Hyperlinking digital libraries on the web

Juan Camilo Zapata
Department of Computing
Macquarie University
Sydney, Australia
juan.zapata-tinoco@students.mq.edu.au

Abstract

In the academic field almost all scholarly documents have a references section that supports the ideas and positions they take. This section creates a connection between documents to help the reader find the article in case they want to know more about the topic. Creating a method of navigating through these connections and finding articles from which the scholar document is based on is a task that has been done by several applications in many ways. This paper explains a research made on how to automate the manual process involved to find scholar documents using a search engine and bibliographic information. The tool had a accuracy of 42% locating 42 references of the 100 cited works.

1 Introduction

The amount of scholar articles has increased on the internet over the past years creating a gap in the interconnectivity between all of this knowledge. In the current time, there have been several solutions to find articles and interconnect them to be able to create a web of knowledge, giving the possibility to navigate on this network. This project it a small part of a bigger project in which the big objective is to gather several PDF articles, extract the references and for each reference create a link to the corresponding article. The project final outcome is a piece of software that by receiving a cited reference it will locate in the web publication of the searched article. This will fill the gap of finding references in the web and the bigger project will add this links to the referee document. This article will describe a solution implemented from the practice of doing the process of searching manually references using only the available information from normal scholar documents and creating a tool that finds URLs to the referenced article on internet. This process will be done by using a search engine and filtering the information that is returned from the search engine. The tool must satisfy that it can be integrated to the previous work done to add hyperlinks to a PDF file by publishing receiving the bibliographic reference and returning a set of possible URLs that may be the referenced article. This is going to be accomplished by developing the tool in python and setting the service in a web server.

In section 2 the paper will discuss some solutions implemented to interconnect articles through the reference section. In section 3 the paper will explain the process done manually and the analysis of the results gathered of the manual process. In section 4 it will explain the solution implemented with a pseudo code to illustrate the process been used and the results of the tool comparing it with the information collected from the manual process. In section 5 the article will with the results and analysis of the testing procedure and an overview of the results. Finally in section 6 the article will summarize the conclusion from the creation of the tool.

2 Overview over existing tools

This section will describe the methods used to interconnect web publications using the reference section and the difference between this solutions and the developed outcome of this project. The two solutions found in the market are Google™ Scholar which works with tags and Cross-Ref which uses identifiers to interconnect the articles.

2.1 Google™ Scholar

Google™ Scholar uses it own system of page ranking to determine which information is relevant using the search parameters, by ranking articles and using their assigned tags Google determines the possible articles to retrieve in the

search result. Nevertheless they use the concept of librarians, publishers and articles to be able to add more tags and information to the articles published [Noruzi 2005]. The Google™ Scholar website (2009) describes the searching method as a search within the tags of the articles to be able to identify the related document, creating a search engine in the tags of the articles. This method has been discussed by Jacsó (2005) by explaining some errors that Google™ have while performing their searches; it shows that Google™ Scholar is considering articles that do not have any relevance to be interconnected, it also have problems to identify the exact documents, showing wrong information in the cited by section of each article. Google™ Scholar behaves like a search engine interconnecting the articles using its search capabilities over the tagged information of those.

The gap for this solution is that it is not an automated tool; it is used as a manual process to find the article that the user is looking for. For the future development a search engine like Google™ can be used to find relevant information in the net but it need a little more of intelligence to filter the information that this kind of search engines retrieved as possible results.

2.2 Cross-Ref

Cross-Ref implemented the Digital Object Identifier (DOI) to be able to reference articles from one to another using this identifier [Cross-Ref Organization: 2009]. DOI is a unique identifier used around the world to identify any type of article that has been published. Cross-Ref is using it to link documents by creating the link directly in the reference section directly to the corresponding article using the unique identifier. This task is done by the publisher and to be able to actually create a link to referenced article that target document has to have a DOI number associated. If the target article is lacking of this identifier, the connection between this two cannot be made [Cross-Ref Organization: 2009]. This concept also provides the functionality of changing the location of the article, it will also be found because its identifier is unique [Cross-Ref Organization: 2009].

The problem with this solution is that if the article does not have a DOI number associated the linkage between the parent document and their references cannot be made. Another important issue is that creating these identifiers is not open to the common public, it is a task done by the publisher to create the DOI for the new article and create the linage in the reference section to other articles, finding out the DOI number of the referenced ones. It also has the disadvantage that the reference has to be created manually and there is not an automatic procedure that looks a database of DOIs and pastes the link to the article.

3 Finding References Manually

After analysing the different tools and how the solve the problem of linking articles together the outcome of this project is a piece of software that finds a web publication of an article on the web and returning some possible URLs. To be able to accomplish this goal of creating a tool a set of steps have to be determined to use a search engine, analyse the information found and filter the search made. To determine these steps 100 references from 10 different scholar documents where used in a search engine. The method used to search the referenced article was by taking advantage of the searching criteria that a search engine provides using the exact search mechanism (double quotation marks) using only the title of the article. Through this task it was found that 56% of the references were found in the first 3 results retrieved. The following subsections explain the process defined and the analysis made to determine the heuristic rules to analyse the content of the retrieved data.

3.1 Steps implemented

To be able to find any type of information on the internet every person has different methods to search and different tools to accomplish their goals. Moreover, every person creates and uses its' own algorithm to find what they are looking for. This section will explain the step implemented to find references and the algorithm used. First, it was defined that the search engine that was going to be used was Google™ because it has an API to access the search engine functionality. Secondly, it was defined to search using exact parameters (by using quotation marks). After these definitions the process was defined as follows:

```
Search using quotation marks in Google™
Repeat (Procedure_1)
    Access Url retrieved
    Verify type of document
    If type of document is PDF
      Open the PDF document
      Review if the document has the title of the referenced
      article
      Review if the document is not an abstract
      Review that the document has the author
      If match exist
        Add the URL as a result
        Save number of position found for the URL in the
        results retrieved
        Save type of document
        Finish loop process
      End
   End
   If type of document is a webpage
      Open the webpage
      Review it the title and the authors match with the
      referenced document
      If match exist
        Add the URL as a result
        Save number of position found for the URL in the
        results retrieved
        Save type of document
        Finish loop process
      Else If the article is the one retrieved from the search
      engine
        Find a possible link to download the article
        Execute procedure_1 with the link found
      End
   End
Until 20 first elements checked or a match is found
```

The algorithm uses a title already extracted from the reference section of a PDF file, each title is searched using a search engine and a set of results is retrieve. Then for each result it has to be analyzed each page looking at the title, the extent of the document and the year of publication, if the visual analysis fit with the searched reference its position and other information is saved to be analyzed. It was saved the type of document, position of appearance in the result and number of levels needed to find the result (if necessary). These results will help for the analysis and the creation of the parameters when developing the tool.

3.2 Analyzing the Results

After searching for a hundred references using this procedure it was determined that 54% of the references were successfully found and the other 46% of the references were in a group of not found. The not found group is categorized as links that were abstracts of the referenced article because an abstract is not the complete article, that the researcher needed authentication to review the article or that the article was been referenced by other article, this is that appears in the reference section of the result. The types of articles found in the search were in PDF format and normal html web pages in which from the 54% found 60% were html files and having the other 40% as PDF files. It was also determined that the articles found were in the five first results retrieved from the search engine this is in 44% of the references were found in the first hit of the result, 8% in the second hit and 2% in the following results. These findings helped define the rules to be incorporated into a tool that finds possible URLs from a search result.

4 Hyperlinkage to digital libraries creating the tool

After investigating the previous solutions and the manual process implemented in section 3.1 an application can simulate the manual process with some intelligence when reviewing the possible solution retrieved by a search engine. These URLs comes in extension as html or PDF in which the web pages have to be processed to identify the content and extract hyperlinks to another pages. Making the process iterative simulating a tree view navigation. For each web page it has to be validated if it is the desire page or extract the child nodes and validate again for each one. In the case that the landing page is a PDF, the system just needs to verify the content of the article.

4.1 Architecture

The development will be done in python because of its portability and easy reuse of different modules that has been tested in different applications. It will be implemented in Client-Server architecture to be able to serve different users at the same time [Python Software Foundation: 2009].

4.2 Modules

The tool will needed the following modules to be able to process the bibliographic reference, search for the article using a search Engine, in this case was used Yahoo API because of its easy extent of connecting to the search service and also because Google™ stop emitting licenses to be able to connect to their search engine, and process the different results retrieved:

- pySearch 3.1: Object which creates the connection to Yahoo search engine to be able retrieve the possible results.

- pyXml 0.8.4: Object to read the xml that is returned from yahoo search engine.
- htmlParser: Object which extracts the tags <a> from a html page.
- urllib2: Object which send request to server and retrieves a response in html format.
- pyPdf: Module to processes the information of the PDF files found.

All this modules are installed and used to be able to process a reference using the title of the article and the author and be able to connect to a search engine and process the search performed. The architecture used is a client server application in which the server will have installed this libraries and the search engine module and will create an html page to be able to visualize how the solution is behaving.

4.3 Process implementation

To be able to successfully retrieve valid information from the search engine a process must be define to locate and analyze the data. This process has to be iterative and repeatable with all possible combinations of references. First the application need to ask for the title and author of the reference, then it should execute a query in a search engine using exact matching and word by word matching. From the results retrieve it has to process each page to define if the URL is the article been search. In case the article matches the selection criteria it has to be saved in and returned at the end of the iteration. For each new URL that is going to be returned the system should validate that it has not selected that URL previously. In the following section it will be defined the selection criteria for the possible articles.

The algorithm developed from the defined procedure is explained table 1.

```
def searchReferences(self):
    #looking for references with a regular search
      results = self.searchNormal()
      parser = xmlpYahoo()
      parser.handleResultSet(results)
      for url in parser.urls:
          self.processUrl(url, self.innerPages )
    #looking for references with exact search
      results = self.searchExact()
      parser.handleResultSet(results)
      for url in parser.urls:
          self.processUrl(url, self.innerPages )
```

Table-1 Python code using Yahoo API

Table-1 shows the code implemented to retrieve the URLs from Yahoo API by performing to types of searches an exact search and a normal search. The exact search send parameters to Yahoo API to search the title of the article as it is, using double quotation marks, and the normal search function send the parameters so that the search engine searches using all the words that are in the title. For each retrieved URL it is processed using the function "processUrl()". This function defines if the URL passed is an html or a PDF file and process it to define if the URL is the searched article, using the methodology describe in section 4.4 Defining Possible URLs. The python code is shown in table-2.

```
def processUrl(self, url, recursion):
    if self.validUrl(url) == False:
        return
    html = urllib.urlopen(url).read()
    #determine if the link is a document or a pdf
    urlsplit = string.split(url,'.')
    #process html page to check if it is the final document
    if urlsplit[len(urlsplit)-1] == 'html':
        self.processHtml(html, url)
    #process pdf file to check if it is the final document
    if urlsplit[len(urlsplit)-1] == 'pdf':
        self.processPDF(url)
    if recursion > 0 :
        recursion = recursion - 1;
        htmlparser = htmlParser()
        htmlparser.parse(html)
        #get a tags and pdf files
        hyper = htmlparser.get_hyperlinks()
        for aurl in hyper:
            aurlcomplete = self.completeUrl(url,aurl)
            self.processUrl(aurlcomplete,recursion)
```

Table-2 Process URL Function

Table-2 is the code used to process each of the URLs retrieved from the search engine, it decides to process the PDF or to process the html page and its links. From the manual experience the system should retrieve the first 20 results from the search engine and visiting only 2 levels down in the child hyperlinks of the retrieve URL. This was implemented by adding a recursion parameter to define how many levels the application should look for links to other pages.

4.4 Defining Possible URLs

After an analysis on the retrieved information and the possible user input, there are several complication on processing and selecting a URL as the possible landing page:

1. The web page could be a summary or an abstract of the searched document.
2. The web page or PDF can have the title but using the article as a reference in the reference page.
3. There could be several pages with the same title of the document.

To be able to solve this problems there are some considerations that have to be implemented in the application:

1. Checking that the article has more than 500 words in its content will help to solve the problem that the retrieve document is an abstract or a summary of the searched article.
2. Checking for the title in the first 500 words of the document will filter the result by taking out the articles that have referenced the searched document.
3. Looking for the author in first 500 words. It was analyzed that some documents have the author or authors in the first part of the article.

These three considerations will help to overcome the previous complications that retrieving several pages will bring. In the code this rules have been defined by doing string search functions that Python had in his framework.

Table-3 shows the criteria implemented to check if the webpage or PDF has the title in the first half of the document, overcoming the problem of selecting articles that reference to the searched document. It also searches for occurrences of the author and add this URL to the returned array of URLs.

```
def processHtml(self, s, url):
    #count the occurrences of the title
    titleOcc = string.find(s, self.title);
   #count the occurrences of the title
    authorsOcc = string.find(s, self.authors,0,s.size/2);
    if titleOcc > 0 and authorsOcc > 0:
        self.addUrl(url)


def processPDF(self, pdfurl):
    print "processing pdf"
    input = PdfFileReader(file(pdfurl, "rb"))
    #count the occurrences of the title
    titleOcc = string.find(input, self.title);
   #count the occurrences of the title
    authorsOcc = string.find(input,
self.authors,0,s.size/2);
    if titleOcc > 0 and authorsOcc > 0:
        self.addUrl(pdfurl)
```

Table-3 Process HTML and PDF functions

4.5 Problems and outcomes

The final outcome of the application is a set of possible URLs to which an article can be retrieved using the selection criteria explained previously.

Developing this application had incurred in the following problems:

1. Navigating through the child nodes of a page, understanding the child nodes as hyperlinks to another pages.
2. Fast processing of big html pages and PDF files.
3. Finding the title of the article in big chunks of data.

All of this processing of text and hyperlinks will be reflected in the time of response from the server with an answer. The average time of response was 45 seconds.

5 Evaluating the tool

Comparing the application with the manual process in terms of the URLS retrieved, the tool is retrieved successfully 80% of the references found in section 3.1. This is from 54 references found in section 3.1 44 references were successfully retrieve in the tool but as it is a tool with heuristic rules there the other possible URLs retrieved were irrelevant creating future project to add more filters for the possible URLs.

6 Conclusions

This document has explained the manual process used to find references on the web using a search engine, the results of the process and the development of a tool that automates this process. The decision rules are limited because of the small amount of information in the bibliographic reference to be able to find the exact article on the web. For further investigations the parent article should also provide some relevant words to be able to perform a clearer search with the search engine this will help the search engine to find easily the article and the tool will be in charge of analyzing the content of the URLs.

References

Bakkalbasi, N., Kathleen Bauer, Janis Glover & Lei Wang. June 2006. Three options for citation tracking: Google Scholar, Scopus and Web of Science. Biomedical Digital Libraries. URL: http://www.bio-diglib.com/content/3/1/7. Retrieved March 2009.

Cross-Ref Organization. February 2009. Cited-by linking. DOIS for research content. URL: http://www.crossref.org/citedby.html. Retrieved March 2009.

Google™ Scholar Website. 2009. About Google™ Scholar. URL: http://scholar.google.com.au/intl/en/scholar/about.html. Retrieved on May 2009.

Jacsó, P. 2005. Google Scholar: the pros and the cons, Online Information Review, vol. 29, no. 2, pp. 208-214.

Noruzi, A. 2005. Google Scholar: The New Generation of Citation Indexes. Libri. Department of Library and Information Science, University of Tehran, Tehran, Iran. Vol. 55 pp.170-180.

Python Software Foundation. (2009). Python Programming Language -- Official Website. URL: http://www.python.org/. Retrieved on May 2009.