CHAPTER 4

WEB DATA EXTRACTION BASED ON HEURISTICS

4.1 OVERVIEW

This chapter discusses the design of a Web Data Extractor (WDE) based on Heuristics. It details the technique used for crawling the scientific publishers’ website to gather the URL of journal home pages. In addition to that, it discusses the heuristics used to extract the detailed specification of journal such as title, aims and scope, Source Normalized Impact Factor (SNIP), SCIMAGO Journal Rank (SJR), etc. The applicability of the approach is proved by using it to extract journal specifications from multiple publishers’ websites. The accuracy of the proposed technique is expressed using the two classic metrics, namely, precision and recall. Finally, the advantages and limitations of the proposed WDE based on Heuristics are discussed.

In this chapter section 4.2 gives introduction to Web Data Extraction, section 4.3 details the architecture of Journal Information Extraction System, section 4.4 presents the experimental results, advantages and limitations of the proposed methodology and section 4.5 summarizes the chapter.

4.2 INTRODUCTION

The World Wide Web is a useful source of information which contains data in many different formats. The diverse formats of web pages act as a barrier for performing automated processing. Many business organizations require data from the World Wide Web for carrying out analytical tasks such as business intelligence, product intelligence, competitive intelligence, decision making, opinion mining, sentiment analysis, etc. The information from the WWW can be obtained using search engines. The search engines make use of web robots to crawl through the web and retrieve the links that might contain the information searched for, and presents the information to the user in the form of list of hyperlinks. The search engines are capable of retrieving information from the surface
web but not from the invisible web or hidden web which requires form submission. From [77], it is clear that the amount of pages not indexed to search engines is 400 to 550 times greater than the size of the surface web. The deep web acts as a huge repository of information for many data analytics applications. The websites are structured using different templates and therefore, the extraction process has to deal with heterogeneity in terms of technologies used in website creation. The scripting languages like Java Server Pages, Active Server Pages, Perl, PHP etc. are used for creating dynamic web pages and technology like CSS is used for beautifying web pages. In order to improve the user’s web browsing experience and make the website visually engaging, web pages are structured using the latest technologies such as HTML, CSS and AJAX. This makes data extraction a highly complex task. The challenges faced by extractors include heterogeneous formats, changes in structure of web pages, usage of highly sophisticated technologies for enhancing web browsing experience, etc. The commercial tools such as Lixto [54], Mozenda [78] and KDNuggets [79] offer Graphical User Interface to guide the extraction process but the level of human intervention required is very high.

The general picture of web data extractor is shown in Figure 4.1. It involves four steps, namely, Search, Locate, Filter and Extract. The search step involves finding the target web pages given the URL of the website as input. The locate step refers to determining data rich regions in the target web pages and filter step involves the process of extracting the data rich nodes after eliminating non-informative sections such as navigation links, pop-up menus, advertisements, etc. The last step of web data extraction involves extracting the attribute-value pairs from the target web pages, which represents the description of the object.

![Figure 4.1 Steps in Web Data Extraction](image-url)
4.2.1 Motivation

Many researchers face difficulty in finding the most appropriate journal for their research article publication. The publishers’ home page contains links to various journals organized in a chronological order. The user should crawl through the sequence of hyperlinks to reach the journal home page which contains detailed information about the journal such as Impact Factor (IF), Five Year Impact Factor, SCIMAGO Journal Rank (SJR), Source Normalized Impact Factor (SNIP), etc. Manually carrying out the process is time-consuming and therefore, an automated data extraction technique is proposed. It automatically crawls through webpage in the scientific publishers’ websites which contain the journal links organized in alphabetical order A-Z. After identifying the home pages of the scientific journal websites, heuristic algorithm is used to determine the location of the journal specification. The journal specification is extracted and stored in structured format which aids in carrying out various kinds of analysis on the extracted journal information. As shown in Figure 4.1, in the first step, a heuristic crawler is used to identify the URL of the journal home pages linked to the publishers’ website. In the next step, DOM tree of the journal home page is generated and text nodes are identified. The text nodes that match with the domain keywords are determined and the XPATH [72] of all those nodes are obtained. It can be observed that attribute values are located very close to the attribute name and therefore, this heuristic can be used to retrieve the corresponding value of the attributes. The XPATH determined for the attribute name-value pairs corresponding to journal specification are stored and can be used to extract the attribute name-value pairs in similarly formatted journal home pages. The extracted journal information is stored in a structured form onto a relational database.

4.2.2 Problem Definition

The problem of Journal Specification Extraction can be stated as follows: Given the URL of the publishers’ website, the problem is to determine the set of journal home pages \{p_1, p_2, \ldots, p_n\} which contain the set of keywords like SCIMAGO Journal Rank (SJR), Source Normalized Impact Factor (SNIP), impact factor, etc. representing the search
object (journal) and then, to extract the structured records belonging to each search object (journal).

Constraint: The web pages \( \{p_1, p_2, \ldots, p_n \} \) from different publishers’ websites might have been created using heterogeneous server-side templates.

### 4.3 ARCHITECTURE OF JOURNAL INFORMATION EXTRACTION SYSTEM

The overall architecture of the journal information extraction system is shown in Figure 4.2. The first step is to crawl through the scientific publishers’ website given by the seed URL and to extract the URLs of the journal home pages which contain the journal specification. A heuristic algorithm is used for extracting data records and then, the extracted data records are stored in a structured form in a relational database. The relational database can then be queried to obtain the needed information. Figure 4.3 shows the phases of the proposed system.

URL of the publisher’s website is given as input to the journal information extraction system. Heuristic crawling algorithm is designed based on the fact that the journal links are arranged in a chronological order from A-Z. After clicking the alphabet, a list of journals, the names of which start with the specified alphabet gets displayed. Again clicking on a specific link, results in the journal information page which might contain a link to the journal’s home page. Thus, the user has to navigate through a series of links in order to reach the journal’s home page. These steps can be automated by using the heuristic crawler and the URL of all the journals linked to the publisher’s website can be retrieved and stored.

In phase 2, WDE technique based on heuristics is used for the extraction of journal specification such as title, ISSN, SCIMAGO Journal Rank (SJR), Source Normalized Impact Factor (SNIP), Impact Factor, etc. from the journal’s home page. As a first step, DOM tree corresponding to the journal’s home page has been constructed using JSOUP API [73]. The algorithm is based on the fact that the journal specifications displayed on the rendered web page by the browser correspond to text nodes in the DOM tree. Instead of processing the entire DOM tree, only the leaf nodes are considered. These nodes are
compared with the domain keywords such as Impact Factor, SCIMAGO Journal Rank (SJR), Source Normalized Impact Factor (SNIP), etc. in order to identify the location of the target journal specification in the DOM tree. After determining the location of attribute names in the DOM tree, the values of attributes are extracted based on the observation that the values are present as sibling or child node to the node representing attribute name in the DOM tree.

Location of the attribute name and values in the DOM tree are represented as XPATH expression [72]. The XPATH language [72] is used to locate a node in a structured XML document. Once the XPATH expression is determined for all the journal specification information expressed as attribute name-value pairs, the same XPATH can be used to extract information from the remaining journal home pages since they are generated using the same server-side template. The extracted journal specification information is stored in a structured form in a relational database. Finally, a Graphical User Interface (GUI) is designed to answer user queries in a single interaction with the system.

4.3.1 Mathematical Model

Heuristic Algorithm for Automatic Navigation

Given the URL of the scientific publisher’s site url_p, navigate to the site and extract the set of URLs corresponding to the journal home pages arranged in alphabetical order (i.e.) A-Z. After navigating to journal’s home page, it produces a set of journal specification representing information such as title, ISSN, URL of the journal’s home page, etc.

URLs for all the journals are ordered by A-Z. The set of target URLs is denoted by H = \{h_i, 1 \leq i \leq n\}. Each journal home page h_i is parsed to collect the needed attributes such as title, ISSN, Impact Factor, Source Normalized Impact Factor, SCImago Journal Rank, etc.
The proposed approach is based on the observation that the journal home pages linked to publishers’ website are well formatted. The uniformity in presentation across several web pages of the website is achieved since they are generated using the same server-side template. If the location of target data is identified for a single web page (XPATH for attribute name-value pairs) from the scientific publisher’s website, then the same XPATH can be used for extraction from similarly formatted journal specification pages. It is clear from Table 4.1 representing the journal information obtained from Nature [66] publisher’s website, the XPATH corresponding to attribute value pairs remains the same.
4.3.2 Algorithm for target URL extraction

The algorithm takes base URL of the publisher’s website as input. The algorithm extractURLs shown in Figure 4.4 is based on the observation shown in Figure 4.5 that the hyperlinks to the journals are organized in alphabetical order from A|B|C...|Z. The algorithm filters the anchor elements whose text matches the regular expression ^[a-z]$|^[A-Z]$$. It then crawls to that page in order to find and extract the journals’ home page URL.

Table 4.1 Similarly-structured web pages and the XPATH of the attributes

<table>
<thead>
<tr>
<th>NATURE JOURNALS [59]</th>
<th>XPATH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IMPACT FACTOR</strong></td>
<td>/html/body[@id='home']/div[@id='constrain']/div[@class='content']/div[@id='content']/div[@id='journalnav']/div[@id='content']/div[@class='journal-details']/p[2]/span[@class='impact']</td>
</tr>
<tr>
<td><strong>ISSN</strong></td>
<td>/html/body[@id='home']/div[@id='constrain']/div[@class='content']/div[@id='content']/div[@id='journalnav']/div[@id='content']/div[@class='journal-details']/p[1]</td>
</tr>
<tr>
<td><strong>IMPACT FACTOR</strong></td>
<td>/html/body[@id='home']/div[@id='constrain']/div[@class='content']/div[@id='content']/div[@id='journalnav']/div[@id='content']/div[@class='journal-details']/p[2]/span[@class='impact']</td>
</tr>
<tr>
<td><strong>ISSN</strong></td>
<td>/html/body[@id='home']/div[@id='constrain']/div[@class='content']/div[@id='content']/div[@id='journalnav']/div[@id='content']/div[@class='journal-details']/p[1]</td>
</tr>
</tbody>
</table>
4.3.3 Heuristic Technique for Data Extraction

The aim of this technique is to identify the location of the target attributes, extract the attribute values, annotate and store the extracted data records containing attribute values into a relational database. The technique is based on the heuristics that the target information to be extracted is present as visible content which is displayed in the content area of the browser. In the DOM tree representing the web page, all the text nodes will be present as leaf nodes. The technique involves visiting each URL and obtaining the DOM tree of the web page containing the journal specification. The algorithm `getLeafNodes` is used to obtain the leaf nodes and also, to check whether it is a text node. If they match with the domain keywords such as SJR, SNIP, Impact Factor, etc., then its XPath is determined using the procedure `getFullXPath`. The algorithm `convert_XPath_to_Selector` is used for converting the XPath to a selector string. The same selector string can be used to carry out extraction from similarly structured web pages. The algorithms are explained in detail in the remainder of this chapter.
The algorithm Extract shown in Figure 4.6 takes, seed URL of the publishers’ website as input and calls the procedure extractURLs in order to extract the URLs of the journal home pages. It then calls the procedure displayJournalMetrics for extracting attribute value pairs from each journal’s home page. It establishes the database connectivity in order to store the structured records into a relational database.

The algorithm displayJournalMetrics shown in Figure 4.7 takes as input URL of the journal’s home page. It checks whether the XPATH of ISSN and XPATH of Impact Factor had been determined. If they are not found already, then it finds the XPATH by matching the leaf nodes (obtained using the algorithm getLeafNodes) with the domain keywords such as ISSN, impact factor, etc. and determining the path to those matched nodes.
Algorithm Extract (Seed_URL)

// Input: Seed URL (URL of Publishers' site)
// Output: Structured records
begin
    target_urls = extractURLs(Seed_URL);
    for each url in target_urls do
        impact_factor, issn = display_journal_metrics(url)
        create database connectivity
        store impact_factor, issn
    end for
end

Figure 4.6 Algorithm Extract

Algorithm displayJournalMetrics(target_url)

// Input: target_url
// Output: impact_factor, issn
begin
    useragent.visit(target_url);
    issn_pattern = "\d{1,3}\d\{1,3\}.*\d\{1,3\}"  
    impact_pattern = "\d{1,3}\d\{1,3\}\d\{1,3\}" 
    if (issn_xpath is NULL) then
        begin
            doc = construct DOM tree
            root = node corresponding to Body element
            leaf = getLeafNodes(root)
            for each node in leaf do
                if node.text matches issn_pattern then
                    issn_xpath = getFullXPath(node)
                    issn = node.text.substring(start, end)
                end if
                if node.text matches impact_pattern then
                    impact_xpath = getFullXPath(node)
                    impact_factor = node.text.substring(start, end)
                end if
            else
                impact_factor_selector = convert_xpath_to_selector(impact_xpath)
                issn_selector = convert_xpath_to_selector(issn_xpath)
                impact_node = doc.select(impact_factor_selector)
                impact_factor = impact_node.text.substring(start, end)
                issn_node = doc.select(issn_selector)
                issn = issn_node.text.substring(start, end)
            end if
        end
        return impact_factor, issn
    end
The XPATH is then converted to JQUERY selector string [81], since the Jaunt API [70] supports only the selector string for locating node in the DOM tree. The same XPATH is used for extraction from the remaining journal home pages linked to the scientific publisher’s website.

The algorithm getLeafNodes shown in Figure 4.8 takes as input the root node of the DOM tree and produces leaf nodes that are of type text.

```
Algorithm getLeafNodes (Node root)
//Input: root
//Output: leaf_nodes
begin
    if (root.getType() equals Node.ELEMENT_TYPE) then
        for each node in root.childNodes()
            begin
                if node.getType() equals Node.TEXT_TYPE then
                    leaf_nodes.add(node)
                else
                    getLeafNodes(node)
                end if
            end for
    else
        getLeafNodes(node)
    end if
return leaf_nodes
end
```

**Figure 4.8 Algorithm getLeafNodes**

The algorithm getFullXPath shown in Figure 4.9 constructs the XPATH by taking as input the node for which XPATH has to be determined. It finds the hierarchy of nodes from root to the given node and separates them by /. The algorithm returns the XPATH which is converted to the corresponding selector string using the algorithm convert_xpath_to_selector.

The algorithm convert_xpath_to_selector shown in Figure 4.10 takes the XPATH as input and produces JQUERY selector [74] as output. JQUERY selector [74] is similar to
the XPATH and it is used to select nodes from HTML documents using attribute such as name, id, class, etc. Space in the XPATH is replaced by “.”, “[@class=” by “.”, “[@id=” by “#” and “/” by “>”.

**Algorithm getFullXPath (Node n)**

```plaintext
// Input: Node n
// Output: XPath
begin
hierarchy.push(n)
parent = n.getParent()
while (parent is NOT NULL AND parent.getType() NOT equals DOCUMENT_TYPE)
    hierarchy.push(parent)
    // get parent of parent
    parent = parent.getParent()
end while
// construct xpath
while (!hierarchy.isEmpty() AND (node=hierarchy.pop()) IS NOT NULL)
    if (node.getType() equals ELEMENT_TYPE) then
        XPath.append("/")
        XPath.append(node.getName())
    end if
    if (node.hasAttribute("class")) then
        XPath.append("[@class=" + e.getAt("class") + "]")
    else if (e.hasAttribute("id")) then
        // id attribute found - use that
        XPath.append("[@id=" + e.getAt("id") + "]")
    end if
    if (node.getName().equals("table")) then
        XPath.append("/tbody")
    end if
end while
return XPath
end
```

Figure 4.9 Algorithm getFullXPath
Algorithm convert_xpath_to_selector (String xpath)

input: xpath  
output: selector_string  
begin  
if (xpath.contains(" ")) then  
xpath1 = xpath.replace(" ", ".")  
else  
xpath1 = xpath  
end if  
if (xpath1.contains("["])) then  
xpath2 = xpath1.replace("[", ")")  
else  
xpath2 = xpath1  
end if  
if (xpath2.contains("[@class=""])) then  
xpath3 = xpath2.replace("[@class="", "]")  
else  
xpath3 = xpath2  
end if  
if (xpath3.contains("[@id=""])) then  
xpath4 = xpath3.replace("[@id="", "]")  
else  
xpath4 = xpath3  
end if  
if (xpath4.contains("/")) then  
xpath5 = xpath4.replace("/", ">")  
else  
xpath5 = xpath4  
end if  
selector_string=xpath5  
return selector_string

Figure 4.10 Algorithm convert_xpath_to_selector
4.4 EXPERIMENTAL RESULTS

The experiment was conducted on *Science Direct [64], Wiley [65], Nature [66], BMJ [67], Elsevier [82] and Oxford publishers’ website*, in which journal hyperlinks are organized in lexicographical order and clicking on each link gives a listing of journals starting with the corresponding alphabet. In order to know the journal information, one more level of navigation is required (i.e. “About Journal” hyperlink has to be clicked). The proposed methodology performs navigation to the appropriate level, fetches the structured data embedded in unstructured HTML pages and then, stores it onto relational database. The extraction results are shown in the Table 4.2. The three classic metrics of information retrieval, namely, precision, recall and F1-Measure are used to evaluate the effectiveness of this technique in journal information extraction. The graph in Figure 4.11 shows that for well-structured websites like Elsevier [82] and Wiley [65], accuracy of extraction is good whereas for websites like BMJ [67] and Nature [66] which contains certain differently formatted pages, the use of XPATH expression derived from a sample, does not work well for all the target web pages.

<table>
<thead>
<tr>
<th>Table 4.2 Precision, Recall and F-Measure values for various attributes extracted from journals associated with various publishers’ sites</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elsevier</strong></td>
</tr>
<tr>
<td>SNIP</td>
</tr>
<tr>
<td>SIR</td>
</tr>
<tr>
<td>IMPACT FACTOR</td>
</tr>
<tr>
<td>IMPACT FACTOR FOR 5 YEARS</td>
</tr>
<tr>
<td><strong>BMJ</strong></td>
</tr>
<tr>
<td>IMPACT FACTOR</td>
</tr>
<tr>
<td>ISSN</td>
</tr>
<tr>
<td><strong>Nature</strong></td>
</tr>
<tr>
<td>IMPACT FACTOR</td>
</tr>
<tr>
<td>ISSN</td>
</tr>
<tr>
<td><strong>Wiley</strong></td>
</tr>
<tr>
<td>IMPACT FACTOR</td>
</tr>
<tr>
<td>ISSN</td>
</tr>
<tr>
<td><strong>Oxford</strong></td>
</tr>
<tr>
<td>IMPACT FACTOR</td>
</tr>
<tr>
<td>ISSN</td>
</tr>
<tr>
<td>EISSN</td>
</tr>
<tr>
<td><strong>Science Direct</strong></td>
</tr>
<tr>
<td>IMPACT FACTOR</td>
</tr>
<tr>
<td>ISSN</td>
</tr>
</tbody>
</table>
4.4.1 Advantages

This technique differs from other unsupervised approaches in the following ways:

1. Unlike other DOM tree-based techniques FivaTech [5], DEPTA [29], it does not require processing the entire DOM tree to identify the location of attribute value pairs. Usually, they represent text nodes and text nodes are always leaf nodes in the DOM tree.

2. It requires less processing time since it does not involve complex pattern matching or string alignment algorithms as in [5, 6].

3. The complexity of the extraction process is greatly reduced. This is because once the XPATH is determined for attribute name-value pairs representing journal information for a single journal home page, the same XPATH can be used for extraction from similarly formatted journal home pages linked to the publishers’ website.
4. It can be extended to perform data extraction from deep web pages belonging to different domains, provided domain keywords and their synonyms are known.

4.4.2 Limitations

1. It does not work well in case of differently structured journal pages linked to a single publisher’s website.

2. It does not handle ordering of attributes.

3. It does not accommodate new templates in an ad hoc manner.

4. It does not work for websites in which domain keywords are not mentioned explicitly viz. Discussion Forums.

4.5 CONCLUDING REMARKS

The extraction of structured data from hidden or deep web is a challenging problem because of the intricate structure of the web pages which changes from one website to another. Researchers and publishers tend to use search engines to find appropriate journals for their research article publication. It involves a tedious task of navigating through a series of links to obtain detailed information about the journal such as SCIMAGO Journal Rank, SNIP, Impact Factor, scope, etc. The proposed framework uses a heuristic-based approach for navigating through a series of hyperlinks in the publishers’ website, to find the URL of the journal home pages linked to the website and also to extract the needed journal specification. The versatility of the proposed approach is proved by applying the technique for extracting journal information from different publishers’ websites like Elsevier, BMJ, Nature, Wiley, etc. The same can be extended to get information from different domains provided the domain keywords are present explicitly in the web page.