Homework 3: Comparing Search Engine Ranking Algorithms

Objectives:

- Experience using Solr
- Investigating ranking strategies

Preparation

In the previous exercise you used crawler4j to crawl a portion of the USC website. As a result of this crawl you should have downloaded and saved HTML/PDF/DOC files. In this exercise you will index those pages using Solr and then modify Solr to compare different ranking strategies.

Solr can be installed on Unix or Windows machines. However, it is much easier to run Solr on a Unix computer. Therefore, we have provided instructions for installing an implementation of Unix, called Ubuntu, on a Windows computer. For details see

http://www-scf.usc.edu/~csci572/Exercises/UbuntuVirtualBoxFinal.pdf

Once Ubuntu is successfully installed, or if you are using a Mac or some other Unix computer, you can then follow the instructions for installing Solr directly, which can be found here

http://www-scf.usc.edu/~csci572/2016Spring/hw3/SolrInstallation.pdf

The above instructions are for solr-5.3.1.zip. You can either download the file from

http://lucene.apache.org/solr/downloads.html/

or from the class website at

http://www-scf.usc.edu/~csci572/software/solr-5.3.1-src.tgz

Once Solr is installed you need to have it index the web pages that you saved. Instructions for doing this can be found here

http://www-scf.usc.edu/~csci572/2016Spring/hw3/IndexingwithTIKAV3.pdf

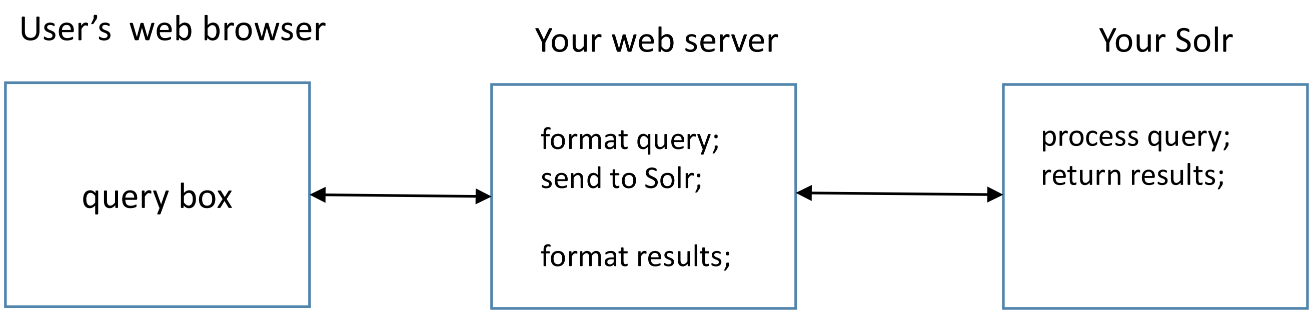
Solr provides a simple user interface that you can use to explore your indexed web pages.

Description of the Exercise

Step 1

Now that your set up is complete you need to have access to a web server that can deliver web pages and run scripts. Using this web server you will create a web page with a text box which a user can retrieve and then enter a query. The user’s query will be processed by a program at your web server which formats the query and sends it to Solr. Solr will process the query and return some results in JSON format. A program on your web server will re-format the results and present them to the user as any search engine would do.

Below is a rough outline for how you could structure your solution to the exercise. All three elements: web browser, web server, and Solr would be located on your laptop. Your web server might be the Apache web server coupled with the PhP programming language. An alternative solution would be to use node.js as the server/programming component. In the case of node.js, the programming language is JavaScript. Whatever you use, your program would send the query web page to the user, and then send the user's query to Solr which produces the results. The results are returned by Solr to the same web server and converts the results into a nice looking web page that is eventually returned.



Solr server supports several clients (different languages). Clients use requests to ask Solr to do things like perform queries or index documents. Client applications can reach Solr by creating HTTP requests and parsing the HTTP responses. Client APIs encapsulate much of the work of sending requests and parsing responses, which makes it much easier to write client applications.

Clients use Solr's five fundamental operations to work with Solr. The operations are query, index, delete, commit, and optimize. Queries are executed by creating a URL that contains all the query parameters. Solr examines the request URL, performs the query, and returns the results. The other operations are similar, although in certain cases the HTTP request is a POST operation and contains information beyond whatever is included in the request URL. An index operation, for example, may contain a document in the body of the request.

There are several client APIs available for Solr, refer https://wiki.apache.org/solr/IntegratingSolr . As an example, here we explain how to create a PHP client that accepts input from the user in a HTML form, and sends the request to the Solr server. After the Solr server processes the query, it returns the results which are parsed by the PHP program and formatted for display.

We are using the solr-php-client which is available here https://github.com/PTCInc/solr-php-client . Clone this repository on your computer in the folder where you are developing the User Interface. (**git clone https://github.com/PTCInc/solr-php-client.git**). Below is the sample code from the wiki of this repository.

```
<?php

// make sure browsers see this page as utf-8 encoded HTML
header('Content-Type: text/html; charset=utf-8');

$limit = 10;
$query = isset($_REQUEST['q']) ? $_REQUEST['q'] : false;
```

```
$results = false;

if ($query)
{
 // The Apache Solr Client library should be on the include path
 // which is usually most easily accomplished by placing in the
 // same directory as this script ( . or current directory is a default
 // php include path entry in the php.ini)
 require_once('Apache/Solr/Service.php');

 // create a new solr service instance - host, port, and corename
 // path (all defaults in this example)
 $solr = new Apache_Solr_Service('localhost', 8983, '/solr/core_name/');

 // if magic quotes is enabled then stripslashes will be needed
 if (get_magic_quotes_gpc() == 1)
 {
  $query = stripslashes($query);
 }

 // in production code you'll always want to use a try /catch for any
 // possible exceptions emitted  by searching (i.e. connection
 // problems or a query parsing error)
 try
 {
  $results = $solr->search($query, 0, $limit);
 }
 catch (Exception $e)
 {
  // in production you'd probably log or email this error to an admin
  // and then show a special message to the user but for this example
  // we're going to show the full exception
  die("<html><head><title>SEARCH EXCEPTION</title><body><pre>{$e->__toString()}</pre></body></html>");
 }
}

?>
<html>
 <head>
  <title>PHP Solr Client Example</title>
 </head>
 <body>
  <form  accept-charset="utf-8" method="get">
   <label for="q">Search:</label>
   <input id="q" name="q" type="text" value="<?php echo htmlspecialchars($query, ENT_QUOTES, 'utf-8'); ?>"/>
   <input type="submit"/>
  </form>
<?php

// display results
if ($results)
{
 $total = (int) $results->response->numFound;
 $start = min(1, $total);
 $end = min($limit, $total);
?>
  <div>Results <?php echo $start; ?> - <?php echo $end;?> of <?php echo $total; ?>:</div>
  <ol>
<?php
 // iterate result documents
 foreach ($results->response->docs as $doc)
 {
?>
    <li>
      <table style="border: 1px solid black; text-align: left">
<?php
  // iterate document fields / values
  foreach ($doc as $field => $value)
  {
?>
        <tr>
          <th><?php echo htmlspecialchars($field, ENT_NOQUOTES, 'utf-8'); ?></th>
          <td><?php echo htmlspecialchars($value, ENT_NOQUOTES, 'utf-8'); ?></td>
        </tr>
<?php
  }
?>
      </table>
    </li>
<?php
 }
?>
  </ol>
<?php
}
?>
 </body>
</html>
```

In order to provide additional parameters to the Solr server, create an array of these parameters as shown below:

```
$additionalParameters = array(
  'fq' => 'a filtering query',
  'facet' => 'true',
  // notice I use an array for a multi-valued parameter
  'facet.field' => array(
    'field_1',
    'field_2'
  )
);

$results = $solr->search($query, $start, $rows, $additionalParameters);
```

Step 2

In this step you are going to run a series of queries on the Solr index. The queries you are going to use are the same ones you used for Exercise #1. The comparison now will be to use two different ranking algorithms.

Solr uses Lucene to facilitate ranking. Lucene uses a combination of the Vector Space Model and the Boolean model to determine how relevant a given document is to a user's query. The vector space model is based on term frequency. The Boolean model is first used to narrow down the documents that need to be scored based on the use of Boolean logic in the query specification.

Solr permits us to change the ranking algorithm. This gives us an opportunity to use the Page rank algorithm and see how the results differ. There are several ways to manipulate the ranking of a document in Solr. Here, we explain the method which uses a field that refers to an External File that stores the Page Rank scores. This external file basically contains a mapping from a key field to the field value.

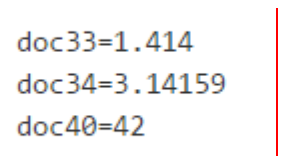
In order to create this External File you must first carry out the Page Rank process on your set of collected web pages. You can use the web graph structure that you created in your assignment #2. There is no need to re-compute the edge relationships (see **pagerankdata.csv**).

here are several libraries available to help you compute the Page Rank given a graph. One of them is the **NetworkX** library –

http://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html

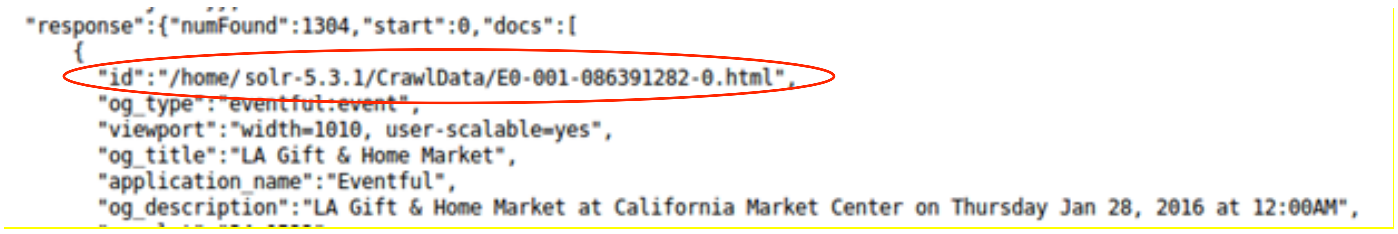
This Page Rank function takes a NetworkX graph (http://networkx.github.io/documentation/networkx-1.10/reference/classes.digraph.html#networkx.DiGraph) as input and returns a dictionary of graph nodes with corresponding Page Rank scores. These are the steps you would need to perform:

i. Compute the incoming and outgoing links to the web pages, and create a NetworkX graph
ii. Compute the Page Rank for this graph and store this in a file in the format <document_id>=<page_rank_score>



```
doc33=1.414
doc34=3.14159
doc40=42
```

Make sure the document id is the same as the one which is present in your index. In your Solr index, you would typically have the filename as the id as shown below:



```
"response":{"numFound":1304,"start":0,"docs":[
    {
      "id":"/home/ solr-5.3.1/CrawlData/E0-001-086391282-0.html",
      "og_type":"eventful:event",
      "viewport":"width=1010, user-scalable=yes",
      "og_title":"LA Gift & Home Market",
      "application_name":"Eventful",
      "og_description":"LA Gift & Home Market at California Market Center on Thursday Jan 28, 2016 at 12:00AM",
```

Once you have computed the Page Rank scores, you need to place this file in the data folder of the core you have defined. This can be found in the path solr-5.3.1/server/solr/core_name/. The name of the file should be external_*fieldname* or external_*fieldname*.*. For this example the file could be named external_pageRankFile or external_pageRankFile.txt.

The next step is to add the field in the schema.xml which refers to this score.

```
<fieldType name="external" keyField="id" defVal="0" class="solr.ExternalFileField" valType="pfloat"/>
<field name=" pageRankFile" type="external" stored="false" indexed="false" />
```

We are defining a field "pageRankFile" which is of the type "external". The field name should be the suffix after "external_" you provided for your file in the previous step. The keyField attribute defines the key that will be defined in the external file. It is usually the unique key for the index. A defVal defines a default value that will be used if there is no entry in the external file for a particular document. The valType attribute specifies the actual type of values that will be found in the file. The type specified must be either a float field type, so valid values for this attribute are pfloat, float or tfloat. This attribute can be omitted.

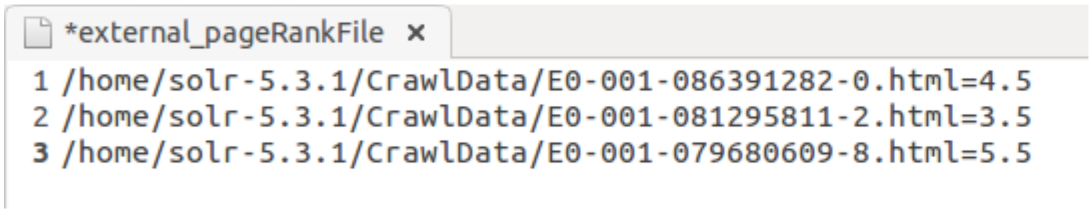
Once the field has been defined, we need to make sure that when the index is reloaded, it is able to access the rank file. In order to do that, there are some modifications required in the solrconfig.xml file. We need to define eventListeners to reload the external file when either a searcher is loaded or when a new searcher is started. The searcher is basically a Lucene class which enables searching across the Lucene index. Define these listeners within the <query> element in the solrconfig.xml file.

```
<listener event="newSearcher" class="org.apache.solr.schema.ExternalFileFieldReloader"/>
<listener event="firstSearcher" class="org.apache.solr.schema.ExternalFileFieldReloader"/>
```

Now reload the index, by going to the Solr Dashboard UI ->Core Admin and clicking on the "Reload" button.

Now, you can run the queries and compare the results with and without page rank scores. In the Solr query view, there is a "sort" text field, where you can specify the field on which the results have to be sorted along with the order (ascending, descending)

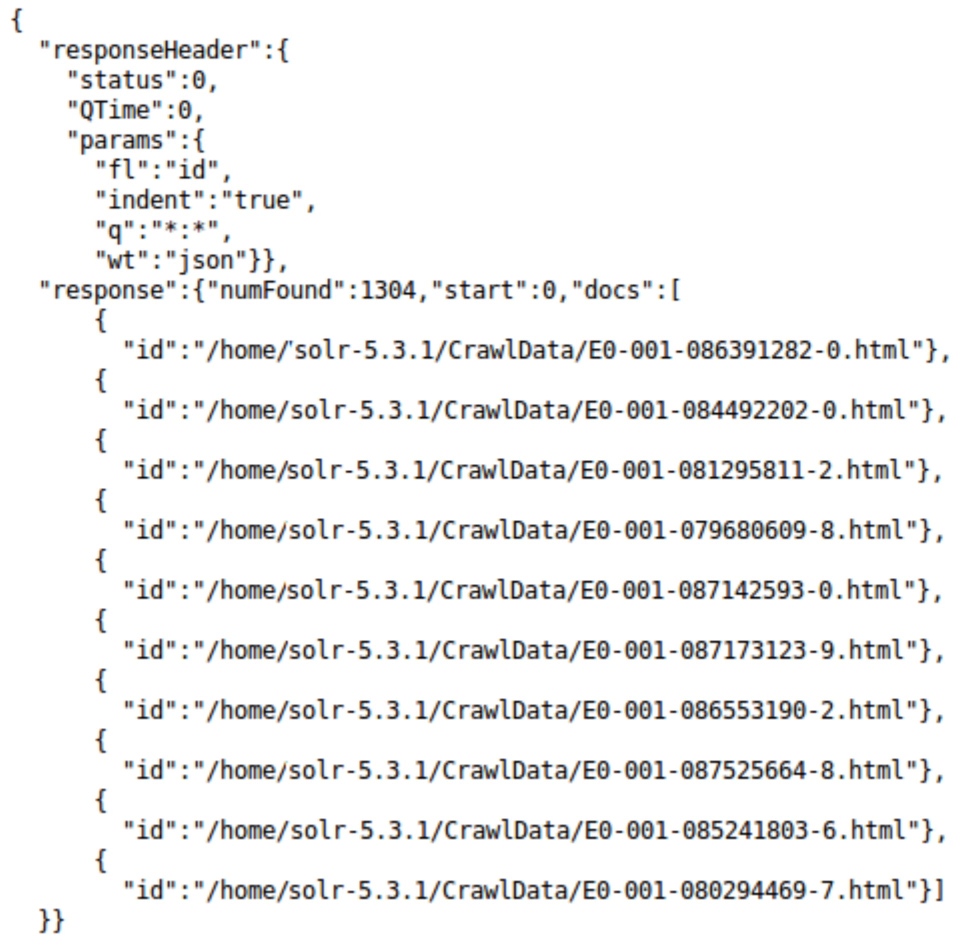
For example, in the below screenshot I have created an external file and added page rank scores for a few documents in the index.



```
*external_pageRankFile
/home/solr-5.3.1/CrawlData/E0-001-086391282-0.html=4.5
/home/solr-5.3.1/CrawlData/E0-001-081295811-2.html=3.5
/home/solr-5.3.1/CrawlData/E0-001-079680609-8.html=5.5
```

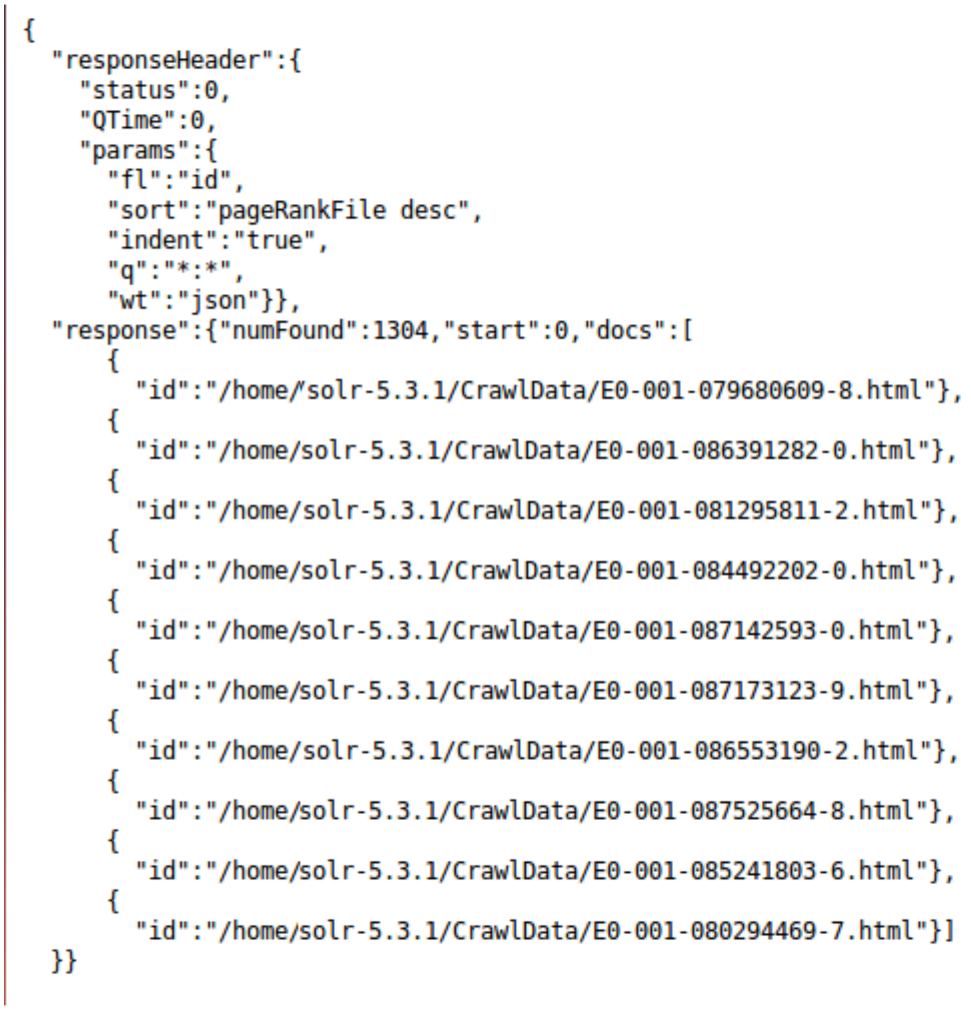
"/home/solr-5.3.1/CrawlData/E0-001-086391282-0.html" is the id of the file that has been indexed. The values indicate the page rank scores.

The following screenshot consists of the results which uses Solr's internal ranking algorithm based on tf-idf for the default query "*:*". For the sake of clarity, only the ids have been displayed.



```
{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "fl":"id",
      "indent":"true",
      "q":"*:*",
      "wt":"json"}},
  "response":{"numFound":1304,"start":0,"docs":[
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-086391282-0.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-084492202-0.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-081295811-2.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-079680609-8.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-087142593-0.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-087173123-9.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-086553190-2.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-087525664-8.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-085241803-6.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-080294469-7.html"}]
  }}
```

The next screenshot is the result of the default query again, but with the ranking based on page rank that was defined in the external file.



```
{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "fl":"id",
      "sort":"pageRankFile desc",
      "indent":"true",
      "q":"*:*",
      "wt":"json"}},
  "response":{"numFound":1304,"start":0,"docs":[
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-079680609-8.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-086391282-0.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-081295811-2.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-084492202-0.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-087142593-0.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-087173123-9.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-086553190-2.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-087525664-8.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-085241803-6.html"},
      {
        "id":"/home/solr-5.3.1/CrawlData/E0-001-080294469-7.html"}]
  }}
```

Comparing the two results we can see that the order of the results has changed. The file ending with "E0-001-079680609-8.html" is the first result in the page rank scoring results, as we had defined a high score(5.5) for it in the external file.

Once that is done you should run the same set of queries from the previous assignment, and save the results.

Step 3

Returning the results:

The results that you return should include the total number of documents found at the top of the page. Each result should include a: title, author, date created, size in KB, and a link that you can click on and be re-directed to the corresponding document.

Comparing the results:

Compare the results of the two ranking algorithms. You can do this by returning to your assignment #1. Use the same queries that you used in assignment #1. For each query collect the results given by both ranking algorithms, and in a document compare them.

What needs to be submitted

There should be a report describing what you have done and analyzing the results of the two ranking algorithms.

Using the **submit** command you should provide the following files:

- the external_PageRankFile.txt that was used as input to the PageRank algorithm
- all source code that you wrote, including the code for creating the web page that accepts the query, the program that sends the query to Solr and the program that processes the Solr results