Benchmarking Current Deep Web Crawlers

Amruta Chavan (agc9066@rit.edu)

Advisor: Dr. Carlos Rivero

Dept. of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York
Fall 2016
## Table of Contents

1. Introduction ......................................................................................................................... 4  
2. Related work .......................................................................................................................... 5  
3. Traditional Crawlers ............................................................................................................. 6  
4. Hidden Web Crawlers ........................................................................................................... 7  
   4.1 Advantages of Hidden Web Crawlers ............................................................................. 8  
   4.2 Types of Hidden Crawlers ............................................................................................ 8  
   4.2.1 Classification based on crawling method: ................................................................. 8  
   4.2.2 Classification based on keyword selection method: .................................................. 8  
5. Project Scope and Additional Features ................................................................................ 9  
6. Technology ............................................................................................................................ 9  
7. Design and Implementation ................................................................................................... 10  
   7.1 Front-end Design: ............................................................................................................ 10  
   7.2 Crawler Design and Implementation .......................................................................... 11  
   7.2.1 Select Seed URL: ...................................................................................................... 11  
   7.2.2 Handling Robots.txt: ................................................................................................. 11  
   7.2.3 Heuristics for form Filling: ......................................................................................... 12  
   7.2.4 Classifying Results: .................................................................................................. 13  
   7.2.4.1 Heuristics to classify page as valid result page ...................................................... 14  
   7.2.4.2 Heuristics to classify page as error page ............................................................... 15  
   7.2.4 Result Page Navigation: ............................................................................................ 15  
8. Politeness Policy .................................................................................................................. 16  
9. Random Keyword Selection Crawler Algorithm ................................................................... 16  
10. Experiments and Results .................................................................................................... 18  
11. Conclusion .......................................................................................................................... 19  
12. Future Work ....................................................................................................................... 19  
13. References ......................................................................................................................... 20
Table of Figures

Figure 1: Traditional Web Crawler .................................................................................. 6
Figure 2: Hidden Web Crawler ........................................................................................ 7
Figure 3: Front end design................................................................................................. 10
Figure 4: Success Page Classification .............................................................................. 13
Figure 5: Error Page Classification ................................................................................... 15
1. Introduction

The content of a website is listed in a web engine search result only when the web content can be indexed by the web search engine. A web crawler is an automatic program that is used to traverse the world wide web with the objective to index the web content. The web crawlers are sometimes also used for automated web site maintenance task to check the validation of html code.

Traditionally, crawlers have only targeted a portion of the Web called the publicly indexable Web (PIW). PIW is the set of pages reachable purely by following hypertext links, ignoring search forms and pages that require authentication. Traditional web crawlers generally traverse through a set of links on the web to discover and download data.

However, most of the web sites now-a-days require some input from the user to give results. Data from such websites cannot be obtained from such sources by only following the hyper-text links. Such content is available to user only after the user has filled out a form or submitted a request. This content is known as hidden web content. The content which is created dynamically depending on the user input is also hidden data. The proportion of hidden web content is significantly high. This data can prove to be high value for some searches.

This web content however cannot be indexed by traditional web crawlers and hence does not show in the web engine search results. To solve this problem, the concept of hidden web crawlers is introduced. The hidden web crawlers employ a mechanism for automatic form filling, navigation and result classification. Since these web crawlers can access the hidden web content, this content can be indexed by the web engine and hence shows up in the search result.
2. Related work

The first paper “Extracting Data Behind Web Forms” [1] discusses mostly about the form filling process. The authors have made a few assumptions while implementing the model. Assumptions made are no user authentication would be required by any website that is being traversed, whenever a form a filled the result returned would be relevant, the http get/post request made to the web site would return results. The authors used the following approach to for form filling- 1) If the input was a text field then blank input was passed in request. 2) If the input had a default value then the default value was passed in the result. 3) If the input was a selection field then all possible permutations of the input query were tried. This resulted in lots of http post/get request being made. The authors discuss an approach to classify the returned pages as relevant or irrelevant. The authors also present a mathematical model to retrieve a query which will return maximum amount of data.

The second paper ‘Crawling the Hidden Web’ [2] discusses a framework called Hidden web exposer (HiWE) which is task specific deep web crawler. This paper mainly focuses on the task of form filling and also provides a mathematical model for page classification. The most important take-a-way from this paper is the method used to classify the search results. If the response contained text like ‘no result’ or ‘zero result’ then it is classified as error page. The authors have also suggested methods to retrieve valid result links excluding the header and footer links. The LVS (Label value set) table is updated depending on the content of the result page.

The third paper “Downloading Textual Hidden Web Content through Keyword Queries” [3] focuses on the different methods for keyword selection. The authors suggest three different approaches to issue query to database. The first approach is random, in this the keyword is picked randomly from a dictionary. The second approach is frequency based in which the keywords are determined based on their frequency of occurrence in search result. The third approach is adaptive in which a good query is picked based on content of downloaded pages.
3. Traditional Crawlers

For a search engine to return different files or documents location, it must have first found it. Crawlers or spiders are used for this. Crawlers or spiders are basically robots that traverse the world wide web to scan the information and store it to a database. The process of traversing to web sites and creating the list is called crawling. The spider indexes the web content and keywords so that it can be returned in search results efficiently.

Traditional web crawlers work on only publically indexable web. That means it can access and index only that content which can be obtained by directly visiting the links. It does not perform the operation of issuing any query or filling any form. Hence traditional web crawlers cannot visit and index the hidden content.

![Figure 1: Traditional Web Crawler](image_url)
4. Hidden Web Crawlers

Hidden web crawlers employ a mechanism to access and index the hidden web data. The general architecture and working of a hidden web crawler is as follows:

The most important step of a hidden web crawler is automated form filling or query issuing. For this the crawler must be provided a set of label value pair (LVS table) or keywords to fill the form. The crawlers parse the request form to find different input fields, these fields are then filled with the keywords value. The crawler then submits this form in an automated manner.

The next step is to update the LVS with valid keywords and classify the dynamic web content as valid or invalid. A result page is considered invalid if no results are returned after the form submission. If the result page is valid then important result links are extracted from the result web page. Results distributed over multiple pages are crawled and hidden data is indexed and stored in database.

![Figure 2: Hidden Web Crawler](image)
4.1 Advantages of Hidden Web Crawlers

1. Because the hidden data is not indexed, user has to visit these websites holding hidden data and issue multiple queries to obtain it. This data is not indexed and returned in the search result for most search engines. Since this hidden data can be indexed using hidden crawlers, users will save significant time in search operation as they will not have to visit each website and issue queries.

2. As hidden data is not shown in search results, users do not perceive what actually exists on the web. This can result in bias as only indexed data is returned.

3. Hidden crawlers also provide a mechanism to index dynamic content which is dependent on user input.

4.2 Types of Hidden Crawlers

4.2.1 Classification based on crawling method:

a. Breadth-Oriented Crawling: These crawlers focus on covering wide range of data/URL sources rather than exhaustively crawling content inside single resources.

b. Depth-Oriented Crawling: The focus of these crawlers is to extract maximum data from a single source.

4.2.2 Classification based on keyword selection method:

a. Random: In this crawling method the keyword used for form filling is obtained from a random dictionary. The dictionary can be domain specific or random.

b. Generic-frequency: In this method generic frequency distribution of each keyword is obtained and the most frequent keyword is used in form filling. This helps to return more matching content and also save form filling time.

c. Adaptive: In this crawler the documents returned from the queries are analyzed and keywords returning most content are shortlisted. Based on these promising keywords the crawler can issue queries to obtain maximum data.
5. Project Scope and Additional Features

1. For this project, we will mostly be concentrating on form filling for websites with only one text field to reduce the form filling complexity.
2. It is assumed that the search result links will not lead to another search page.
3. For this project we will be concentrating on health domain websites and a predefined set of websites will be used for testing.
4. The keywords for form filling will be obtained from websites like WordStream.com.
5. Additional step of checking the ‘robots.txt’ of each website before crawling will be added to the crawler design.
6. Additional features of ‘politeness’ will be added to the original crawler design to avoid overloading the servers.
7. The form filling method suggested in the research papers [1][2][3] is to make direct get/post request to the website. However, this is difficult to replicate as most websites require an api key when making a get/post request. In order to solve this problem a decision was made to use ‘Selenium WebDriver’ for this step. Selenium WebDriver is a tool which is mainly used to automate web application testing.
8. A simple UI will be developed to provide inputs like seed URL, keywords and pattern file to the crawler.

6. Technology

1. Selenium: Selenium is an automation testing tool. This tool is mainly used for testing web applications. But the scope of this tool is not limited to only web testing. In this project Selenium is used in the automatic form filling phase. Selenium WebDriver java api is used for this.
   **Installation**: The Selenium java project can be setup using the maven. The pom.xml file content is provided on the Selenium website. After creating the pom.xml file the maven clean install command is issued from the console. This will download Selenium and all its dependencies and add them to the java project.

2. Java Swings: Java Swings is used to develop a simple UI that helps user to provide the URL file, pattern file, keywords file. It will also display the crawled links as shortlisted keywords.
7. Design and Implementation

7.1 Front-end Design:
Front-end is developed using Java Swings. The UI is used to accept input file from the user and display the classified valid keywords and links. The user has to select the seed URL file; this file contains the links that the crawler should crawling to retrieve hidden content. The next input is the keyword file which will be used as input during form filling. The user is given an option to upload the pattern file. The pattern file specifies the pattern to extract valid result links.

![Figure 3: Front end design](image-url)
7.2 Crawler Design and Implementation

7.2.1 Select Seed URL:
The first step in implementation was to shortlist a bunch of websites which will be used for crawling. Alexa( http://www.alexa.com/) was used to find these websites. We were focusing on websites with only one search text field. So, initially 5 websites from the health domain were selected. The website URL was stored in a list. Initial one week was spent on installing and learning Selenium WebDriver. A simple java program to perform operations like access search text field on a website, insert keyword in the text field, click on submit button was written. This code was specific to one website. The search field and submit button xpath values were hardcoded in this code.

7.2.2 Handling Robots.txt:
Robots.txt is a file used by websites owners to give instructions to web crawlers or robots about their website. The website owners can specify instructions for the robots in this file. For example, if the web site owner does not want the robot to crawl a particular folder or a particular file then such instructions are mentioned in the robots.txt. If these instructions are not followed by the web crawler, then it might be considered as a malware robot.

Sample robots.txt:
User-agent: *
Disallow: /
This ‘User-agent: *’ means that this robot.txt is for all the robots ‘Disallow: /’ means that the robot should not crawl the web site at all.
‘Disallow:’ means that the robots have complete access to all the folders of the web site.

Robots.txt Location:
Almost all websites have robots.txt. The file is always located in the same place on any web site. It can be accessed at domain URL/robots.txt location.
Algorithm:
In order to follow the robots.txt rules, before crawling any website the robot.txt content was read. A function was written to navigate to the robot.txt.
if the robot.txt has ‘Disallow: /’ statement then
    return false (web site should not be crawled)
else if this statement was not present then
    return true (web site can be crawled)

7.2.3 Heuristics for form Filling:
1. For the project a generic code was required which could work for all shortlisted websites without any modification. The shortlisted websites webpages were studied to find common factors.
2. Heuristics to find the search input text field on websites home page:
   Selenium WebDriver method ‘findElements’ was used. The search input text box was selected and added to an arraylist if one of the following conditions was true.
   a. input tag id property contains text ‘search’.
   b. input tag name property contains text ‘Search’
   c. input tag value property contains text ‘Search’
3. After finding the search text the next step was to insert the search keyword in the text field. Some websites had hidden input text fields and these input elements were also the arraylist mentioned in step 4. The heuristics to select the correct input field from these list elements was as follows:
   a. The elements tag name was input.
   b. The elements type was text.
   If the above conditions were satisfied, then the search keyword was inserted in the input text field using Selenium’s sendKey function.
4. The next step was to automate the submit button click operation. Heuristics to find the submit button are as follows:
   a. input tag id attribute contains text ‘submit.
   b. input tag type attribute contains text ‘submit’
   c. input tag src attribute contains text ‘Search’
   d. button tag type attribute ‘submit’
All the DOM elements satisfying these properties were added to a button list. The next step now was to write heuristics to find the exact submit button from this list. The heuristics to find the submit button from the button list is as follows:

If the element in list has tag name ‘input’ then
  If the element type is submit or element type is image then
    If element id attribute contains text search then
      Click the element(button)
    Else if element value attribute contains text ‘search’ then
      Click the element(button)
    Else if element onclick attribute contains text ‘search’ then
      Click the element(button)
    Else if element class attribute contains text ‘search’ then
      Click the element(button)
  Else if element in the list has tag name ‘button’ and element type attribute equals ‘submit’
    Click the element(button)

7.2.4 Classifying Results:
The result page after form filling operation is performed has to be parsed to get valid result links. These result links after traversal will give data related to the keyword. Each valid link is stored only once.

Figure 4: Success Page Classification
7.2.4.1 Heuristics to classify page as valid result page

The success page contains the valid result links as well as non result links that are present in the header and footer section. These invalid links can be links to social networking websites or website links outside the current domain.

To extract valid search results from a web page, two approaches are used.

a. **URL pattern provided by user:** The user is given an option to input the result URL pattern. If the user has provided the pattern, then only the links that match the pattern are extracted and stored. For this the Java Pattern and Matcher classes are used.
   
   if the link is well formed then
   
   if link matches pattern
   
   Store the link in ShortlistedURL file.
   
   Store the keyword and link count in ShortlistedKeywords file
   
   else
   
   discarded the link

b. **URL pattern not provided by user:** If the user has not provided the pattern for result matching then generic heuristics are applied to extract search result links.

   if the link is well formed then,
   
   if the link is outside current domain then
   
   discarded the link
   
   else
   
   Store the link in ShortlistedURL file.
   
   Store the keyword and link count in ShortlistedKeywords file
7.2.4.2 Heuristics to classify page as error page

a. Get the result page source using Selenium’s ‘getPageSource’ function.

b. If this page contains words like ‘sorry’, ‘no pages’, ‘sorry no result’, ‘sorry no match’ then the page was classified as error page.

c. The search keyword for which this page was obtained was discarded from the dictionary list.

7.2.4 Result Page Navigation:

For some keywords the number of results are more and hence are split on multiple pages. The links to the next page usually have text like ‘next’, ‘show more’, ‘1’, ‘2’ etc. It is important to traverse all these pages so that the correct coverage of a keyword can be obtained. The heuristics for page navigation are:

if the result page contains next link then
    click the next link and load new page
    Store the links in ShortlistedURL file.
If next link exists then
    Repeat the page navigation process
8. Politeness Policy

It is possible for a web crawler to perform its operations must faster when compared to human. This might result in overloading of the web server. Continuous fast requests by crawlers can sometimes be considered a DOS attack by some servers. In order to solve this problem, sleep commands have been added after each operation. So the crawler gives a perception as if a human was performing the operations. This also helps in giving some time for the new pages to load.

9. Random Keyword Selection Crawler Algorithm

In this method the search keyword is selected randomly from a dictionary. For this project however domain specific dictionary was used. So a dictionary with health domain keywords was used as input. Algorithm to feed data to the crawler:

1. Read seed URL from seedURL file.
2. Check if the robots.txt of web site that is to be crawled
3. Open the web site in browser using the WebDriver.
4. Find the search input text field in the web
5. Read a random keyword from the input dictionary file.
6. Write the keyword in the input field.
7. Find the search button in the page using using heuristics mentioned.
8. Click the submit button.
9. Use the result page classification to decide if the keyword should be retained or discarded.
Figure 6: Design of Random keyword hidden web crawler
10. Experiments and Results

The experimental setup for the random keyword web crawler is as follows:

**Domain:** Medical

**Test Websites:** Lab Tests Online (https://labtestsonline.org/)
- familyDoctor (https://familydoctor.org/)
- healthdirect (https://www.healthdirect.gov.au/)

**Number of keywords:** 1000

**Keywords shortlisted:** 636

A simple java program was written for testing purpose. This program gives the list and count of all links present on a particular website. This count is used to calculate the coverage of keywords.

**Coverage** = Links extracted by keyword/ total number of links

**Submission efficiency** = No. of successful submission/ Total number of submissions

\[
= \frac{636}{1000} = 0.636
\]

**Time required for processing:** ~8hr

The keywords that gave a valid result were classified into three categories: High coverage, Medium coverage, and low coverage.

<table>
<thead>
<tr>
<th>High coverage</th>
<th>Medium coverage</th>
<th>low coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screening(432/1000)</td>
<td>Pain(62/1000)</td>
<td>Surgery(16/1000)</td>
</tr>
<tr>
<td>Diabetes(390/1000)</td>
<td>Health(58/1000)</td>
<td>Transplantation(13/1000)</td>
</tr>
<tr>
<td>Cardiovascular(191/1000)</td>
<td>Endocrinology(49/1000)</td>
<td>Hypertension(13/1000)</td>
</tr>
<tr>
<td>Drugs(190/1000)</td>
<td>Flu(40/1000)</td>
<td>Neonatology(2/1000)</td>
</tr>
<tr>
<td>Transplantation(166/1000)</td>
<td>Liver(38/1000)</td>
<td>Dentistry(3/1000)</td>
</tr>
<tr>
<td>Medicine(144/1000)</td>
<td>Myocardial(29/1000)</td>
<td>Radiotherapy(2/1000)</td>
</tr>
</tbody>
</table>
11. Conclusion

Traditional web crawlers usually follow links on the Web to discover and download pages. They can only access the publicly indexable web and hence cannot get to the Hidden Web pages which are only accessible through query interfaces. Since the hidden data cannot be indexed it does not show in search results for search engines. In this project, we studied the importance of deep web and also implemented a Hidden Web crawler that can automatically query a hidden web site and download pages from it. We added new features like checking robots.txt for permission to crawl website and politeness policy to avoid overloading servers to the original crawler design. An interface was developed to accept inputs and display the results. All the data obtained while crawling was written to files which was used for keyword coverage analysis.

We were able to successfully test the crawler implementation with three medical domain websites. We have successfully classified keywords as high coverage, medium coverage and low coverage depending on the search results they returned. Using the keywords with high coverage maximum hidden data can be obtained in minimum time.

12. Future Work

1. In the current implementation the pattern to extract valid links is provided as an input by the user. CALA[5] is a URL-based web page classification system. Using CALA valid URL pattern can be obtained which can be used to classify link as valid or invalid. Merging the current implementation of random keyword crawler with CALA is a part of future work. This will enhance the performance of web crawler as more accurate patterns can be generated for link extraction.

2. Currently the project scope is limited to only on input field for form filling. The next step is to include more fields in the automated form filling phase.
13. References


