

# Searching CiteSeer Metadata Using Nutch

Larry Reeve  
for Dr. Lin  
INFO624, Winter 2005  
Term Project

## Table of Contents

Background.....	2
Project Design.....	3
Evaluation.....	10
Future Work.....	15
References.....	17
Appendix A - CiteSeer Metadata Record Example .....	18
Appendix B - Converted HTML Example .....	19
Appendix C - Shell Script to Generate HTML Files from XML Metadata Files .....	20
Appendix D - Parse XML Metadata and Generate HTML.....	21
Appendix E - Shell Script to Crawl the Generated HTML Files .....	24
Appendix F - Nutch Parse Filter for CiteSeer Metadata .....	25
Appendix G - Simple Parser to Split Author Name into Two Parts .....	28
Appendix H - Simple Parser to Split Date Field into Three Parts ...	29
Appendix I - Nutch Indexing Filter for CiteSeer Metadata .....	30
Appendix J - Nutch Query Filter for CiteSeer Metadata.....	32
Appendix K - Plugin Build File for Apache Ant.....	33
Appendix L - Nutch Plugin Directives for CiteSeer Plugin.....	34
Appendix M - JSP Source for Search Application.....	35

## Background

CiteSeer (<http://citeseer.ist.psu.edu/>) is a scientific literature digital library that allows searching for scholarly publications by document or by citations. The search engine used by CiteSeer is based on keyword search. A problem with the keyword approach is that searches may return irrelevant publications. For example, searching for author affiliations may return all publications containing the keyword, which is not what was intended. Another example is searching by year of publication, which will return all publications containing the year value. CiteSeer tries to overcome some of the limitation of the keyword search by providing help on query formulation. The reference site for CiteSeer (<http://smealsearch2.psu.edu/help/help.html>) demonstrates how queries should be expressed in order to be most effective. For example, a query looking for publications by "Kurt Bollacker" could be expressed, quite naturally, as "kurt bollacker". According to the reference site, however, this query will not return all citations to "Kurt Bollacker", but instead only documents where "kurt" is adjacent to "bollacker". The suggested query is to use the last name only, or to list all variants of the author. For example, "k bollacker or kurt bollacker". This query is a result of ambiguity of the search engine in understanding what is being requested.

In order to overcome the limits of the existing CiteSeer keyword search for documents, this project aims to populate the open-source search application Nutch with CiteSeer metadata so that searches for specific types of information, such as author first and last name and author affiliation, can be initiated. The idea is to improve the precision of searches by using fielded searches, a standard feature of Nutch.

Metadata for CiteSeer is available directly from the CiteSeer web site. The metadata conforms to the Open Archives Initiative Protocol for Metadata Harvesting standard. The metadata is downloadable in two formats. The first contains Dublin Core metadata, and the second contains Dublin Core plus additional metadata, such as author name and affiliation. For this project, the Dublin Core Plus dataset will be used. The metadata extracted and placed into Nutch fields are: 1) date of publication: year and month; 2) publication title; 3) author first and last name; and 4) author affiliation.

## Project Design

The tasks for integrating the metadata into Nutch are as follows:

- 1) *Download and install all required software to build and run Nutch.* The development and testing environment was done using Microsoft Windows XP on a Pentium 4 2.8 Mhz HT processor with 1GB of memory and a 60GB hard disk. The required packages that need to be installed to build and test Nutch are Apache Tomcat, Cygwin, Ant, and Nutch itself. The Nutch web site (<http://www.nutch.org/docs/en/tutorial.html>) has a one page tutorial that gives short, but not comprehensive, instructions on how to build, crawl, and search using Nutch. This document was helpful in identifying the necessary software needed to get Nutch up and running.

The Cygwin environment provides Unix commands under Windows. It is primarily used to support shell scripting, as many Nutch facilities are provided using shell scripts. For example, the crawling is done using a shell script that calls Java. While this is not strictly needed, it saves the trouble of converting the shell script to a Windows script. There is no additional value in performing the shell script to Windows script conversion for this project, so the Cygwin environment is used.

The Ant tool is needed to automate the build process. Nutch is made of many components that must be built according to their own rules, and are then integrated together into a single jar file. Nutch provides the build.xml files necessary to build and integrate all of its components. The build.xml files contain Ant instructions for building Nutch.

Apache Tomcat is a servlet container that is used to display the Nutch search page (written using JSP), execute a search using Nutch, and then display the search results. Tomcat was used in the last step of this project, after the crawler had run and the index built.

2) *Split the CiteSeer metadata data files into one publication entry per file.* The Nutch crawler expects to crawl one file at a time. In order to support this model, the CiteSeer metadata files are preprocessed into multiple HTML files. Each CiteSeer metadata file contains 100 document entries. The split process takes a single metadata file and generates 100 HTML documents, each containing the metadata fields that will be used by Nutch for fielded searching. The metadata stored for each entry will be placed in the generated HTML document using META statements. Appendix A shows an example document entry from a CiteSeer metadata file. Appendix B shows the same document entry after it has been converted to HTML. This HTML is what will be crawled by the Nutch crawler. The crawling is done from the local file system, rather than from an HTTP server. It can be done either way; I preferred the file system method because it was more direct on my system.

Nutch has extensive support for HTML, allowing extensions to be easily developed and integrated into the Nutch crawler. The alternative to using HTML-based files is to write a custom crawler that understands the CiteSeer metadata file XML file format. The obvious disadvantage is a longer development cycle. The advantage is that the XML to HTML conversion requires the generation of thousands of small HTML files. Appendix D shows the Java source code used to read a CiteSeer metadata using the XML DOM API, extract the desired metadata, and generate the HTML file for each publication entry in the metadata file.

The split process generates the following META tag fields (shown with sample data):

```
<META name="citeseer.identifier" content="1">  
This tag stores the CiteSeer identifier, which is a unique id that  
CiteSeer uses to identify a publication entry.
```

```
<META name="citeseer.datestamp" content="1993-08-11">  
This tag stores the year, month, and day of the publication.
```

```
<META name="citeseer.dc.title" content="Semantic Interpretation">  
This tag stores the title of the publication.
```

```
<META name="citeseer.author0" content="Gabriele Scheler">
<META name="citeseer.authoraffiliation0"
  content="Institut fur Informatik">
```

These tags stores the authors and author affiliations of the publication. A number suffixed to the end of the tag indicates which author this is. The author number starts at 0 and is incremented with each additional author.

```
<META name="citeseer.dc.identifier"
  content="http://citeseer.ist.psu.edu/1.html">
```

This tag is used to store the URL of the CiteSeer page which contains information about this document. The Nutch crawler will store the URL of the page from the file system, since this is the location where the HTML page is read from. During searching, however, the user does not want the metadata page, but the CiteSeer page containing complete document information. This tag is used to replace the file system URL in the search results page.

A shell script has been written (shown in Appendix E) to call the Java program that performs the file splitting. Since a full indexing of the CiteSeer metadata consisting of 575,000 publication entries was completed as part of the project, the 575,000 generated HTML files are organized into a directory hierarchy. This is to reduce the burden on the file system for creating so many files within a single directory. The first directory level holds 10,000 documents. The second directory level consists of 100 directories holding 100 publication entries each (10,000 total). Figure 1 shows a screen shot with the directory structure. The shell script is responsible for determining the directory a specific CiteSeer metadata file belongs to and then creating the directory structure before calling the XML-to-HTML split program. The CiteSeer metadata organization helps in this process. Each metadata file is named with its starting document number. For example, the first metadata file is named `oai_citeseer1`, the second `oai_citeseer101`, the third `oai_citeseer201`, and so forth. The final step of the shell script is to generate a list of URLs to be crawled. The URLs are the file system path to each generated HTML file. This list is passed as an argument to the Nutch crawler.

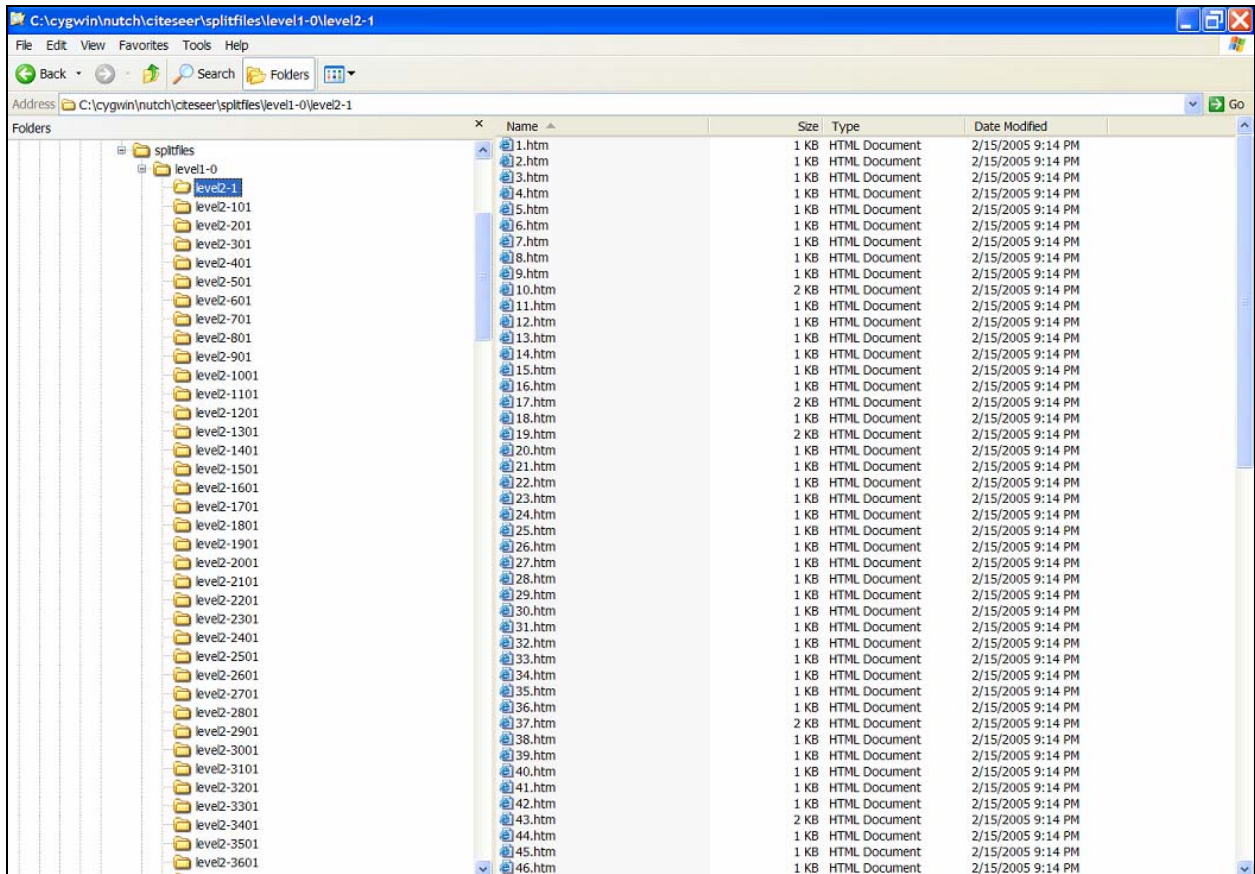


Figure 1: Directory structure organization for generated HTML files

3) Create a Nutch plug-in that contains three filters to handle the metadata now stored in each HTML file. The filters needed to extend Nutch to support CiteSeer metadata are: parsing, indexing, and querying. The parsing and indexing filters are used during the crawling process. The query filter is used during the query process to find CiteSeer-specific query fields and build a query in Nutch internal format. The built query is then executed by Nutch. The three filters are compiled during the build process, and generated into a single jar file. This is discussed more in part 4 below.

The *parsing* filter is used extract metadata from the generated HTML files during crawler processing. It implements the standard Nutch `HtmlParseFilter` interface. The implementation of the filter is straightforward: parse the HTML file using the XML DOM API, find the META

tags, extract the tag values and place them into an internal properties container. Appendix F lists the source code for the parse filter.

- 4) The *indexing* filter takes the metadata stored internally by the parsing filter, and adds it to the Nutch index. At this point, each of the metadata items is associated with a field name. The field name is used when forming a query to specify the field to search on. For this project, full text indexing is not performed, since the point of the project is to look at the precision of doing fielded searches versus full-text. The project is demonstration of metadata searching only, and is expected to show that the use of metadata fields returns more relevant results than searching just by keywords. The source code for the indexing filter is shown in Appendix I.

The following fields are generated during the indexing step:

- title: the title of the publication entry
- cs\_pubyear: the year of publication
- cs\_pubmonth: the month of publication
- cs\_pubday: the day of publication
- cs\_authorfirst: the first name of an author
- cs\_authorlast: the last name of an author
- cs\_authoraaffiliation: the institution an author is affiliated with

Some code manipulation is necessary to make use of the metadata stored during the parsing filter step. In particular, the author and publication date fields must be modified to be useful for searching. The author name is a single field within CiteSeer, so code has been written to split author names into first and last name fields. This is a simplistic split - the author name string is split based on space separator, and the first and last fields from the split are used as the first and last names, respectively. Additional name splitting heuristics can be added to the code as necessary. For example, an author name with "Jacques Le Beau" will incorrectly have the first name of "Jacques" and the last name of "Beau" rather than "Le Beau". The code for performing the author name split is shown in Appendix G. The date field is expressed in CiteSeer in YYYY-MM-DD format. For indexing, the date fields are split into their component year, month, and day values, and indexed separately from each other. The code

for performing the date split is shown in Appendix H. The code implements the standard Nutch IndexingFilter interface.

The *query* filter is written to look for specific field names that have been indexed, such as `title` and `cs_authorfirst`. If these field names are found, they are added to the Nutch internal query format. Nutch builds a chain of these query filters, and calls them one-at-a-time to parse a query string entered by a user. The interface used is the standard Nutch QueryFilter interface. The source code is shown in Appendix J. For this project, all existing query filters were removed from the Tomcat application so that only CiteSeer field names are processed. This eliminates any potential confusion as to what the final query Nutch processes is.

The design of Nutch is such that extending it is a built-in part of its design. This is evident in this project, as the primary coding effort was implementing three standard Nutch interfaces, and customizing their behavior to process CiteSeer metadata.

- 5) *Rebuild Nutch and install the Nutch application within Tomcat, and perform several searches with queries using the metadata fields.*

Figure 2 shows the directory structure used for building Nutch. `nutch-.05` is the root directory of the source tree. The code for this project is located under the `src/plugin/citeseer` directory. The source code is compiled into a single jar file called `citeseer.jar` that contains all three Nutch plugin filters: `parse`, `index`, and `query`.

Nutch is built using the Apache Ant build tool, which incrementally builds the complete source to Nutch, including all plugins. The incremental build allows only those source files which have changed to be recompiled, resulting in faster build times. Ant requires as input an XML file containing a list of instructions on how to build a project. The standard Nutch project includes several Ant build files. The base build file is located in the root directory of the Nutch project (`nutch-.05` in Figure 2). The base build file imports build files in subdirectories off of the root directory. The important sub-build file for this project is in the

*plugin* directory. The *build.xml* file needs to be modified to include the CiteSeer plugin. The modified *plugin/build.xml* file that includes the citeseer plugin information is shown in Appendix K.

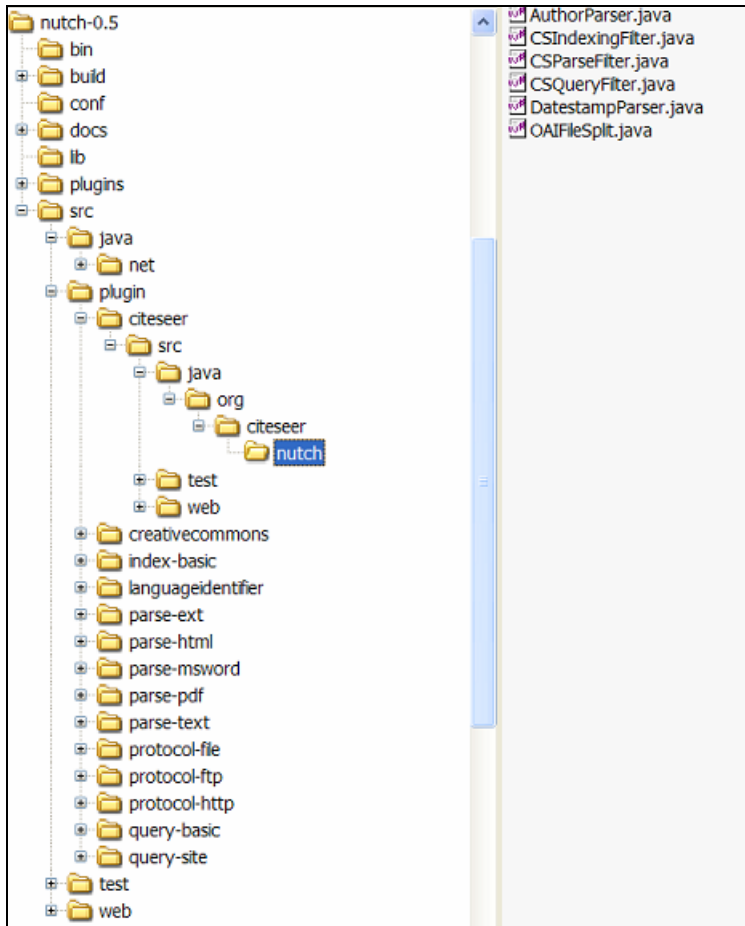


Figure 2: Nutch source directory structure

When Ant is run (from the root folder, using the command `ant`), the CiteSeer plug-in jar file is generated. The single jar file is then copied into the Cygwin area for use during parsing and indexing, also into the Tomcat area for use during searching. A single jar file eases the distribution complexity.

This project was made possible because the architecture of Nutch includes a plug-in facility, which allows for extensions to be easily created and integrated. Once the CiteSeer plug-in jar file is generated and copied to

the appropriate area (Tomcat for searching, Cygwin for crawling), Nutch must be notified that the plug-in is available. This is done by building an XML file that tells Nutch about the CiteSeer plug-ins. This file is shown in Appendix L.

The search application is a single JSP file that calls the CiteSeer plug-in for query evaluation. The JSP file is shown in Appendix M. This source is a modification of the standard Nutch search application. The changes are 1) return only 20 hits instead of 25, to match CiteSeer, and 2) modify the result page so that only the title and URL are shown. The URL has been modified to return not the URL to the crawled document, but to the URL on CiteSeer, so that the user can go directly to the CiteSeer document information page.

At this point, the CiteSeer plug-in jar file has been built, copied into the application areas, and integrated into Nutch using the plugin.xml file. The next step is to run the crawling and indexing step, followed by searching the index.

### Evaluation

The system used to perform the evaluation is a laptop running Windows XP SP2 with 1GB of memory, 60GB of hard disk, and Mobile Pentium 4 2.8 hyper-threaded processor.

To begin the evaluation, the XML-to-HTML conversion of CiteSeer documents is performed. For this I downloaded the entire metadata dataset from CiteSeer, which consisted of 575,000 document entries. It is interesting to note that this is far less than the 716,797 documents advertised on the CiteSeer homepage. It appears the metadata available for download is not kept in sync with the live application. The conversion is done by calling the split shell script, shown in Appendix C. It took the evaluation system 3.5 hours to split 575,000 documents. The output of the split program is 574,900 documents. Since this is exactly 100 documents off, and each XML file contains 100 publication entries, I assumed that one of XML files could not be parsed. Due to the volume of the documents, I was not able to track down which XML file failed.

After the XML-to-HTML CiteSeer documents have been converted, the next step is to crawl and index the generated documents. This is done by using Cygwin to call the crawling/indexing shell script, shown in Appendix E. The shell script is simple, launching the standard Nutch crawler tool with a list of URLs to be crawled.

Three crawls were run with different dataset sizes. One crawl used 100 documents (a single XML file worth of documents), a second used 10,000 documents, and the final used all 574,900 documents. 100 documents took 1-2 minutes to crawl and index. The 10,000 documents took a few more minutes, while the complete crawl of 574,900 documents took 12 hours. So, the generation and crawling of the entire CiteSeer dataset took 15.5 hours!

The screenshot shows the Luke - Lucene Index Toolbox v 0.6 (2005-02-16) interface. The index name is C:\cygwin\nutch\citeseer\crawl-100\index. Statistics include 18 fields, 100 documents, and 2090 terms. The last modified date is Sun Feb 20 18:41:08 EST 2005. The directory implementation is org.apache.lucene.store.FSDirectory.

Available Fields:

- <>
- <anchor>
- <boost>
- <content>
- <cs\_authorfirst>
- <cs\_authorlast>
- <cs\_dcidentifier>
- <cs\_pubday>
- <cs\_pubmonth>
- <cs\_pubyear>
- <cs\_title>
- <digest>
- <docNo>
- <lang>
- <segment>
- <site>
- <title>
- <url>

Top ranking terms. (Right-click for more options)

No	Rank	Field	Text
1	32	<cs_pubyear>	1998
2	25	<cs_pubyear>	1997
3	17	<cs_pubyear>	1995
4	12	<cs_pubyear>	1996
5	5	<cs_pubyear>	1994
6	5	<cs_pubyear>	1999
7	3	<cs_pubyear>	1970
8	1	<cs_pubyear>	1993

Number of top terms: 50

Figure 3: Luke output for 100 documents

To verify the output of the crawl, I used a tool offered by the Apache Lucene project called Luke. Luke examines Lucene index output and reports information about the index. Figure 3 shows the screen capture of Luke running against the 100 document set. It shows 1) that 100 documents have been successfully indexed, and 2) that the CiteSeer custom fields (shown prefixed with cs\_) have been also been added to the index. Interestingly, you can highlight one of the index fields and it will show the top values for the index field. In Figure 3, the publication year field is highlighted. It shows that the top value for publication year is 1998. The Luke tool is useful for gaining insight into the output of the crawling and indexing process. While there were no issues with the 100 and 10,000 document sets, the crawl of 574,900 documents shows that about 551,000 documents were indexed. This means that approximately 24,000 documents were not indexed. This is about 4% of the total number of documents. The crawl/indexing process generates many messages per file crawled, making diagnosing the reason why the crawl failed for these documents failed difficult. I was able to find messages for a few documents. They appear to fail in the code for the custom parse and/or index filters. However, and I crawled the files individually without an error. This leads me to believe that the issue may be related to threading, since the crawler queues the crawl list and has many threads performing crawling and indexing. I added additional defensive programming, such as null checks, but this did not appear to help. To track down the problem, I would need to implement additional logging that generates output separate from the main Nutch log facility, so that the number of output messages is more manageable. The exclusion of these documents does not impact the evaluation results for this project, since the 100 document set is used for evaluation. The complete load is more to test what would happen if the entire document set is loaded.

To evaluate the precision of the fielded searches, two approaches were used. Both approaches use the standard measures of recall and precision, where  $\text{Precision} = (\text{Retrieved} \ \& \ \text{Relevant}) / (\text{All Retrieved})$ , and  $\text{Recall} = (\text{Retrieved} \ \& \ \text{Relevant}) / (\text{All Relevant})$ . The approaches are:

- 1) The first XML file (documents 1 to 100) from the CiteSeer metadata download were used. This is done to keep the dataset manageable so that recall can be measured. Four queries were issued, and recall and precision were measured for each.

The queries are:

- a. Look for all papers where Peter Lee is an author.
  - i. `cs_authorlast:lee`
  - ii. `cs_authorlast:lee cs_authorfirst:peter`
- b. Look for all papers from with authors from Carnegie Mellon
  - i. `cs_affiliation:Carnegie`

For query (a):

There are 3 documents that have authors with last name of Lee, but only one has the author of Peter Lee. Using `cs_authorlast:lee` as the query, Nutch returns the correct 3 documents. The precision for using only the last name field is 1/3, while the recall is 1/1. Using `(cs_authorlast:lee cs_authorfirst:lee)` as the query, Nutch returns the correct single document. The precision for using both the last name and first name fields is 1/1=1, while the recall is 1/1=1. For this query, adding more fields increases the precision.

For query (b):

There are 6 documents that have authors with an affiliation to Carnegie. Using the query `(cs_authoraaffiliation:Carnegie)`, all six of the documents are returned. This leads to a precision of 6/6=1 and a recall of 6/6=1.

2) Author first and last name queries are issued against both the project system and the CiteSeer system. The project system has approximately 551,000 documents indexed, while CiteSeer advertises having 716,797 documents indexed. Average precision is measured by using fixed intervals of retrieved documents. The intervals are set to 20, since this is the number of documents returned by both CiteSeer and the project system for each page of hits.

The queries are:

- Q1: Find all papers by author Peter Lee
- Project system: `cs_authorfirst:peter cs_authorlast:lee`
- CiteSeer: `peter w/2 lee`

Q2: Find all papers by author Jeffrey Ullman

Project system: cs\_authorfirst:jeffrey cs\_authorlast:ullman

CiteSeer: jeffrey w/2 ullman

Q3: Find all papers by author John Smith

Project system: cs\_authorfirst:john cs\_authorlast:smith

CiteSeer: john w/2 smith

Number of relevant documents

CiteSeer	Query 1	Query 2	Query 3	Avg Precision
N=20	13	18	10	.68
N=40	27	36	17	.67
N=60	39	51	26	.64

N=number of documents retrieved

Number of relevant documents

Project	Query 1	Query 2	Query 3	Avg Precision
N=20	18	20	13	.85
N=40	37	40	26	.86
N=60	53	60	43	.87

N=number of documents retrieved

### Future Work:

The project demonstrates that fielded search can improve precision of searches. It can also be extended in many ways. First, there are a few problems that need to be resolved. As mentioned in the Evaluation section, there are several technical problems that need to be addressed to allow the complete CiteSeer dataset to be loaded. The split has one XML file that could not be parsed. The crawl/index has issues with about 4% of the document population. This means that not all files were generated, and of all files generated, not all were indexed. These are most likely some failure in the implementation of the split and parsing/indexing filters, or there interaction with the Nutch plugin system.

There are also issues that came up during the evaluation of fielded search. For example, when searching authors, there is no way force the first and last names of the author to not match, for instance, the first name of author #1 and the last name of author #2. That is, there is no way to force the author first and last names to be together. During the evaluation, this effect was pronounced the more common the first and last names are. If the first and last names are not very common, there is a small possibility to have author first and last names so close within a publication. However, having both a common first and last name dramatically increases the probability. This was demonstrated in the evaluation. Performing an author search for 'John Smith' results in a precision of  $P=.65$ , but an author search for 'Jeffrey Ullman' results in a precision of  $P=1$ . Another related issue that came up is with the splitting of author names. The existing filter splits the author name into first and last using space separators. Additional heuristics would need to be implemented to handle the variations of author names, particularly first and last names composed of more than one word.

To get the most up-to-date list of documents, the XML metadata file approach cannot be used, since it appears the CiteSeer system does not keep its contents current. There are two other approaches that might be used. The first is to use the Open Archives Initiative protocol, which is a web service provided by CiteSeer that will provide information about documents. The second method is to directly crawl the CiteSeer web pages for each document.

The system can also be extended with additional functionality. For example, it might be useful to search for documents having a minimum number of

citation counts so that only the most popular papers are retrieved. I could not find a way to specify a greater-than operator in the query filter for a field, but such a facility probably exists within Nutch (and its subsystem Lucene). Additional work could also provide other ways to search citation information. The use of citation information was not explored in this project.

Overall, the project showed that fielded search can improve search precision with only a few fields available. It also demonstrated that the application can be scaled to handle tens of thousands of documents. The remaining issues to put the application into production are mostly technical issues in the split and crawling/indexing process. It would be easy to modify the source to change the fields indexed or add new ones. The base functionality has been implemented.

References:

Apache Ant. Retrieved March 3, 2005, from <http://ant.apache.org>.

Apache Lucene. Retrieved March 3, 2005, from <http://lucene.apache.org>.

Apache Nutch. Retrieved March 3, 2005, from <http://www.nutch.org>.

Apache Tomcat. Retrieved March 3, 2005, from  
<http://jakarta.apache.org/tomcat/>.

CiteSeer. Retrieved March 3, 2005, from <http://citeseer.ist.psu.edu/cs>.

Khare, R., Cutting, D., Sitaker, K., and Rifkin, A. (2005). *Nutch: A Flexible and Scalable Open Source Search Engine*. Commerce Labs Technical Report. Retrieved March 3, 2005, from <http://labs.commerce.net/wiki/images/0/06/CN-TR-04-04.pdf>.

Giles, C. L., Bollacker, K., and Lawrence, S. (1998). *CiteSeer: An Automatic Citation Indexing System*. The Third ACM Conference on Digital Libraries. Retrieved March 3, 2005, from  
<http://citeseer.ist.psu.edu/giles98citeseer.html>.

Appendix A: CiteSeer Metadata Record Example  
(Dublin Core Standard plus additional fields)

```
<record>
<header>
<identifier>oai:CiteSeerPSU:1</identifier>
<datestamp>1993-08-11</datestamp>
<setSpec>CiteSeerPSUset</setSpec>
</header>
<metadata>
<oai_citeseer:oai_citeseer
xmlns:oai_citeseer="http://copper.ist.psu.edu/oai/oai_citeseer/" xmlns:dc
="http://purl.org/dc/elements/1.1/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://copper.ist.psu.edu/oai/oai_citeseer/
http://copper.ist.psu.edu/oai/oai_citeseer.xsd ">
  <dc:title>36 Problems for Semantic Interpretation</dc:title>
  <oai_citeseer:author name="Gabriele Scheler">
    <address>80290 Munchen , Germany</address>
    <affiliation>Institut fur Informatik; Technische Universitat
Munchen</affiliation>
  </oai_citeseer:author>
  <dc:subject>Gabriele Scheler 36 Problems for Semantic
Interpretation</dc:subject>
  <dc:description>This paper presents a collection of problems for natural
language analysis...
</dc:description>
  <dc:contributor>The Pennsylvania State University CiteSeer
Archives</dc:contributor>
  <dc:publisher>unknown</dc:publisher>
  <dc:date>1993-08-11</dc:date>
  <dc:format>ps</dc:format>
  <dc:identifier>http://citeseer.ist.psu.edu/1.html</dc:identifier>
  <dc:source>ftp://flop.informatik.tu-muenchen.de/pub/fki/fki-179-
93.ps.gz</dc:source>
  <dc:language>en</dc:language>
  <dc:rights>unrestricted</dc:rights>
</oai_citeseer:oai_citeseer>
</metadata>
</record>
```

## Appendix B: Converted HTML Example

```
<HEAD>
<TITLE>CiteSeer Document - 1</TITLE>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
<META name="citeseer.identifier" content="1">
<META name="citeseer.datestamp" content="1993-08-11">
<META name="citeseer.dc.title" content="36 Problems for Semantic
Interpretation">
<META name="citeseer.author0" content="Gabriele Scheler">
<META name="citeseer.authoraffiliation0" content="Institut fur Informatik;
Technische Universitat Munchen">
<META name="citeseer.dc.identifier"
content="http://citeseer.ist.psu.edu/1.html">
</HEAD>
<BODY>
</BODY>
</HTML>
```

## Appendix C: Shell Script to Generate HTML Files from XML Metadata Files

```
#!/bin/sh
#
#   Larry Reeve
#   INFO624 - Dr. Lin - Winter 2005
#
#   shsplit
#
#   This script will split the CiteSeer metadata files into separate HTML files
#   - The organization of the final tree is as follows:
#       Level1 = 10000 docs
#       Level2 = 100 folders of 100 documents each
#
FileCounter=1
while [ $FileCounter -lt 574900 ]
do

    Level1=`expr $FileCounter / 10000`
    Level2=`expr $FileCounter % 10000`
    DirectoryName=splitfiles/level1-`${Level1}`/level2-`${Level2}`

    echo "Processing $FileCounter into $DirectoryName ..."

    echo "  Generating XML file..."
    cat XMLPrefix.txt                                > CiteseerXML.txt
    cat datafiles/oai_citeseer$FileCounter           >> CiteseerXML.txt
    cat XMLSuffix.txt                                >> CiteseerXML.txt

    echo "  Creating directory..."
    mkdir -p $DirectoryName

    echo "  Splitting XML file..."
    java -cp citeseer.jar org.citeseer.nutch.OAIFileSplit CiteseerXML.txt $DirectoryName

    echo "  Generating URL list..."

    UrlIndexCounterFrom=$FileCounter
    UrlIndexCounterTo=`expr $FileCounter + 100`
    while [ $UrlIndexCounterFrom -lt $UrlIndexCounterTo ]
    do
        echo "file://`${DirectoryName}`/`${UrlIndexCounterFrom}`.htm" >> urls-all.txt
        UrlIndexCounterFrom=`expr $UrlIndexCounterFrom + 1`
    done

    FileCounter=`expr $FileCounter + 100`
done
echo
```

## Appendix D: Parse XML Metadata and Generate HTML

```
/* Larry Reeve */
/* INFO624 - Dr. Lin - Winter 2005 */

package org.citeseer.nutch;

import java.util.*;
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

/** Splits a Open Archives Initiative file from CiteSeer into multiple HTML files for crawling */
public class OAIFileSplit {

    /** Splits a Open Archives Initiative file from CiteSeer */
    public static void main(String[] args) throws Exception {

        if (args.length != 2) {
            System.err.println("Usage: OAIFileSplit <OAIFilename> <OutputDirectory>");
            return;
        }

        String paramOAIFilename = args[0];
        String paramOutputDirectory = args[1];

        File OAIFile = new File(paramOAIFilename);
        File OAIDirectory = new File(paramOutputDirectory);

        if (!OAIFile.exists())
            throw new FileNotFoundException("Expected file: " + paramOAIFilename);

        if (!OAIDirectory.exists())
            throw new FileNotFoundException("Expected directory: " + paramOutputDirectory);

        splitCiteSeerFile(paramOAIFilename, paramOutputDirectory);
    }

    public static void splitCiteSeerFile(String paramOAIFilename, String paramOutputDirectory) {
        try {
            // create new parser instance using DOM
            DocumentBuilderFactory xmlDocBuilder = DocumentBuilderFactory.newInstance();

            DocumentBuilder xmlParser = xmlDocBuilder.newDocumentBuilder();

            // parse the XML file into a DOM tree
            Document xmlDocument = xmlParser.parse(paramOAIFilename);

            // parse the DOM tree containing all of the document metadata
            // get list of all document nodes in root
            NodeList recordNodes = xmlDocument.getElementsByTagName("record");
            for (int recordNodesIdx=0; recordNodesIdx < recordNodes.getLength(); recordNodesIdx++) {

                // Initialize field values
                String fieldCSIdentifier = null;
                String fieldCSDateStamp = null;
                String fieldDCIdentifier = null;
                String fieldDCTitle = null;
                String fieldCSAuthor = null;
                String fieldCSAuthorAffiliation = null;

                // Get next node from XML that is a record element
                Element recordNode = (Element) recordNodes.item(recordNodesIdx);

                // header
                NodeList headerNodes = recordNode.getElementsByTagName("header");
                Element headerNode = (Element) headerNodes.item(0);
            }
        }
    }
}
```

```

// header.identifier
NodeList identifierNodes = headerNode.getElementsByTagName("identifier");
Element identifierNode = (Element) identifierNodes.item(0);
fieldCSIdentifier = getCSFieldIdentifierNumber(getNodeTextValue(identifierNode));

// Have the identifier, starting writing out a HTML file containing metadata values
String fileName = paramOutputDirectory + File.separator + fieldCSIdentifier.toString() +
".htm";
PrintWriter htmlWriter = new PrintWriter(new BufferedWriter(new FileWriter(fileName)));
htmlWriter.println("<HEAD>");
htmlWriter.println("<TITLE>CiteSeer Document - " + fieldCSIdentifier + "</TITLE>");
htmlWriter.println("<META http-equiv=Content-Type content=\"text/html; charset=iso-8859-1\">");
htmlWriter.println("<META name=\"citeseer.identifier\" content=\"" + fieldCSIdentifier +
\">");

// header.datestamp
NodeList datestampNodes = headerNode.getElementsByTagName("datestamp");
Element datestampNode = (Element) datestampNodes.item(0);
fieldCSDateStamp = getNodeTextValue(datestampNode);
//System.out.println(" datestamp: " + fieldCSDateStamp);
htmlWriter.println("<META name=\"citeseer.datestamp\" content=\"" + fieldCSDateStamp +
\">");

// metadata
NodeList metadataNodes = recordNode.getElementsByTagName("metadata");
Element metadataNode = (Element) metadataNodes.item(0);

// metadata.dc.title
NodeList dcTitleNodes = metadataNode.getElementsByTagName("dc:title");
Element dcTitleNode = (Element) dcTitleNodes.item(0);
fieldDCTitle = getNodeTextValue(dcTitleNode);
//System.out.println(" dc.title: " + fieldDCTitle);
htmlWriter.println("<META name=\"citeseer.dc.title\" content=\"" + fieldDCTitle + "\">");

// metadata.citeseer.author
NodeList csAuthorNodes = metadataNode.getElementsByTagName("oai_citeseer:author");
for (int authorIdx=0; authorIdx < csAuthorNodes.getLength(); authorIdx++) {
    Element csAuthorNode = (Element) csAuthorNodes.item(authorIdx);
    fieldCSAuthor = getNodeTextValue(csAuthorNode.getAttributeNode("name"));
    htmlWriter.println("<META name=\"citeseer.author" + authorIdx + "\" content=\"" +
fieldCSAuthor + "\">");

    // metadata.citeseer.author.affiliation (optional)
    NodeList csAuthorAffiliationNodes = csAuthorNode.getElementsByTagName("affiliation");
    if (csAuthorAffiliationNodes.getLength() > 0) {
        Element csAuthorAffiliationNode = (Element) csAuthorAffiliationNodes.item(0);
        fieldCSAuthorAffiliation = getNodeTextValue(csAuthorAffiliationNode);
    }
    // always write affiliation if there is an author, even if it is empty; this simplifies
the parse filter logic
    if (fieldCSAuthorAffiliation == null)
        fieldCSAuthorAffiliation = "";
    htmlWriter.println("<META name=\"citeseer.authoraffiliation" + authorIdx + "\"
content=\"" + fieldCSAuthorAffiliation + "\">");
}

// metadata.dc.identifier
NodeList dcIdentifierNodes = metadataNode.getElementsByTagName("dc:identifier");
Element dcIdentifierNode = (Element) dcIdentifierNodes.item(0);
fieldDCIdentifier = getNodeTextValue(dcIdentifierNode);
htmlWriter.println("<META name=\"citeseer.dc.identifier\" content=\"" + fieldDCIdentifier
+ "\">");

// Append closing tags to output file
htmlWriter.println("</HEAD>");
htmlWriter.println("<BODY>");
htmlWriter.println("</BODY>");
htmlWriter.println("</HTML>");
htmlWriter.close();

```

```

    }
}
// Catch any exceptions thrown
catch(Exception e) {
    System.err.println(e);
    e.printStackTrace();
}
}

// A helper method that will examine a node's children looking for
// text nodes, and will construct a single string containing all
// child text node values.
private static String getNodeTextValue(Node nodeToGetContentsFor) {
    NodeList childNodes = nodeToGetContentsFor.getChildNodes();

    // if there are no child nodes then return empty string
    if (childNodes.getLength() == 0) {
        return "";
    }

    // if there is only 1 child and its a text node, then return the node's value
    else if (childNodes.getLength() == 1 && childNodes.item(0).getNodeTypes() == Node.TEXT_NODE) {
        return childNodes.item(0).getNodeValue();
    }

    // if there are multiple children, then use a StringBuffer to build
    // a single string value from possibly multiple children.
    else {
        StringBuffer textValue = new StringBuffer();

        for (int idx=0; idx < childNodes.getLength(); idx++) {
            // Search for text nodes
            if (childNodes.item(idx).getNodeTypes() == Node.TEXT_NODE) {
                // Get the text value of the text node and append it
                // to the string buffer, dynamically building a single
                // string from 0 or more multiple child elements.
                textValue.append(childNodes.item(idx).getNodeValue());
            }
        }

        // return the string that has been constructed
        return textValue.toString();
    }
}

// Extract the identifier number from the Open Archives identifier string
// The identifier looks like oai:CiteSeerPSU:1
private static String getCSFieldIdentifierNumber(String identifier) {
    int lastColon = identifier.lastIndexOf(':');
    return identifier.substring(lastColon+1);
}
}

```

## Appendix E: Shell Script to Crawl the Generated HTML Files

```
#!/bin/sh
#
#       Larry Reeve
#       INFO624 - Dr. Lin - Winter 2005
#
#       shcrawl
#
#       This script will run the Nutch crawler against the set of urls specified in the urls.txt
file
#       - it should be run from citeseer directory
#
JAVA_HOME=/cygdrive/c/j2sdk1.4.2_05
export JAVA_HOME
DateStart=`date`
../bin/nutch crawl urls-all.txt -dir crawl -depth 1
DateEnd=`date`
echo "Start: $DateStart"
echo "End:   $DateEnd"
```

## Appendix F: Nutch Parse Filter for CiteSeer Metadata

```
/* Larry Reeve */
/* INFO624 - Dr. Lin - Winter 2005 */

package org.citeseer.nutch;

import net.nutch.parse.*;
import net.nutch.protocol.Content;
import net.nutch.util.NutchConf;

import java.util.*;
import java.io.*;
import java.net.*;
import javax.xml.parsers.*;
import org.xml.sax.InputSource;
import org.w3c.dom.*;

import java.util.logging.Logger;
import net.nutch.util.LogFormatter;

/** Adds metadata from parsing CiteSeer metadata generated HTML files */
public class CSParseFilter implements HtmlParseFilter {
    private static final Logger LOG = LogFormatter.getLogger(CSParseFilter.class.getName());

    /** Adds metadata from the DOM tree of a HTML page. */
    public Parse filter(Content content, Parse parse, DocumentFragment doc)
        throws ParseException {

        // storage for metadata values
        String metaValueCSIdentifier = null;
        String metaValueCSDatestamp = null;
        String metaValueDCTitle = null;
        String metaValueDCIdentifier = null;
        ArrayList metaValueDCAuthors = new ArrayList();
        ArrayList metaValueCSAffiliations = new ArrayList();
        int authorIndex = 0;
        int affiliationIndex = 0;

        // check that contentType is the type expected
        String contentType = content.getContentType();
        if (contentType == null || !contentType.startsWith("text/html"))
            throw new ParseException("Content-Type not text/html: " + contentType);

        try {
            // Extract metadata from pre-defined fields
            NodeList topLevelNodes = doc.getChildNodes();
            for (int topLevelIdx = 0; topLevelNodes != null && topLevelIdx < topLevelNodes.getLength();
                topLevelIdx++) {

                if (topLevelNodes.item(topLevelIdx).getNodeName() != null) {
                    if (topLevelNodes.item(topLevelIdx).getNodeName().equalsIgnoreCase("head")) {
                        NodeList headNodes = topLevelNodes.item(topLevelIdx).getChildNodes();

                        if (headNodes != null) {
                            for (int headNodesIdx=0; headNodesIdx < headNodes.getLength(); headNodesIdx++) {
                                if (headNodes.item(headNodesIdx).getNodeName().equalsIgnoreCase("meta")) {
                                    Element metaNode = (Element) headNodes.item(headNodesIdx);

                                    String metaName = metaNode.getAttribute("name");

                                    if (metaName != null) {
                                        //LOG.info("found meta " + metaName + "=" +
                                        metaNode.getAttribute("content"));

                                        if ("citeSeer.identifier".equalsIgnoreCase(metaName)) {
                                            metaValueCSIdentifier = metaNode.getAttribute("content");
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```



```

metadata.put("DCIdentifier", metaValueDCIdentifier);

// author
for (int idx=0; idx < metaValueDCAuthors.size(); idx++) {
    metadata.put("DCAuthor" + Integer.toString(idx), (String) metaValueDCAuthors.get(idx));
}

// author affiliation
for (int idx=0; idx < metaValueCSAffiliations.size(); idx++) {
    metadata.put("CSAuthorAffiliation" + Integer.toString(idx), (String)
metaValueCSAffiliations.get(idx));
}

// Construct a new ParseData with new title, and additional metadata
ParseData parseData = new ParseData(metaValueDCTitle, outlinks, metadata);
if (parseData == null) {
    throw new ParseException("Unable to create new ParseData for '" +
parse.getData().getTitle() + "'");
}

// No text will be indexed (""), only the metadata
return new ParseImpl("", parseData);
}
catch(Exception e) {
    LOG.info("CSParseFilter exception: " + e.getMessage());
    throw new ParseException(e.getMessage());
}
}
}
}

```

## Appendix G: Simple Parser to Split Author Name into Two Parts

```
/* Larry Reeve */
/* INFO624 - Dr. Lin - Winter 2005 */

package org.citeseer.nutch;

import java.util.logging.Logger;
import net.nutch.util.LogFormatter;

import java.util.*;

/** Parses an author name into first and last name */
public class AuthorParser
{
    private static final Logger LOG = LogFormatter.getLogger(AuthorParser.class.getName());

    private static String authorFirst = "";
    private static String authorLast = "";

    /** Parse a single string containing a full author name into first and last names */
    public static void Parse(String author) {
        if (author == null) {
            LOG.info("author parse failed: author is null");
            return;
        }

        // Split string into parts based on spaces
        String[] authorParts = author.split(" ");
        if (authorParts.length < 2) {
            LOG.info("author parse failed: split does not have enough parts: " + author + "(" +
authorParts.length + ")");
            return;
        }

        /** First name is assumed to be first substring */
        authorFirst = authorParts[0];

        /** Last name is assumed to be last substring */
        authorLast = authorParts[authorParts.length-1];
    }

    /** Property getter for the first name field */
    public static String getFirstName() {
        return authorFirst;
    }

    /** Property getter for last name field */
    public static String getLastName() {
        return authorLast;
    }
}
```

## Appendix H: Simple Parser to Split Date Field into Three Parts

```
/* Larry Reeve */
/* INFO624 - Dr. Lin - Winter 2005 */

package org.citeseer.nutch;

import java.util.logging.Logger;
import net.nutch.util.LogFormatter;

import java.util.*;

/** Parse a date field that is in format YYYY-MM-DD into its 3 parts */
public class DatestampParser {
    private static final Logger LOG = LogFormatter.getLogger(DatestampParser.class.getName());

    private static String partYear = "";
    private static String partMonth = "";
    private static String partDay = "";

    /** Parse a date string in YYYY-MM-DD format into 3 parts and store */
    public static void Parse(String datestamp) {
        if (datestamp == null) {
            LOG.info("datestamp parse failed: datestamp is null");
            return;
        }

        String[] datestampParts = datestamp.split("-");
        if (datestampParts.length != 3) {
            LOG.info("datestamp parse failed: split does not have enough parts: " + datestamp + "(" +
datestampParts.length + ")");
            return;
        }

        partYear = datestampParts[0];
        partMonth = datestampParts[1];
        partDay = datestampParts[2];
    }

    /** Property getter for Year field */
    public static String getYear() {
        return partYear;
    }

    /** Property getter for Month field */
    public static String getMonth() {
        return partMonth;
    }

    /** Property getter for Day field */
    public static String getDay() {
        return partDay;
    }
}
```

## Appendix I: Nutch Indexing Filter for CiteSeer Metadata

```
/* Larry Reeve */
/* INFO624 - Dr. Lin - Winter 2005 */

package org.citeseer.nutch;

import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;

import net.nutch.parse.Parse;

import net.nutch.indexer.IndexingFilter;
import net.nutch.indexer.IndexingException;

import net.nutch.fetcher.FetcherOutput;
import net.nutch.pagedb.FetchListEntry;

import java.util.logging.Logger;
import net.nutch.util.LogFormatter;

import java.util.*;
import java.net.URL;
import java.net.MalformedURLException;

/** Adds searchable fields to a document. */
public class CSIndexingFilter implements IndexingFilter {
    private static final Logger LOG = LogFormatter.getLogger(CSIndexingFilter.class.getName());

    public Document filter(Document doc, Parse parse, FetcherOutput fo)
        throws IndexingException {
        // Index the metadata generated by the citeseer parse filter
        // in the format cs_<metaname>=

        // Title
        String metaTitle = parse.getData().get("DCTitle");
        if (metaTitle != null) {
            doc.add(Field.Text("title", metaTitle));
        }

        // Datestamp
        String metaDatestamp = parse.getData().get("CSDatestamp");
        if (metaDatestamp != null) {
            DatestampParser.Parse(metaDatestamp);
            doc.add(Field.Keyword("cs_pubyear", DatestampParser.getYear()));
            doc.add(Field.Keyword("cs_pubmonth", DatestampParser.getMonth()));
            doc.add(Field.Keyword("cs_pubday", DatestampParser.getDay()));
        }

        // DC Identifier (URL to CiteSeer web page)
        String dcIdentifier = parse.getData().get("DCIdentifier");
        doc.add(Field.UnIndexed("cs_dcidentifier", dcIdentifier));

        // Author list
        for (int idx=0; idx < 10; idx++) {
            String metaAuthorName = parse.getData().get("DCAuthor" + Integer.toString(idx));

            if (metaAuthorName == null) // a null author name indicates end of list
                break;

            AuthorParser.Parse(metaAuthorName);

            doc.add(Field.Keyword("cs_authorfirst", AuthorParser.getFirstName().toLowerCase()));
            doc.add(Field.Keyword("cs_authorlast", AuthorParser.getLastName().toLowerCase()));

            // Author affiliation
            String metaAuthorAffiliation = parse.getData().get("CSAuthorAffiliation" +
                Integer.toString(idx));
        }
    }
}
```

```
        if (metaAuthorAffiliation != null) {  
            doc.add(Field.Text("cs_authoraffiliation", metaAuthorAffiliation));  
        }  
    }  
    return doc;  
}
```

## Appendix J: Nutch Query Filter for CiteSeer Metadata

```
/* Larry Reeve */
/* INFO624 - Dr. Lin - Winter 2005 */

package org.citeseer.nutch;

import java.util.logging.Logger;
import net.nutch.util.LogFormatter;

import org.apache.lucene.search.BooleanQuery;
import org.apache.lucene.search.TermQuery;
import org.apache.lucene.index.Term;

import net.nutch.searcher.Query;
import net.nutch.searcher.Query.*;
import net.nutch.searcher.QueryFilter;

/** Adds a query filter for CiteSeer fields (prefixed with cs_) */
public class CSQueryFilter implements QueryFilter
{
    private static final Logger LOG
=
LogFormatter.getLogger(CSIndexingFilter.class.getName());
    private static final String FIELD_PREFIX = "cs_";

    public BooleanQuery filter(Query input, BooleanQuery output) {
        // examine each clause in the Nutch query
        Clause[] clauses = input.getClauses();
        for (int i = 0; i < clauses.length; i++)
        {
            Clause c = clauses[i];

            //LOG.info("csquery=" + input.toString() + ", field=" + c.getField() + ", " + FIELD);

            // skip non-matching clauses
            if (!c.getField().equalsIgnoreCase("title") && !c.getField().startsWith(FIELD_PREFIX))
                continue;

            //LOG.info("query clause match");

            // get the field value from the clause
            // raw fields are guaranteed to be Terms, not Phrases
            String value = c.getTerm().toString();
            value = value.toLowerCase();

            // add a Lucene TermQuery for this clause
            TermQuery clause = new TermQuery(new Term(c.getField(), value));

            // set boost to zero, so that it does not affect ranking
            clause.setBoost(1.0f);

            // add it as specified in query
            output.add(clause, c.isRequired(), c.isProhibited());
        }

        LOG.info("csfinal query=" + output.toString());

        // return the modified Lucene query
        return output;
    }
}
```

## Appendix K: Plugin Build File for Apache Ant

```
<?xml version="1.0"?>
<project name="Nutch" default="deploy" basedir=".">
  <!-- ===== -->
  <!-- Build & deploy all the plugin jars. -->
  <!-- ===== -->
  <target name="deploy">
    <ant dir="protocol-file" target="deploy"/>
    <ant dir="protocol-ftp" target="deploy"/>
    <ant dir="protocol-http" target="deploy"/>
    <ant dir="parse-html" target="deploy"/>
    <ant dir="parse-text" target="deploy"/>
    <ant dir="parse-pdf" target="deploy"/>
    <ant dir="parse-msword" target="deploy"/>
    <ant dir="parse-ext" target="deploy"/>
    <ant dir="index-basic" target="deploy"/>
    <ant dir="query-basic" target="deploy"/>
    <ant dir="query-site" target="deploy"/>
    <ant dir="creativecommons" target="deploy"/>
    <ant dir="citeseer" target="deploy"/>
    <ant dir="languageidentifier" target="deploy"/>
  </target>

  <!-- ===== -->
  <!-- Test all of the plugins. -->
  <!-- ===== -->
  <target name="test">
    <ant dir="protocol-http" target="test"/>
    <ant dir="parse-html" target="test"/>
    <ant dir="parse-pdf" target="test"/>
    <ant dir="parse-msword" target="test"/>
    <ant dir="parse-ext" target="test"/>
    <ant dir="creativecommons" target="test"/>
  </target>

  <!-- ===== -->
  <!-- Clean all of the plugins. -->
  <!-- ===== -->
  <target name="clean">
    <ant dir="protocol-file" target="clean"/>
    <ant dir="protocol-ftp" target="clean"/>
    <ant dir="protocol-http" target="clean"/>
    <ant dir="parse-html" target="clean"/>
    <ant dir="parse-text" target="clean"/>
    <ant dir="parse-pdf" target="clean"/>
    <ant dir="parse-msword" target="clean"/>
    <ant dir="parse-ext" target="clean"/>
    <ant dir="index-basic" target="clean"/>
    <ant dir="query-basic" target="clean"/>
    <ant dir="query-site" target="clean"/>
    <ant dir="creativecommons" target="clean"/>
    <ant dir="citeseer" target="clean"/>
    <ant dir="languageidentifier" target="clean"/>
  </target>
</project>
```

## Appendix L: Nutch Plugin Directives for CiteSeer Plugin

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="citereer"
  name="CiteSeer Plugins"
  version="1.0.0"
  provider-name="nutch.org">

  <extension-point
    id="net.nutch.parse.HtmlParseFilter"
    name="HTML Parse Filter"/>

  <extension-point
    id="net.nutch.indexer.IndexingFilter"
    name="Nutch Indexing Filter"/>

  <extension-point
    id="net.nutch.searcher.QueryFilter"
    name="Nutch Query Filter"/>

  <runtime>
    <library name="citereer.jar">
      <export name="*" />
    </library>
  </runtime>

  <extension id="org.citereer.nutch.CSParseFilter"
    name="CiteSeer Metadata Filter"
    point="net.nutch.parse.HtmlParseFilter">
    <implementation id="CSParseFilter"
      class="org.citereer.nutch.CSParseFilter"/>
  </extension>

  <extension id="org.creativecommons.nutch.CSIndexingFilter"
    name="CiteSeer Indexing Filter"
    point="net.nutch.indexer.IndexingFilter">
    <implementation id="CSIndexingFilter"
      class="org.citereer.nutch.CSIndexingFilter"/>
  </extension>

  <extension id="org.creativecommons.nutch.CSQueryFilter"
    name="CiteSeer Query Filter"
    point="net.nutch.searcher.QueryFilter">
    <implementation id="CSQueryFilter"
      class="org.citereer.nutch.CSQueryFilter"
      fields="cs"/>
  </extension>
</plugin>
```

## Appendix M: JSP Source for Search Application

```

<%@ page
  contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"

  import="javax.servlet.*"
  import="javax.servlet.http.*"
  import="java.io.*"
  import="java.util.*"
  import="java.net.*"

  import="net.nutch.html.Entities"
  import="net.nutch.searcher.*"
%><%
NutchBean bean = NutchBean.get(application);
// set the character encoding to use when interpreting request values
request.setCharacterEncoding("UTF-8");

bean.LOG.info("query request from " + request.getRemoteAddr());

// get query from request
String queryString = request.getParameter("query");
if (queryString == null)
  queryString = "";
String htmlQueryString = Entities.encode(queryString);

int start = 0; // first hit to display
String startString = request.getParameter("start");
if (startString != null)
  start = Integer.parseInt(startString);

int hitsPerPage = 20; // number of hits to display
//String hitsString = request.getParameter("hitsPerPage");
//if (hitsString != null)
// hitsPerPage = Integer.parseInt(hitsString);

int hitsPerSite = 2000; // max hits per site
//String hitsPerSiteString = request.getParameter("hitsPerSite");
//if (hitsPerSiteString != null)
// hitsPerSite = Integer.parseInt(hitsPerSiteString);

Query query = Query.parse(queryString);
bean.LOG.info("submitted query: " + queryString);

String language =
  ResourceBundle.getBundle("org.nutch.jsp.search", request.getLocale())
    .getLocale().getLanguage();
String requestURI = HttpUtils.getRequestURL(request).toString();
String base = requestURI.substring(0, requestURI.lastIndexOf('/'));
%><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%
// To prevent the character encoding declared with 'contentType' page
// directive from being overridden by JSTL (apache i18n), we freeze it
// by flushing the output buffer.
// see http://java.sun.com/developer/technicalArticles/Intl/MultilingualJSP/
out.flush();
%>
<%@ taglib uri="http://jakarta.apache.org/taglibs/i18n" prefix="i18n" %>
<i18n:bundle baseName="org.nutch.jsp.search"/>
<html lang="<%= language %>">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<head>
<title>Nutch: <i18n:message key="title"/></title>
<link rel="icon" href="/img/favicon.ico" type="image/x-icon"/>
<link rel="shortcut icon" href="/img/favicon.ico" type="image/x-icon"/>
<jsp:include page="/include/style.html"/>
<base href="<%= base + "/" + language %>/">
</head>

<body>

```

```

<jsp:include page="<%= language + "/include/header.html"%>"/>

<form name="search" action="/search.jsp" method="get">
<input name="query" size=44 value="<%=htmlQueryString%>">
<input type="hidden" name="hitsPerPage" value="<%=hitsPerPage%>">
<input type="hidden" name="hitsPerSite" value="<%=hitsPerSite%>">
<input type="submit" value="il8n:message key="search"/>">
</form>
<%
    // perform query
    bean.LOG.info("issued query: " + queryString);
    Hits hits = bean.search(query, start + hitsPerPage, hitsPerSite);
    int end = (int)Math.min(hits.getLength(), start + hitsPerPage);
    int length = end-start;
    Hit[] show = hits.getHits(start, length);
    HitDetails[] details = bean.getDetails(show);
    String[] summaries = bean.getSummary(details, query);

    bean.LOG.info("total hits: " + hits.getTotal());
%>

<i18n:message key="hits">
    <i18n:messageArg value="<%=new Long(start+1)%>" />
    <i18n:messageArg value="<%=new Long(end)%>" />
    <i18n:messageArg value="<%=new Long(hits.getTotal())%>" />
</i18n:message>

<%
    for (int i = 0; i < length; i++) {                // display the hits
        Hit hit = show[i];
        HitDetails detail = details[i];
        String title = detail.getValue("title");
        String url = detail.getValue("cs_dcidentifier");
        String summary = summaries[i];
        String id = "idx=" + hit.getIndexNo() + "&id=" + hit.getIndexDocNo();

        if (title == null || title.equals(""))        // use url for docs w/o title
            title = url;

        %>
        <br><br><b>
        <a href="<%=url%>"><%=Entities.encode(title)%></a>
        </b>
        <% if (!"".equals(summary)) { %>
        <br><%=summary%>
        <% } %>
        <br>
        <span class="url"><%=Entities.encode(url)%></span>
        (<a href="/explain.jsp?<%=id%>&query=<%=URLEncoder.encode(queryString)%>"><i18n:message
key="explain"/></a>)
        <% } %>

    %>

<%
    if ((hits.totalIsExact() && end < hits.getTotal()) // more hits to show
        || (!hits.totalIsExact() && (hits.getLength() >= start+hitsPerPage))) {
        %>
        <form name="search" action="/search.jsp" method="get">
        <input type="hidden" name="query" value="<%=htmlQueryString%>">
        <input type="hidden" name="start" value="<%=end%>">
        <input type="hidden" name="hitsPerPage" value="<%=hitsPerPage%>">
        <input type="hidden" name="hitsPerSite" value="<%=hitsPerSite%>">
        <input type="submit" value="il8n:message key="next"/>">
        </form>

    %>
    }

    if ((!hits.totalIsExact() && (hits.getLength() < start+hitsPerPage))) {
    %>
        <form name="search" action="/search.jsp" method="get">

```

```
<input type="hidden" name="query" value="<%=htmlQueryString%>">
<input type="hidden" name="hitsPerPage" value="<%=hitsPerPage%>">
<input type="hidden" name="hitsPerSite" value="0">
<input type="submit" value="<i18n:message key="showAllHits"/>">
</form>
<%
  }
%>

<p>
<a href="http://www.nutch.org/">

</a>

</body>
</html>
```