0), Apache Tomcat web server.

**4.2.2 Design**

As discussed above, the pdf files were processed using pdfbox. A jsp front end has been developed. The UI provides a textbox for the users to enter the free text query. If the index is already not present then the web application creates index and then returns the results. If the index is already build then it queries the index using IndexSearch class and returns the results.

**4.2.3 Screenshots**

The web interface operations are as shown below:

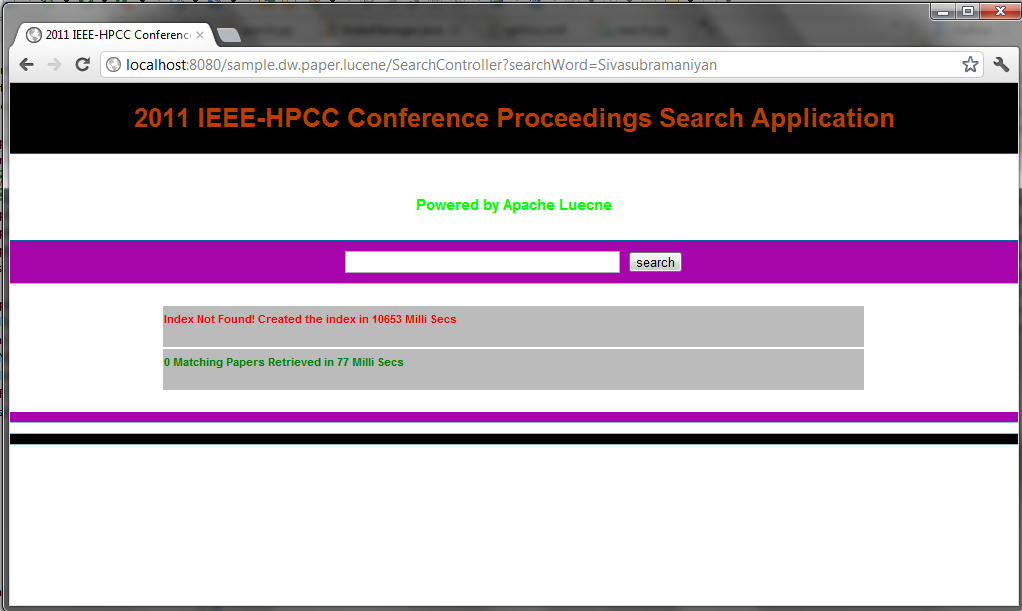


Fig.9. Lucene based HPCC 2011 Conference Proceedings Search System – web interface (indexing)

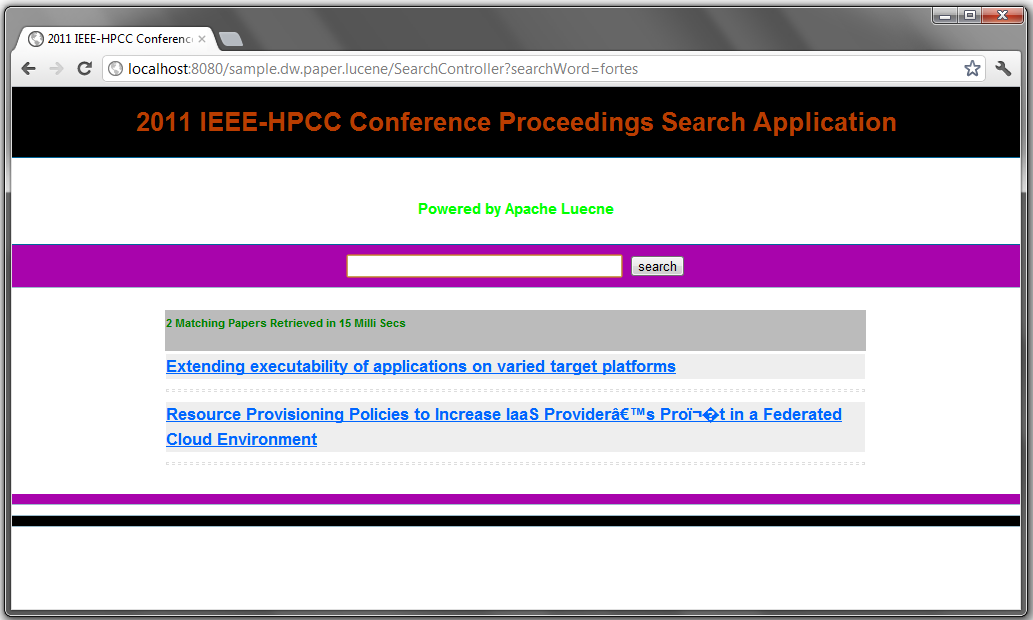


Fig.10. Lucene based HPCC 2011 Conference Proceedings Search System – web interface (searching)

**4.3 Sphinx Based Information Retrieval System**

**4.3.1 Development Tools**

* Php, sphinx-2.0.4-release-win32, pdfbox V1.6.0, jsp, java Servlets.

**4.3.2 Design**

A separate search engine has been developed for the same data collection using the sphinx search server. Sphinx too doesn’t have an in-built built-in parser. It is the developer’s responsibility to parse the pdf document and feed the data to the sphinx using its xmlpipe component.

The jar utility provided by the Apache Tika pdfbox has been used to parse the document. A php script has been written to parse the proceeding papers and it acts as data source to the indexer.

Sphinx provides us a command line search utility as well. We can also run the sphinx search server daemon (searchd) and it can be queried by establishing a connection to it. Figure 6 shows how to start the sphinx server daemon using the configuration file and the index name.



Fig.11. Sphinx based HPCC 2011 Conference Proceedings Search System – sphinx CLI interface (indexing)

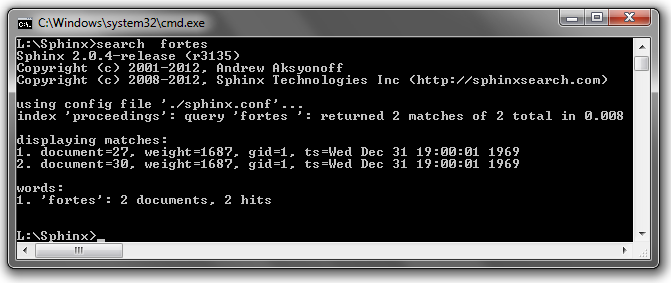


Fig.12. Sphinx based HPCC 2011 Conference Proceedings Search System – sphinx CLI interface (searching)

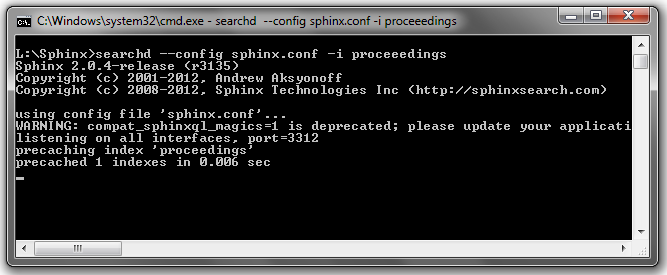


Fig.13. Sphinx based HPCC 2011 Conference Proceedings Search Server daemon

A web based interface is also developed to facilitate the HPCC conference proceedings search by using the index build in figure 4. Figure 7 shows the web interface developed using Java & jsp which connects to the searchd daemon and queries the index.

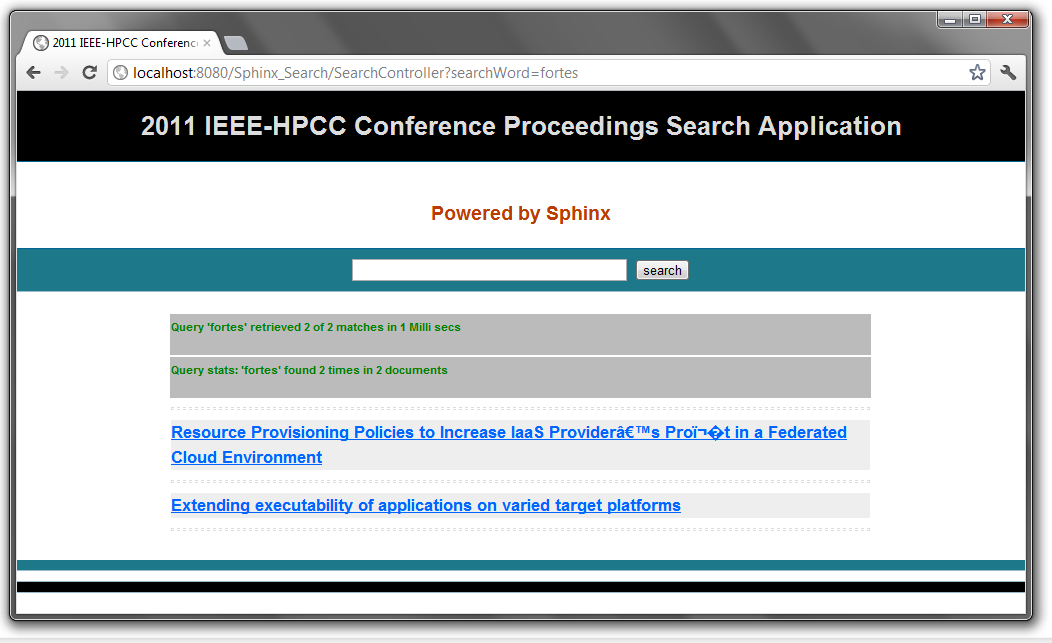


Fig.14. Sphinx based HPCC 2011 Conference Proceedings Search System – web interface (searching)

**4.3.2 Statistics**

Size of the data collection: 35.8 MB

Index size: 1.25 MB

Indexing time: 5273 Milli Secs (on Intel Core i3- 210M CPU @ 2.10 GHz, Windows 7 Operating System and Indexing is done on NTFS file system).

Searching time: 10-20 Milli Secs.

**4.4 Solr Based Information Retrieval System**

**4.4.1 Development tools**

* Php, xml, Apache Solr V3.5.0.

The same data collection has been indexed and searched through solr as well. Solr has the posting component which can accept pdf documents and index them. But in the following implementation, all the pdf documents were pre-processed using pdfbox CLI and an xml file has been created using a php script.

The solr search server has to be configured by altering the scheme.xml & solrconfig.xml. The parameters such as number fields in a document, type of each field, analyzers to be used to process the documents, default Boolean operator, default search field, cache size etc can be configured in these files.

The documents can be posted to the indexer using a post utility present under the examples/examplesdocs directory. Figure 7 show the indexing of the proceedings searching documents xml file generated using the php script discussed above.

The downloaded bundle itself contains the Jetty server. Starting solr is just executing a jar file called start.jar which is present inside the examples directory. The following figure.9 shows the running of HPCC conference proceedings search application.



Fig.15. Solr based HPCC 2011 conference proceedings search System– Indexing the data set in xml format using the Solr ‘post’ utility

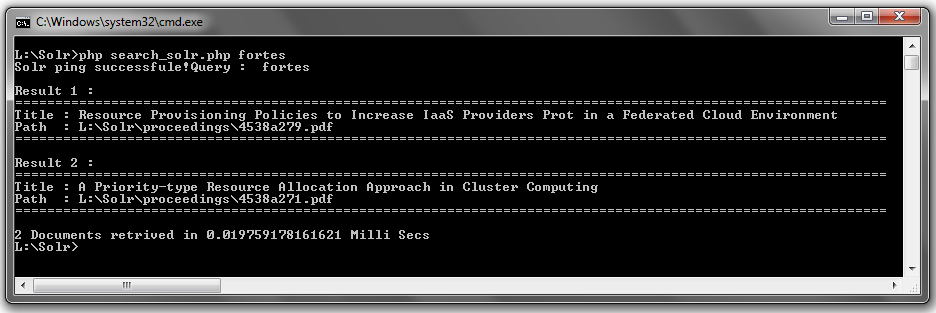


Fig.16. Solr API based HPCC 2011 Conference Proceedings Search System – Command Line Interface

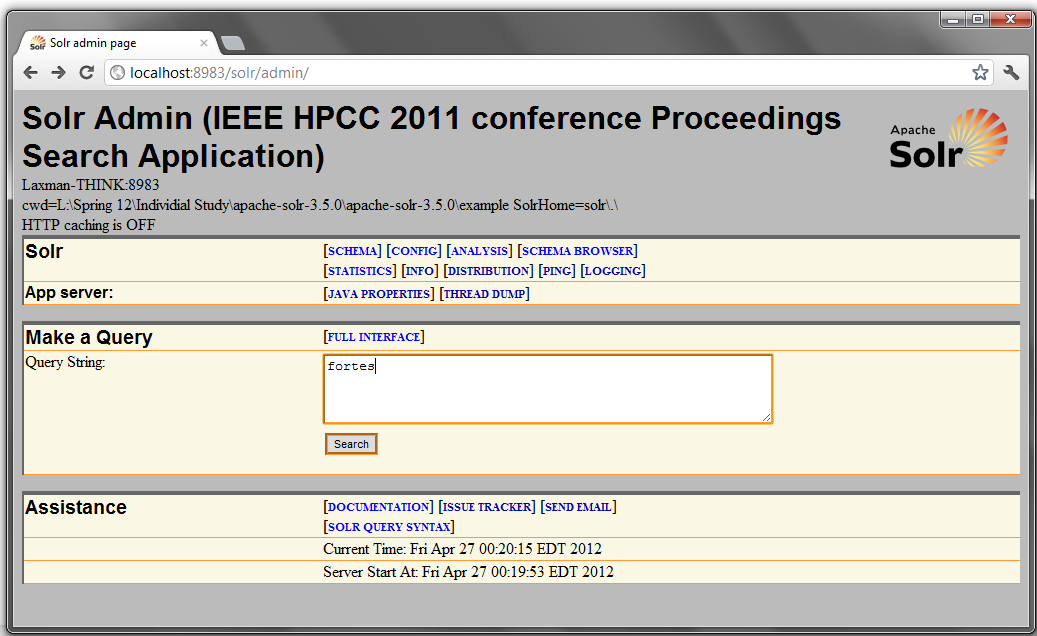


Fig.16. Solr based HPCC 2011 Conference Proceedings Search System – Searching the index

As it was done for sphinx, solr search server also can be consulted using TCP/IP connections and the queries can be processed over the indexes built. A simple CLI based search interface has been developed using php. The search interface is as shown in figure 9.

**4.4.2 Statistics**

Size of the data collection: 35.8 MB

Index size: 40.7 MB (which is greater than the collection size as all the fields in the documents are stored & indexed)

Indexing time: 6509 Milli Secs (on Intel Core i3- 210M CPU @ 2.10 GHz, Windows 7 Operating System and Indexing is done on NTFS file system).

Searching time: ~10 Milli Secs.

1. **SEARCH ENGINE BASED INFORMATION RETRIEVAL vs RDBMS BASED INFORMATION RETRIEVAL**

The following experiment is conducted on the sample morphological data set to compare the RDBMS based information retrieval and search engine based information retrieval approaches qualitatively (precision, recall) and quantitatively (response time).

**5.1 Experiment Description**

The following experiment has been conducted on Darwin core archive format data collection to compare and contrast the differences between the search engines based technologies and tools discussed above and the relation data bases on the sample data set. The data set is available via ftp at this [location](ftp://ammatsun.acis.ufl.edu/datasets/).

The isNULL, Boolean, range and string matching queries were considered and recall, precision & reaction times were compared. Recall gives us a sense of the percentage of correct records retrieved from the index and precision measures us the amount of records which are correct / relevant out of the returned results omitting the false positives in the response. Reaction time is the measure of time taken to process the query. Indexing time is the time taken to build the index / database. MySQL data base is used for this experiment. The data set consist of csv files which contain records in a tab separated file. It contains the following three different data collection in DWCA format.

* [Inventory of the Marie-Victorin Herbarium (MT)](http://data.canadensys.net/ipt/resource.do?r=mt-specimens)
* [Ichthyology](http://collections.flmnh.ufl.edu:8080/ipt-2.0.3/resource.do?r=ichthyology)
* [whole-morph bank](http://code.google.com/p/gbif-providertoolkit/downloads/detail?name=ipt-2.0.3.war)

In general, there are 176 different fields/attributes related to DWCA archival data sets. But the three datasets have very different fields (out of this 175) populated, but they all represent specimens. For example whole morph-bank collection has 46 attributes associated, ichthyology collection has 30 attributes associated to it and MT inventory has 17 attributes set.

**5.2 Data Processing**

A superset table with 177 attributes (176 DWCA fields and a unique document ID field – (sphinx requires a unique ID / primary key for records) has been created. All the tables have been read and the available fields are set and the unavailable fields are set to NULL and added into the larger table. Whole morph bank collection is slightly different in the sense that each specimen may have multiple images records attached to it, which is in a separate file (multimedia.txt), which shares a Foreign key – Primary key relationship (through ID field), which is normalized and expanded through an left outer join with the occurrences csv file and the multimedia csv file. Appropriate data types have been used in mySQL such as Integer, decimal, varchar, Data & Time. All the data is represented in UTF-8 format both in mySQL and in different information retrieval systems.

**5.3 Comparison of Results**

**Note**

1. The precision and recall calculations are not done by pair wise comparison of results (Except for range and Boolean queries where the result set is very small) of mysql and search engines. The false +Ves and –Ves in the results, the possible results in the unreturned collection space are also not verified. The precision & recall % calculations are plainly based on the result set count.

2. The mySQL database is indexed on the fields used in the queries recordedBy, year and preparations.

3. Caching is turned off in all the systems. The processing times of the query for the first time is noted.

**5.3.1 Term Query / String Query**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **mySQL** | **Sphinx** | **Solr** | **Lucene Implementation** |
| **Query** | select \* from dwca\_all where recordedBy like '%Beamish%' | search -c sphinx.conf -i dwca @recordedBy Beamish | recordedBy:Beamish | recordedBy:Beamish |
| **No of records returned** | 105 | 105 | 105 | 105 |
| **Precision** | 100% | 100% | 100% | 100% |
| **Recall** | 100% | 100% | 100% | 100% |

**5.3.2 Range Query**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **mySQL** | **Sphinx** | **Solr** | **Lucene Implementation** |
| **Query** | select \* from dwca\_all where recordedBy = 'Beamish' AND year >= 2002 AND year <= 2003 | search -c sphinx.conf -i dwca @recordedBy Beamish AND  @year >= 2002 AND @year <= 2003 | recordedBy:Beamish AND year:[2002 TO 2003] | recordedBy:Beamish AND year:[2002 TO 2003] |
| **No of records returned** | 5 | 5 | 5 | 5 |
| **Precision** | 100% | 100% | 100% | 100% |
| **Recall** | 100% | 100% | 100% | 100% |

**5.3.3 Boolean Query**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **mySQL** | **Sphinx** | **Solr** | **Lucene Implementation** |
| **Query** | select \* from dwca\_all where recordedBy = 'Beamish' AND year = 2002 | search -c sphinx.conf -i dwca @recordedBy Beamish @year 2002 | recordedBy:Beamish AND year:2002 | recordedBy:Beamish AND year:[2002 TO 2003] |
| **No of records returned** | 4 | 4 | 4 | 4 |
| **Precision** | 100% | 100% | 100% | 100% |
| **Recall** | 100% | 100% | 100% | 100% |

**5.3.4 isNULL Query**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **mySQL** | **Sphinx\*** | **Solr** | **Lucene Implementation** |
| **Query** | SELECT \* FROM dwca\_all where preparations is NULL | search -c sphinx.conf -i dwca @ preparations isNULL | -preparations[\* TO \*] | preparations:isNULL |
| **No of records returned** | 109703 | 109709 | 109709 | 109709 |
| **Precision** | 100% | ~100% | ~100% | ~100% |
| **Recall** | 100% | ~100% | ~100% | ~100% |

By default sphinx doesn’t index null fields. While indexing, in the sphinx.conf the null fields in preparations field is modified and indexed as “isNULL” string for those records which have null preparations field by using the “SELECT …, IF(preparations IS NULL,'isNULL', preparations) AS preparations, ....” query statement. Also in the lucene implementation, the null fields are indexed as isNULL strings.

**5.4 Key Observations**

* + - Easy to integrate Data bases with information retrieval systems such as Solr & sphinx.
    - On a well defined, structured data the information retrieval systems perform well and the precision and recall are close to 100 %.
    - IR systems are well suited for naïve users as free text queries are prevalent and DBMS are suited for advanced users who can express the information need in a complex query language using complex operators.
    - Performing isNULL queries and range queries is difficult on search engine based approach and is very easy on data base based approach.
    - Search engine based solution can score and rank the matching documents.

1. **CONCUSSION**

This course helped me learn the tricks applied behind the process of information retrieval by various web applications that we use in our day to day life such as Wikipedia, search engine etc. This sets up a perfect platform to extend these ideas and explore advanced topics in information retrieval such as indexing spatial data and probabilistic based information retrieval systems.

1. **REFERENCES**
2. Introduction to information retrieval, Cambridge University Press, by Christopher D. Manning, Prabhakar Raghavan & Hinrich Schutze.
3. Introduction to search with Sphinx, from installation to relevance tuning, O'Reilly Media by Andrew Aksyonoff.
4. Manning Lucene in Action, 2nd Edition, Manning publications by Erik Hatcher and Otis Gospodnetić.
5. http://sphinxsearch.com/docs/current.html
6. http://sphinxsearch.com/docs/current.html
7. http://sphinxsearch.com/info/talks/
8. http://sphinxsearch.com/info/articles/
9. http://justcramer.com/2008/02/24/in-depth-django-sphinx-tutorial/
10. http://www.wikivs.com/wiki/Lucene\_vs\_Sphinx
11. http://nicholaschase.com/2010/09/25/comparing-the-apache-solr-and-sphinx-search-platforms/
12. http://sphinxsearch.com/docs/2.0.4/
13. http://www.lucenetutorial.com/lucene-in-5-minutes.html
14. http://www.lucenetutorial.com/lucene-query-syntax.html
15. http://www.lucenetutorial.com/basic-concepts.html
16. http://www.ibm.com/developerworks/opensource/library/os-apache-lucenesearch/index.html
17. http://www.ibm.com/developerworks/opensource/library/wa-lucene/index.html
18. www.wikipedia.org