ISSN 2320 -2602

**International Journal of Advances in Computer Science and Technology (IJACST)**, Vol.3 , No.11, Pages : 19-22 (2014)
*Special Issue of ICCECT 2014 - Held during 01-02 December, 2014,Bangkok, Thailand*

# A Search Engine for Restaurants using Seasonal and Regional Characteristics

**Yeunjung Kim[1], Zhanying Jin[2], Sujoung Oh[3], Minsoo Lee[4]**
[1]Dept. Computer Science and Engineering, Ewha Womans University, Korea, imkimyj@ewhain.net
[2]Dept. Computer Science and Engineering, Ewha Womans University, Korea, jxy5130@gmail.com
[3]Dept. Computer Science and Engineering, Ewha Womans University, Korea, crystal7862@gmail.com
[4]Dept. Computer Science and Engineering, Ewha Womans University, Korea, mlee@ewha.ac.kr

**Abstract *:*** There are so many restaurants and also sites that try to organize and list such restaurants. Some sites also try to provide primitive searching of restaurants but most of them are limited to basic lookup of restaurants by names. Ranking is a very important part of information retrieval systems. There are some basic ranking techniques but search engine ranking algorithms are closely guarded secrets. Therefore we'd like to create a search engine for restaurant using information related to seasonal and regional specialties. We think that it is helpful to search restaurants for seasonal specialties in a certain geographical zone. We implemented the search engine using Lucene, MySQL, and Jsoup and conduct a simple experiment.

**Key words :** Lucene, Jsoup, Search engine, Retrieval System

## INTRODUCTION

There are so many restaurants in the world, and also various types of search methods for restaurants. Most of the search engines have ranking algorithms since ranking is also a very important part of information retrieval systems. There are some basic ranking techniques but search engine ranking algorithms are closely guarded secrets. Search engine ranking algorithms are hidden for at least two reasons: Search engine companies want to protect their methods from their competitors, and they also want to make it difficult for Web site owners to manipulate their rankings.

The purpose of our search engine is to retrieve the address and simple description of restaurants using the information related to seasonal and regional specialties. We think that it is helpful to search restaurants for seasonal specialties in a certain zone. We implemented the search engine using Lucene, MySQL, and Jsoup and conduct a simple experiment using the data of the restaurant blog.

The organization of the paper is as follows. In the second section, we will introduce the related works, and in the third section, we will provide an explanation of the proposed search algorithm. In fourth section, we deal with implementation and simple experiment. In the last section, we provide the conclusion and further study.

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(No. 2012R1A1A2006850).

This work was also supported by the National Research Foundation of Korea grant funded by the government (Ministry of Education, Science and Technology) (NRF-2012M3A9D1054744).

## RELATED RESEARCH

### Search Engine

A web search engine is a software system that is designed to search for information on the World Wide Web. The search results are generally presented in a line of results often referred to as search engine results pages. The information may be a mix of web pages, images, and other types of files. Some search engines also mine data available in databases or open directories. Unlike web directories, which are maintained only by human editors, search engines also maintain real-time information by running an algorithm on a web crawler.

Web search engines work by storing information about many web pages, which they retrieve from the HTML markup of the pages. These pages are retrieved by a Web crawler which follows every link on the site. The site owner can exclude specific pages by using robots.txt.

The search engine then analyzes the contents of each page to determine how it should be indexed (for example, words can be extracted from the titles, page content, headings, or special fields called meta tags). Data about web pages are stored in an index database for use in later queries. A query from a user can be a single word. The index helps find the information relating to the query as quickly as possible. Some search engines, such as Google, store all or part of the source page as well as information about the web pages, whereas others, such as AltaVista, store every word of every page they find. This cached page always holds the actual search text since it is the one that was actually indexed, so it can be very useful when the content of the current page has been updated and the search terms are no longer in it. This problem might be considered a mild form of link rot, and Google's handling of it increases usability by satisfying user expectations that the search terms will be on the returned webpage. This satisfies the principle of least astonishment, since the user normally expects that the search terms will be on the returned pages. Increased search relevance makes these cached pages very useful as they may contain data that may no longer be available elsewhere [1].

### Classic Models in Information Retrieval

There are three classic models in information retrieval: the Boolean, the vector, and the probabilistic models. The Boolean model is based on set theory and Boolean algebra. Retrieval is based on whether or not the documents contain the query terms and makes return exact matches. The traditional Boolean approach does not provide a relevance ranking of the retrieved documents, although modern Boolean approaches can make use of the location and frequency of keywords in document structure.

In the vector model, a document and a user query are represented as vectors in a t-dimensional space, where t equivalent with the number of index terms in the query. The index terms of the query are basis vectors in this space, and the document can express linear combination of the basis vectors. The coefficients equal 1, if the index terms are in the document and 0 otherwise. Document relevance to the query can be quantified by the cosine of the angle between these two vectors.

The probabilistic model ranks the documents based on the quotient of the probability that the document is relevant to the query and the probability that the document is non-relevant to the query. User relevance feedback is very important to this model.

### Lucene

Apache Lucene is a free and open source information retrieval software library, originally created in Java by Doug Cutting. It is supported by the Apache Software Foundation and is released under the Apache Software License. Lucene has been ported to other programming languages including Delphi, Perl, C#, C++, Python, Ruby, and PHP. While suitable for any application which requires full text indexing and searching capability, Lucene has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching [2].

At the core of Lucene's logical architecture is the idea of a document containing fields of text. This flexibility allows Lucene's API to be independent of the file format. Text from PDFs, HTML, Microsoft Word, and OpenDocument documents, as well as many others, can all be indexed as long as their textual information can be extracted [3].

## PROPOSED ALGORITHM

### Basic Approach

Ranking is a very important part of information retrieval systems. There are some ranking techniques but search engine ranking algorithms are closely guarded secrets [4]. Our approach mainly uses the information about seasonal and regional specialties to determine the ranking. We also use the grade of the restaurant and the frequency of the season.

### Ranking Algorithm

The Ranking Algorithm of the search engine is as follows:

- Retrieve restaurant using season and region as keywords (terms). This will retrieve records containing both those terms in any order. In other words, it uses the AND operator for the keywords and the result contains the list of restaurants which serves seasonal food in a certain zone.
- The higher a restaurant's grade, the higher the ranking of the result. The Grade is the score that is calculated by the average of the evaluation score of a user multiplied by the number of users who evaluated the restaurant.
- The higher the word's frequency, the lower the ranking of the result. If the frequency of the word for different seasons is high it means we can eat the food any time.

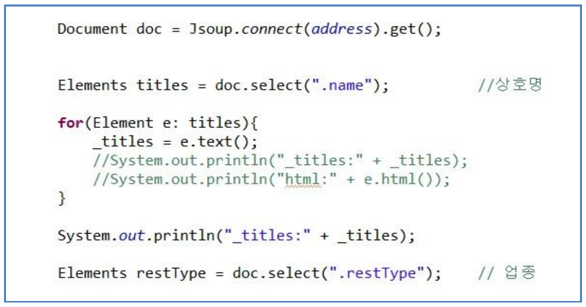
## IMPLEMENTATION & EXPERIMENTS

### Implementation

This application is divided into three parts. One part is the crawler which collects the web pages. The other two parts are the indexing and searching parts using Lucene.

#### *A. Web Page Crawler*

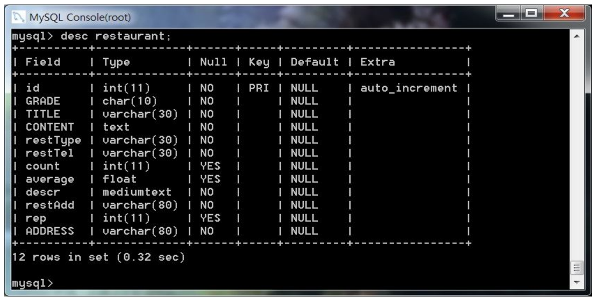
In the 'Web Page Crawling' part, the necessary data of the web page is collected and stored into the Database using MySQL. A Web crawler starts with a list of URLs to visit, called the seeds. Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. We extracted necessary data form the restaurant blog using Jsoup API. Figure 1 illustrates the part of the source code about extracting data from an HTML document [5].



```
Document doc = Jsoup.connect(address).get();


Elements titles = doc.select(".name");          //상호명

for(Element e: titles){
    _titles = e.text();
    //System.out.println("_titles:" + _titles);
    //System.out.println("html:" + e.html());
}

System.out.println("_titles:" + _titles);

Elements restType = doc.select(".restType");    // 업종
```

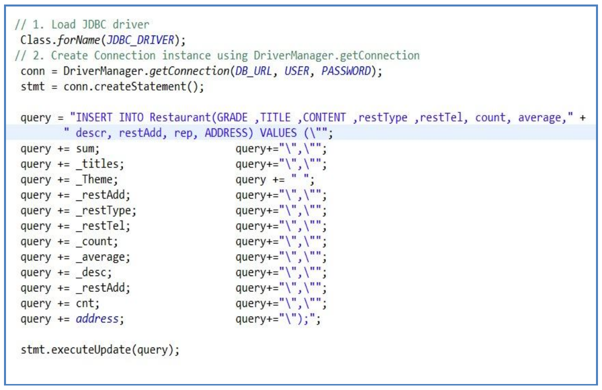
**Figure 1:** Extracting data from an HTML document using Jsoup

In order to store the necessary data for indexing and searching, we created the restaurant table in the MySQL database and connected to the database using the JDBC driver. Figure 2 illustrates the schema for the restaurant table. In this part, the grade and frequency are computed in advance and stored in the DB. As previously stated, the value stored for 'GRADE' field is grade, which is the score calculated by averaging the user's scores multiplied by the number of users who evaluated the restaurant. They are both important values to decide the ranking. The value of the 'rep' field is the word frequency of the different words for 'season'. The Content field which contains restaurant theme and address is used for searching. ADDRESS field is the web address of the restaurant blog.



```
mysql> desc restaurant;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| GRADE    | char(10)     | NO   |     | NULL    |                |
| TITLE    | varchar(30)  | NO   |     | NULL    |                |
| CONTENT  | text         | NO   |     | NULL    |                |
| restType | varchar(30)  | NO   |     | NULL    |                |
| restTel  | varchar(30)  | NO   |     | NULL    |                |
| count    | int(11)      | YES  |     | NULL    |                |
| average  | float        | YES  |     | NULL    |                |
| descr    | mediumtext   | NO   |     | NULL    |                |
| restAdd  | varchar(80)  | NO   |     | NULL    |                |
| rep      | int(11)      | YES  |     | NULL    |                |
| ADDRESS  | varchar(80)  | NO   |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
12 rows in set (0.32 sec)

mysql>
```

**Figure 2:** Restaurant table Schema



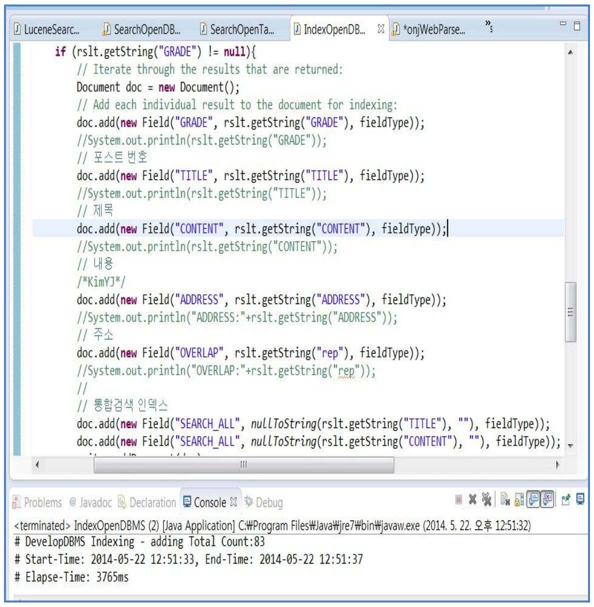
```
// 1. Load JDBC driver
 Class.forName(JDBC_DRIVER);
// 2. Create Connection instance using DriverManager.getConnection
 conn = DriverManager.getConnection(DB_URL, USER, PASSWORD);
 stmt = conn.createStatement();

 query = "INSERT INTO Restaurant(GRADE ,TITLE ,CONTENT ,restType ,restTel, count, average," +
        " descr, restAdd, rep, ADDRESS) VALUES (\"";
 query += sum;                    query+="\",\"";
 query += _titles;                query+="\",\"";
 query += _Theme;                 query += " ";
 query += _restAdd;               query+="\",\"";
 query += _restType;              query+="\",\"";
 query += _restTel;               query+="\",\"";
 query += _count;                 query+="\",\"";
 query += _average;               query+="\",\"";
 query += _desc;                  query+="\",\"";
 query += _restAdd;               query+="\",\"";
 query += cnt;                    query+="\",\"";
 query += address;                query+="\");";

 stmt.executeUpdate(query);
```

**Figure 3:** Store necessary data into restaurant table using JDBC driver

Figure 3 shows the part of the source code connecting to the MySQL database using the JDBC driver.

## *B. Indexing*

To search based on Korean words, the StandardAnalyzer is first converted to the KoreanAnalyzer.



```
    if (rslt.getString("GRADE") != null){
        // Iterate through the results that are returned:
        Document doc = new Document();
        // Add each individual result to the document for indexing:
        doc.add(new Field("GRADE", rslt.getString("GRADE"), fieldType));
        //System.out.println(rslt.getString("GRADE"));
        // 포스트 번호
        doc.add(new Field("TITLE", rslt.getString("TITLE"), fieldType));
        //System.out.println(rslt.getString("TITLE"));
        // 제목
        doc.add(new Field("CONTENT", rslt.getString("CONTENT"), fieldType));
        //System.out.println(rslt.getString("CONTENT"));
        // 내용
        /*KimYJ*/
        doc.add(new Field("ADDRESS", rslt.getString("ADDRESS"), fieldType));
        //System.out.println("ADDRESS:"+rslt.getString("ADDRESS"));
        // 주소
        doc.add(new Field("OVERLAP", rslt.getString("rep"), fieldType));
        //System.out.println("OVERLAP:"+rslt.getString("rep"));
        //
        // 통합검색 인덱스
        doc.add(new Field("SEARCH_ALL", nullToString(rslt.getString("TITLE"), ""), fieldType));
        doc.add(new Field("SEARCH_ALL", nullToString(rslt.getString("CONTENT"), ""), fieldType));
```

```
<terminated> IndexOpenDBMS (2) [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (2014. 5. 22. 오후 12:51:32)
# DevelopDBMS Indexing - adding Total Count:83
# Start-Time: 2014-05-22 12:51:33, End-Time: 2014-05-22 12:51:37
# Elapse-Time: 3765ms
```

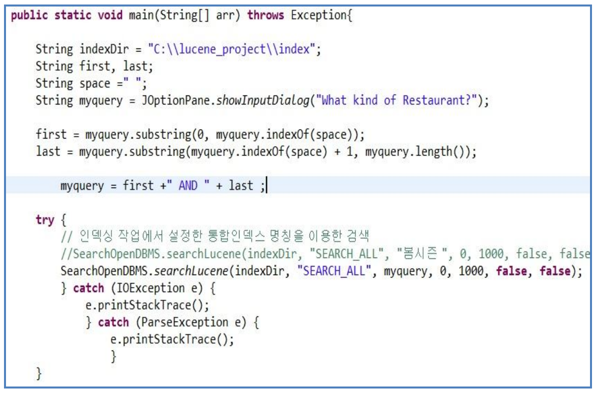
**Figure 4:** Adding documents to an index

A Document is Lucene’s atomic unit of indexing and searching and Fields is a section of a Document. Each field has two parts, a name and a value. Fields contain the “real” content. Lucene should manipulate the Field’s value when you add the document to the index.

In order to index raw content sources in the restaurant table, we first translate them into Lucene’s Documents and Fields. The GRADE, TITLE, CONTENT, ADDRES and rep field in restaurant table have the necessary data. We translated the values of the fields into Lucene’s Documents and Fields. Figure 4 shows that code iterates over the raw content, creating Document and Fields and then adds the Documents to the index.

## *C. Query Rewriting*

The QueryParser translates query expressions into one of Lucene’s built-in query types. If the Query expression is ‘java junit’, Lucene matches documents that contain the term java or junit, or both, in the default field. So the user’s query must be rewritten such as ‘java AND junit’. In other words, we have to insert the AND operator among individual words of the user’s input value to convert it into the appropriate form. Figure 5 shows this rewriting of the user query.

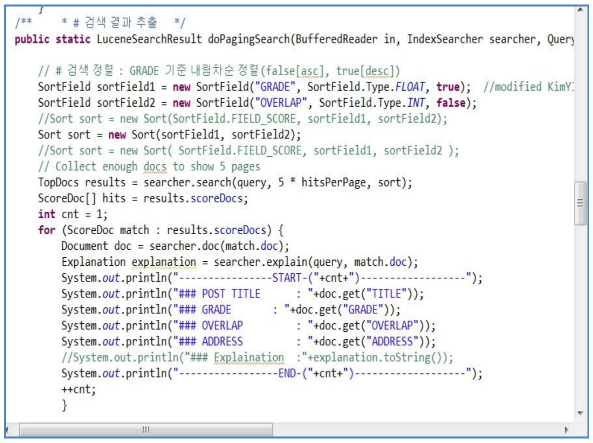


```
public static void main(String[] arr) throws Exception{

    String indexDir = "C:\\lucene_project\\index";
    String first, last;
    String space =" ";
    String myquery = JOptionPane.showInputDialog("What kind of Restaurant?");

    first = myquery.substring(0, myquery.indexOf(space));
    last = myquery.substring(myquery.indexOf(space) + 1, myquery.length());

        myquery = first +" AND " + last ;

    try {
        // 인덱싱 작업에서 설정한 통합인덱스 명칭을 이용한 검색
        //SearchOpenDBMS.searchLucene(indexDir, "SEARCH_ALL", "봄시즌", 0, 1000, false, false
        SearchOpenDBMS.searchLucene(indexDir, "SEARCH_ALL", myquery, 0, 1000, false, false);
        } catch (IOException e) {
            e.printStackTrace();
            } catch (ParseException e) {
                e.printStackTrace();
                }
    }
```

**Figure 5:** Rewriting the user query

## *D. Ranking*

By default, Lucene sorts the documents in descending relevance score order, where the most relevant documents appear first. But, to use grade and frequency value for sorting, we used ways which search results are sorted by multiple field values in either ascending or descending order. Sorting by multiple fields is important whenever your primary sort leaves ambiguity when there are equal values. Implicitly we’ve been sorting by multiple fields, since the Sort object appends a sort by document ID in appropriate cases. But we control the sort fields using an array of SortFields. This code uses grade as a primary sort in descending order and finally, restaurant with equal grade score are sorted by ascending frequency of the words for season. A SortField holds the field name, a field type, and the reverse order flag. The SortField contains constants for several field types.

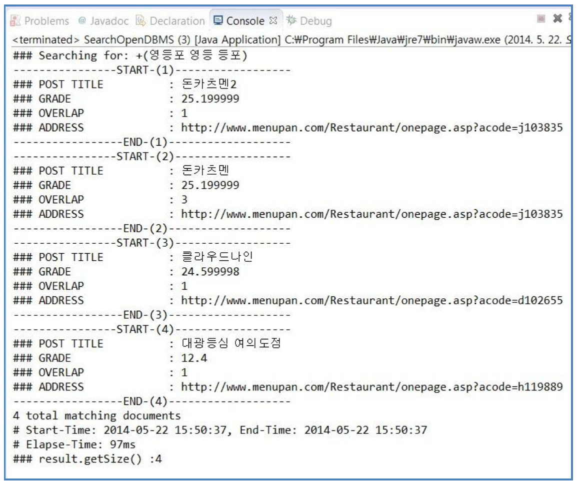


```
    }
/**     * # 검색 결과 추출   */
public static LuceneSearchResult doPagingSearch(BufferedReader in, IndexSearcher searcher, Query

    // # 검색 정렬 : GRADE 기준 내림차순 정렬(false[asc], true[desc])
    SortField sortField1 = new SortField("GRADE", SortField.Type.FLOAT, true);  //modified KimY
    SortField sortField2 = new SortField("OVERLAP", SortField.Type.INT, false);
    //Sort sort = new Sort(SortField.FIELD_SCORE, sortField1, sortField2);
    Sort sort = new Sort(sortField1, sortField2);
    //Sort sort = new Sort( SortField.FIELD_SCORE, sortField1, sortField2 );
    // Collect enough docs to show 5 pages
    TopDocs results = searcher.search(query, 5 * hitsPerPage, sort);
    ScoreDoc[] hits = results.scoreDocs;
    int cnt = 1;
    for (ScoreDoc match : results.scoreDocs) {
        Document doc = searcher.doc(match.doc);
        Explanation explanation = searcher.explain(query, match.doc);
        System.out.println("---------------START-("+cnt+")-----------------");
        System.out.println("### POST TITLE      : "+doc.get("TITLE"));
        System.out.println("### GRADE       : "+doc.get("GRADE"));
        System.out.println("### OVERLAP         : "+doc.get("OVERLAP"));
        System.out.println("### ADDRESS         : "+doc.get("ADDRESS"));
        //System.out.println("### Explaination  :"+explanation.toString());
        System.out.println("----------------END-("+cnt+")------------------");
        ++cnt;
        }
```

**Figure 6:** Sorting by multiple fields

## Experiments

For example, assume that the 1st ranking restaurant and 2nd ranking restaurant have equal value of the grade and the primary sort is ambiguous. This code uses grade as a primary sort in descending order and finally, restaurant with equal grade score are sorted by ascending frequency of the words for season. The 2nd ranking restaurant has a higher score for the frequency of the season words than the 1st ranking restaurant. As a result, the 1st ranking restaurant is ranked higher than the 2nd ranking restaurant as shown in Figure 7.



```
<terminated> SearchOpenDBMS (3) [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (2014. 5. 22.
### Searching for: +(영등포 영등 등포)
----------------START-(1)-----------------
### POST TITLE          : 돈카츠멘2
### GRADE               : 25.199999
### OVERLAP             : 1
### ADDRESS             : http://www.menupan.com/Restaurant/onepage.asp?acode=j103835
-----------------END-(1)------------------
----------------START-(2)-----------------
### POST TITLE          : 돈카츠멘
### GRADE               : 25.199999
### OVERLAP             : 3
### ADDRESS             : http://www.menupan.com/Restaurant/onepage.asp?acode=j103835
-----------------END-(2)------------------
----------------START-(3)-----------------
### POST TITLE          : 클라우드나인
### GRADE               : 24.599998
### OVERLAP             : 1
### ADDRESS             : http://www.menupan.com/Restaurant/onepage.asp?acode=d102655
-----------------END-(3)------------------
----------------START-(4)-----------------
### POST TITLE          : 대광등심 여의도점
### GRADE               : 12.4
### OVERLAP             : 1
### ADDRESS             : http://www.menupan.com/Restaurant/onepage.asp?acode=h119889
-----------------END-(4)------------------
4 total matching documents
# Start-Time: 2014-05-22 15:50:37, End-Time: 2014-05-22 15:50:37
# Elapse-Time: 97ms
### result.getSize() :4
```

**Figure 7:** Search Result

## CONCLUSION AND FUTURE WORK

This paper proposes a ranking mechanism to be used for searching for restaurants with seasonal and regional characteristics. We use the CustomScoreQuery API which is a Query API that sets document scores as a programmatic function of several (sub) scores: the score of its subQuery and the score of its ValueSourceQuery. Subclasses can modify the computation by overriding getCustomScoreProvider. The ranking is uses the grade as a primary sort in descending order and finally, restaurant with equal grade score are sorted by ascending frequency of the words for season.

Further work can be done for implementing more sophisticated rankings based on additional characteristics of restaurants based on price and atmosphere, etc. Also rankings based on more complicated relationships among users having evaluations of restaurants could also be formulated.

## REFERENCES

[1] Web Search Engine, Wikipedia, [Online] Available: http://en.wikipedia.org/wiki/Web_search_engine
[2] Lucene, Apache Software Foundation, [Online] Available: http://lucene.apache.org/core
[3] M. McCandless, E. Hatcher, O. Gospodnetic, *Lucene IN ACTION*, 2nd ed., Manning Publications, 2010.
[4] M. Princz, “Search engine ranking,” in *Proc. 7th International Conference on Applied Informatics*, Eger, Hungary, 2007, Vol. 2. pp. 417–422.
[5] Jsoup: Java HTML parser, [Online] Available: http://jsoup.org