

Web Data Extraction from Scientific Publishers' Website Using Heuristic Algorithm

Umamageswari Kumaresan

Assistant Professor, New Prince Shri Bhavani College of Engineering & Technology, Chennai, 600073, India
E-mail: umamage@gmail.com

Prof. Kalpana Ramanujam

Pondicherry Engineering College, Pillaichavady, Puducherry, India
E-mail: rkalpana@pec.edu

Received: 04 March 2017; Accepted: 05 June 2017; Published: 08 October 2017

Abstract—WWW is a huge repository of information and the amount of information available on the web is growing day by day in an exponential manner. End users make use of search engines like Google, Yahoo, and Bing etc. for retrieving information. Search engines use web crawlers or spiders which crawl through a sequence of web pages in order to locate the relevant pages and provide a set of links ordered by relevancy. Those indexed web pages are part of surface web. Getting data from deep web requires form submission and is not performed by search engines. Data analytics and data mining applications depend on data from deep web pages and automatic extraction of data from deep web is cumbersome due to diverse structure of web pages. In the proposed work, a heuristic algorithm for automatic navigation and information extraction from journal's home page has been devised. The algorithm is applied to many publishers website such as Nature, Elsevier, BMJ, Wiley etc. and the experimental results show that the heuristic technique provides promising results with respect to precision and recall values.

Index Terms—Structured data, Information Extraction, Deep web, wrapper, DOM Tree, template, JQuery, XPATH.

I. INTRODUCTION

World Wide Web is a powerful source of information which contains information in many desperate formats which highly affects the automated processing. Many business organizations require data from WWW for carrying out analytic tasks such as online market intelligence, product intelligence, competitive intelligence, decision making, sentiment analysis[1] etc. Often search engines are used to get the information from WWW. Search engines makes use of web spiders to crawl through the web and retrieve the links that might contain the information searched for, and presents the information to us in the form set of hyperlinks. Search engines are capable of retrieving information from surface web but not from invisible web or hidden which

requires form submission. From [3], it is clear that amount of pages not indexed by search engines is 400 to 550 times greater than the size of the surface web. Data analytics application requires extracting information from deep web and also from several heterogeneous web sites. No two web sites are similarly structured and therefore, extraction process has to deal with heterogeneity in terms of web technologies such as scripting languages like Java script, VB Script, languages used for beautifying web pages like CSS etc. In order to improve user's web experience by providing visual effects and sophisticated rendering, more and more complex structuring of web pages have been done using different versions of HTML, CSS and AJAX. This makes data extraction highly cumbersome task. Challenges faced by extractors include heterogeneous formats, changes in structure of web pages, introduction of more and more sophisticated technologies for enhancing UX etc. Certain commercial tools like Mozenda [28], Lixto [24], KDNuggets [21] provides GUI to guide the extraction process but the level of human intervention they require is high.

General picture of web data extractor steps is shown in figure 1. It involves four steps namely: Search, Locate, Filter and Extract. Searching refers to the problem of automatically locating target web pages given the URL of the web site. Locating refers to identifying data rich regions in the target web pages and filtering refers to the process of filtering the data rich nodes after eliminating noisy sections such as navigation links, pop-ups, advertisements etc. The last step extraction involves extracting the attribute-value pairs from the target web pages that defines the specification of the object.

A. Motivation

Most of the researchers and students face difficulty in finding appropriate journals for their research article publication. The information about journals are present in the journal home page and it requires going through sequence of hyperlinks to reach appropriate web page which contains detailed information about the journals such as Impact Factor, Five Year Impact Factor, SCIMAGO Journal Rank, Source Normalized Impact

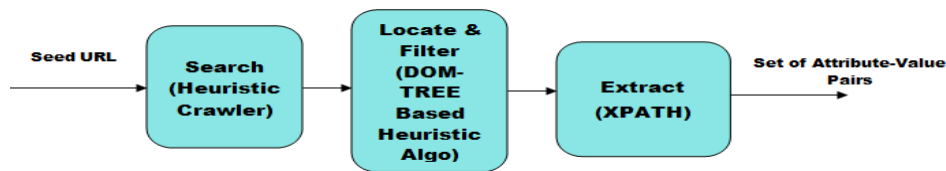


Fig.1. Steps in Web Data Extraction.

Factor etc. Manually carrying out the analysis is time consuming and therefore, a technique is proposed which automatically crawls through web pages in the journal web sites which are usually organized in alphabetical order A-Z. Once the appropriate web pages are determined, heuristic algorithm is used to determine the location of target information. Information is extracted and stored in structured form which facilitates carrying out different kinds of analysis on the extracted data. As shown in figure 1, in the first step a heuristic crawler is used to identify the URL of the journal home pages linked to the publishers' web site. In the next step, DOM tree of a page is generated and text nodes are filtered. Text nodes that match with the domain keywords are selected and the XPATH [39] is determined. It can be observed that values are located very close to the attribute name and therefore, this heuristic can be used to retrieve the corresponding value of the attribute. The XPATH determined for attribute value pairs are stored and can be used to locate the attribute value pairs in similarly structured target web pages. The extracted attribute-value pairs are stored in structured form onto RDBMS.

B. Problem Definition

The problem of Journal Information Extraction can be stated as follows: Given a seed URL of publishers' web site, determine the set of journal home pages $\{p_1, p_2, \dots, p_n\}$ which contains the set of keywords like SJR, SNIP, impact factor etc. representing the search object (journal), the problem is to extract the structured records pertaining to each search object (journal).

Problem: To extract the detailed specification of the journals.

Constraint: The web pages $\{p_1, p_2, \dots, p_n\}$ from a different publishers' web site might have heterogeneous templates.

The research article is organized as follows: Section 2 elaborates the state-of-the-art approaches available for web data extraction, Section 3 discusses the architecture and algorithms of journal information extraction system, Section 4 represents the experimental results and Section 5 concludes the article.

II. RELATED WORKS

Web data extraction is broadly classified into four types: hand-crafted, supervised, semi-supervised, and unsupervised. Initially manually created wrappers are used for extracting information from target pages such as TSIMMIS [13], W4F [32], WIEN [23] etc. These techniques involve creating wrappers using declarative languages for extracting the needed information. The

drawback of these techniques include high level human intervention needed in creating wrappers, error prone and changes in web page structure requires modifying the scripts. In order to overcome the problems encountered with hand crafted wrappers, supervised techniques such as IEPAD [6], OLERA [5] and Thresher [14] which involves automatic wrapper induction from labelled web pages came into existence. These techniques also face the limitations of requiring post efforts of annotating extracted data. In [8], a supervised technique is proposed which involves inferring wrapper dynamically from training sets. Training data contains labelled values generated by membership queries. The approach is highly scalable but requires optimizing the interaction with the crowd.

Commercial tools like ChickenFoot [27], Mozenda [28], Lixto [24] etc. are semi-supervised approaches which provides sophisticated GUI to guide the extraction process. Further research in the area of web data extraction, lead to the era of fully automatic or unsupervised web data extractors such as RoadRunner [7], EXALG [2], DEPTA [40], DELA [37], FIVATECH [20] and TRINITY [34]. RoadRunner [7] starts with a sample page and creates a Union Free Regular Expression representing wrapper. It matches each successive sample with the wrapper. It deals with the mismatches by generalizing the RE. This approach faces the difficulty of dealing with missing attributes. EXALG [2] uses identification of Equivalence class as a mechanism to deduce templates. It considers web page as a sequence of strings and identifies the sets of tokens having the same frequency of occurrence in every page which are known as equivalence classes. It filters those equivalence classes which are large and frequently occurring in most of the pages. FIVATECH [20] uses DOM trees of the web pages to deduce schema. They perform merging of the DOM trees into fixed/variant pattern tree. The pattern tree is used to deduce schema of the website. The experiments were conducted only on 2 or 3 web pages and the feasibility of the approach has not been proved for real world websites consisting of several hundreds of web pages. TRINITY [34] also performs page level extraction just like the previous approaches. TRINITY [34] like EXALG [2] considers web pages as a set of strings. It applies the string matching algorithm Knuth Pratt Morris [22] to deduce template. It involves generating a tree like structure called trinary tree in which each node has three child nodes prefix, separator and suffix. This tree guides the deduction of template. This approach does not work well with web pages having different formatting for the same data records and same sequence of tokens for all the attributes. These extractors

involve learning templates from sample web pages and using the template for extracting data. DWDE [26] uses tag tree algorithm for data extraction. It uses domain classification technique for web page retrieval based on user query. Information extraction from deep web using Repetitive Subject Pattern [36] is based on the hypothesis that information in web page is about a subject item and repetitive pattern around the subject items can be used to identify boundary. The limitation of this approach that it cannot be used for detail pages having a single subject item. Many semantic approaches came into existence. In [41], the information present in the leaf nodes is analyzed. Semantic Information Vector is generated for non-leaf nodes and semantic tree matching technique is used to extract product information. [12], [31], [10] and [9] involve transforming and forwarding queries to multiple web sites

and integrating the local results. Another semantic extraction technique [15] is based on R tool. They have developed an extraction language R which represents extraction rules and its syntax is similar to CSS selectors. The drawbacks of grammar based approaches include huge amount of manual labor is required for writing extraction rules and also it requires domain expertise. In [17], a new approach for data extraction using lexical database WordNet has been proposed. Correct data rich region is determined by using WordNet. The technique also deals with different type of data records namely single-section data record, multiple section data records and loosely structured records. In [33], the authors have proposed a framework for finding appropriate datasets for research problem. It involves using web intelligence for data set names extraction.

Table 1. Comparison of state-of-the-art approaches for web data extraction.

Extraction System	Level of Human Intervention	Applicability	Granularity	Technique
TSIMMIS[9]	Hand crafted	Semi-structured	Record level	None
W4F[21]	High	Semi-structured	Record level	Extraction rules
WIEN[16]	High	Semi-structured	Record level	Uses a family of 6 wrapper classes
OLERA[4]	Semi-supervised	Template	Record level	String Alignment
IEPAD[5]	Semi-supervised	Template	Record level	Pattern Mining and String Alignment
Thresher[9]	Semi-supervised	Template	Record level	Uses tree edit distance between DOM subtrees
DELA[23]	Unsupervised	Template	Record level	Pattern Mining
DEPTA[29]	Unsupervised	Template	Record level	Partial Tree Alignment
RoadRunner[6]	Unsupervised	Template	Page level	String Alignment
FIVATECH[13]	Unsupervised	Template	Page level	Tree Matching, Tree Alignment and Mining
EXALG[2]	Unsupervised	Template	Page level	Equivalence class and Differentiating Roles
TRINITY[22]	Unsupervised	Template	Record level	Trinary tree

The state-of-the-art web data extraction techniques are shown in table 1. These approaches are designed for general purpose web data extraction and have limitations with respect to applicability on differently structured web pages from heterogeneous sites and also for websites having different ordering of attributes. Often, data analytics application requires fetching data from multiple heterogeneous websites. In this work, information extraction from publisher's web sites has been done by focusing on automatic crawling of different journal home pages, using heuristics to locate journal attributes such as title, SJR, SNIP, Impact factor, about the journal, ISSN etc.

III. ARCHITECTURE OF JOURNAL INFORMATION EXTRACTION SYSTEM

The overall architecture of the journal information extraction system is shown in figure 2. The first step is to crawl through the journal web site given by seed URL

and to extract the URLs of the target pages that might contain the journal information. A heuristic algorithm is used for extracting information and then, the information is stored in structured form in RDBMS. The RDBMS can then be queried to obtain the relevant information. Figure 3 shows the phases of the proposed system.

URL of the Publisher's web site is given as input to the system. Heuristic crawling algorithm is designed based on the fact that the Journal's links are arranged in chronological order from A-Z. Clicking on the alphabet results in displaying a list containing names of journals whose name starts with the specified alphabet. Again clicking on a specific link displays the journal detail page which might contain a link to Journal's home page. Thus, it requires on part of the user to navigate through a series of links in order to reach journal's home page. These steps are automated by the heuristic crawler and the URL of all the journals linked to the publisher's web site is retrieved.

In phase 2, Heuristic based WDE technique is used for the extraction of journal features such as title, ISSN, SJR,

SNIP, Impact Factor etc. from journal's home page. First step is to construct a DOM tree representing journal's home page using JSoup [19] API. The algorithm is based on the fact that journal metrics are displayed as part of rendered page and therefore, they constitute text nodes in the DOM tree. Instead of processing the entire DOM tree, in this work only the leaf nodes are filtered and compared with the domain keywords such as Impact Factor, SJR and SNIP etc. to identify the location of target information. Once the location of attributes are identified the values are extracted based on the observation that the

values are available as sibling or child node to the attribute node. Location is represented as XPATH expression [39], a language used to locate a node in any XML document. Once the XPATH expression is determined for all the journal features expressed as attribute value pairs, the same XPATH can be used to extract information from remaining target pages since they are generated using same server side template. The extracted information is stored in structured form in RDBMS. Finally, a GUI is designed to answer user queries in a single interaction with the system.

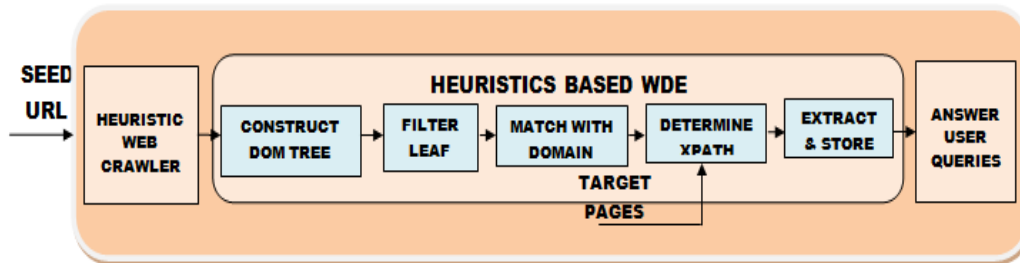


Fig.2. Overall Architecture of the Journal Information Extraction System.

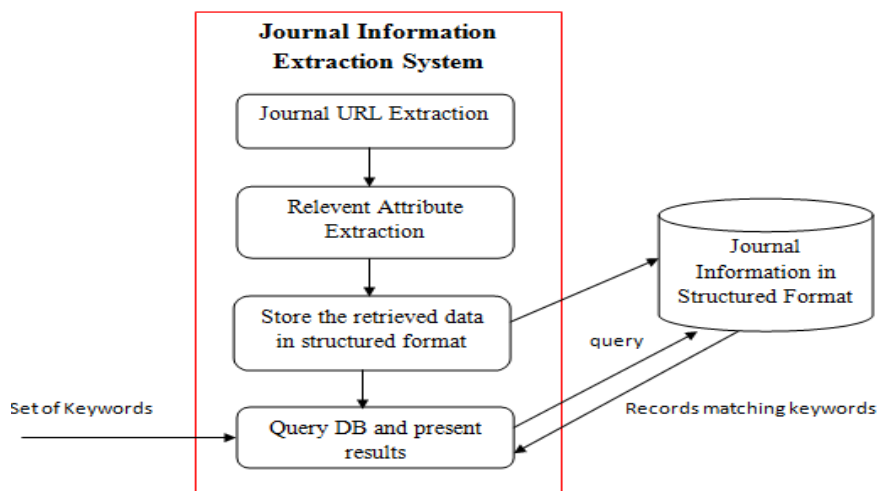


Fig.3. Phases in Journal Information Extraction System.

Wiley Online Library



Fig.4. Organization of Journal Home Page URLs

The proposed approach is based on the observation that the journal home pages linked to publishers' web site are well structured and they are generated using same server-side template. If the location of target data is identified for a single web page (XPATH for attribute-value pairs)

from the publisher's web site, then the same XPATH can be used for extraction from similarly structured pages. It is clear from table 2 representing the journal information obtained from Nature [29] publisher's web site, the XPATH corresponding to attribute value pairs ISSN and

Impact Factor remains the same.

A. *Mathematical Model*

Heuristic Algorithm for Automatic Navigation

Given the URL of publisher's site url_p , navigate to the site and extract the set of URLs arranged in alphabetical order i.e. A-Z. Navigating to each URL again, produces a set of journal records containing information such as title, ISSN, URL of the journal's home page etc. To collect all target URLs for all the journals ordered by A-Z. The set of target URLs is denoted by

$$H = \{hi, 1 \leq i \leq n\} \tag{1}$$

Each journal home page hi is parsed to collect the needed attributes such as title, ISSN, Impact Factor, Source Normalized Impact Factor, SCImago Journal Rank etc.

B. *Algorithm for target URL extraction*

The algorithm takes as input base URL of the publisher's web site. The algorithm is based on the observation as shown in figure 4, that all the journal information are organized in chronological 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.

```

Algorithm extractURLs (baseURL)
//Input: Seed_URL
//Output: list of target_URLs
    
```

```


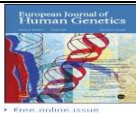
begin
create useragent and visit baseURL
useragent.visit(basehref);
extract all anchor elements
if anchor.text matches  $^{[a-z]\$}^{[A-Z]\$}$ 
target_urls.add(anchor.href)
return target_urls
end
    
```

C. *Heuristic Algorithm for Data Extraction*

The intent of this algorithm is to identify location of target attributes, extract attribute values, annotate and store the extracted records containing attribute values into database. The algorithm is based on the heuristics that all the needed information is visible content displayed in the content area of the browser. In the DOM tree representing the web page, all the text nodes will be present as leafnodes. Visit each URL and obtain the DOM tree of the web page. Find the leaf nodes using the algorithm GetLeafNodes and check whether it is a text node. If they match with the field names such as SJR, SNIP, Impact Factor etc. then determine its XPath using the procedure getFullXPath. Convert XPath to selector string using the algorithm convert_XPath_to_Selector. The same selector string is used to carry out extraction from similarly structured web pages. The algorithms are detailed below:

The algorithm Extract takes as input seed URL of the publishers' website and calls the procedure extractURLs to extract the URLs of the journal home pages. It then calls procedure display_journal_metrics for extracting attribute value pairs from each target page. It establishes database connectivity in order to store the structured records to RDBMS.

Table 2. Similarly structured web pages and the XPATH of attributes.

NATURE JOURNALS[29]	XPATH
 <p> APS Asian Pacific Journal of Biophysics and Biotechnology * Free online issue Volume 37, No 5 May 2016 ISSN: 1673-4033 EISSN: 2745-7254 2014 Impact Factor 2.9125 Multidisciplinary 90/204 Pharmacology & Pharmacy </p>	<p>IMPACT FACTOR</p> <p>/html/body[@id='home']/div[@id='constrain']/div[@class='constrain']/div[@id='content-journalnav']/div[@id='content']/div[@class='journal-details']/p[2]/span[@class='impact']</p> <p>ISSN</p> <p>/html/body[@id='home']/div[@id='constrain']/div[@class='constrain']/div[@id='content-journalnav']/div[@id='content']/div[@class='journal-details']/p[1]</p>
 <p> European Journal of Human Genetics * Free online issue Volume 24, No 6 June 2016 ISSN: 1098-9823 EISSN: 1470-9430 2014 Impact Factor 4.9947 Multidisciplinary 90/204 Pharmacology & Pharmacy </p>	<p>IMPACT FACTOR</p> <p>/html/body[@id='home']/div[@id='constrain']/div[@class='constrain']/div[@id='content-journalnav']/div[@id='content']/div[@class='journal-details']/p[2]/span[@class='impact']</p> <p>ISSN</p> <p>/html/body[@id='home']/div[@id='constrain']/div[@class='constrain']/div[@id='content-journalnav']/div[@id='content']/div[@class='journal-details']/p[1]</p>

```

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
    
```

The algorithm display journal metrics takes as input target url. It checks whether the ISSN XPATH and Impact Factor XPATH are 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. The XPATH is then converted to JQUERY selector string [18], since the Jaunt API [16] supports only selector string for locating node in the DOM tree. The same

XPATH is used for extraction from remainder of the web pages linked to the publisher's web site.

Algorithm display_journal_metrics (target_url)

```
//Input: target url
//Output: impact_factor,issn
begin
  useragent.visit(target_url);
  issn_pattern = "\\d\\d\\d\\d\\d-\\d\\d\\d\\d\\d\\d\\d\\d\\d\\d\\dX"
  impact_pattern = "\\d\\.\\d{1,3}\\d\\d\\d\\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
  end 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
  return impact_factor,issn
end
```

The algorithm getLeafNodes 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.getChildNodes()
  begin
    if node.getType() equals Node.TEXT_TYPE then
    leaf_nodes.add(node)
    else
    getLeafNodes(node)
    end if
  return leaf_nodes
end
```

The algorithm getFullXPath constructs XPATH by taking the node for which XPATH has to be determined as input. 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 corresponding selector string using the algorithm convert_xpath_to_selector.

Algorithm getFullXPath (Node n)

```
//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") + "]")
  else if (e.hasAttribute("name")) then
  XPath.append("[@name=" + e.getAt("name") + "]")
  end if
  if (node.getName().equals("table")) then
  XPath.append("/tbody")
  end if
  end while
  return XPath
end
```

Algorithm convert_xpath_to_selector takes XPATH as input and produces JQUERY selector [12] as output. JQUERY selector [18] like XPATH is used to select nodes from HTML documents using attribute such as name, id, class etc. Space in XPATH is replaced by “.”, “[@class=” by “.”, “[@id=” by “#” and “/” by “>”.

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
end
```

IV. EXPERIMENTAL RESULTS

The experiment was conducted on Elsevier [11], BMJ [4], Nature [29], Wiley [38], Oxford Journal [30] and

Springer Open Journal [35] publishers' web site, in which journals are organized in lexicographical order and clicking on each link gives a listing of journals starting with the alphabet. To know the journal information, one more level of navigation is needed i.e. About Journal hyperlink has to be clicked. The proposed module performs appropriate level of navigation, fetches the structured data embedded in unstructured HTML pages and then, stores it onto RDBMS. The extraction results are shown in the table 3. The three classic metrics of information retrieval namely precision, recall and F-Measure are used to evaluate the effectiveness of this

technique in journal information extraction. The graph in figure 5, shows that for well-structured web sites like Elsevier [11] and Wiley [24], accuracy of extraction is good whereas for websites like BMJ [4] and Nature [28] which contains certain differently structured pages, the use of XPATH expression derived from a sample, does not work well for all the target web pages. In future, machine learning approaches can be studied and adapted to information extraction which may learn new templates when encountered and therefore may increase the accuracy of extraction in case of websites having heterogeneous web pages linked to it.

Table 3. Precision, Recall and F-Measure values for various attributes extracted from journals associated with various Publishers' sites.

	Precision	Recall	F-Measure
Elsevier			
SNIP	1	0.990991	0.995475
SJR	1	0.990991	0.995475
IMPACT FACTOR	1	0.993243	0.99661
IMPACT FACTOR FOR 5 YEARS	1	0.898649	0.946619
BMJ			
IMPACT FACTOR	0.955556	0.826923	0.886598
ISSN	0.888889	0.740741	0.808081
Nature			
IMPACT FACTOR	0.896552	0.855263	0.875421
ISSN	0.951724	0.907895	0.929293
Wiley			
IMPACT FACTOR	1	1	1
ISSN	1	1	1
Oxford			
IMPACT FACTOR	0.961538	0.892857	0.925926
ISSN	0.909091	0.833333	0.869565
EISSN	0.928571	0.902778	0.915493
Springer			
IMPACT FACTOR	0.977778	0.942857	0.96
ISSN	0.972222	0.933333	0.952381

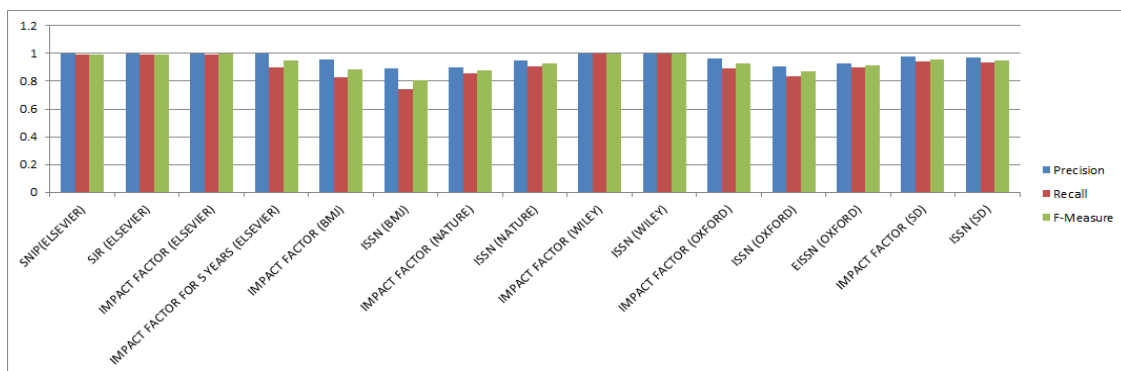


Fig.5. Comparison of Precision, Recall and F-Measure values for attributes extracted from journal home pages associated with various Publishers' web site.

A. Advantages

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

1. Unlike other DOM tree based techniques [20], [40], it does not require processing the entire DOM tree to identify location of attribute value pairs. Usually they

represent text nodes and text nodes are always leaf nodes in the DOM tree.

2. It requires linear time since it does not involve complex pattern matching or string alignment algorithms as in [3] [7].

3. Running time is greatly reduced because once the XPATH is determined for a journal home page, the same XPATH can be used for extraction from similarly

structured journal home pages linked to the publishers' web site.

4. It can be extended for data extraction belonging to different domains provided domain keywords and their synonyms are known.

B. Limitations

1. It does not work well in case of differently structured journal pages linked to single publishers' web site.

2. It does not handle ordering of attributes

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

V. CONCLUSION

Extracting structured data from hidden or deep web is a challenging problem because of the intricate structure of the web pages which changes from one web site to another. Researchers and publishers tend to use search engines to find appropriate journals for research article publication which involves a tedious task of navigating through a series of links to obtain detailed information about the journals such as SJR, SNIP, Impact Factor, scope etc. The proposed framework uses a heuristic based approach for navigating the publishers' web site to find the journal home pages and to extract the needed information. The versatility of the proposed work is proved by making use of the approach for extracting journal information from different publishers' web site like Elsevier, BMJ, Nature, Wiley, Springer and Oxford journals. The same can be extended to get information from different domains provided we know the domain keywords used across sites. The limitation associated with our approach is that the data to be extracted should be enclosed within HTML tags. If the journal metrics are available as part of text, then the system won't be able to recognize and extract the data. In future, we are planning to incorporate NLP techniques in order to recognize metrics available as part of text nodes and also machine learning techniques to learn new templates in an ad hoc manner. We are also planning to deduce template from DOM tree by applying Adaptive Weighted Frequent Itemsets algorithm[25].

REFERENCES

- [1] Akshi Kumar, Teeja Mary Sebastian (2012). Sentiment Analysis: A Perspective on its Past, Present and Future, *IJISA*, vol.4, no.10, pp.1-14, 2012. DOI: 10.5815/ijisa.2012.10.01
- [2] Arasu, A., & Garcia-Molina, H. (2003). Extracting structured data from Web pages. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, California, pp.337-348.
- [3] Bergman, M. (2001). The deep Web: Surfacing hidden value. *The Journal of Electronic Publishing*, Vol. 7.
- [4] BioMed Journal. Accessed May 18, 2016 from <https://www.biomedcentral.com/journals>
- [5] Chang, C.-H., & Kuo, S.-C. (2004). OLERA: A Semi-Supervised Approach for Web Data Extraction with Visual Support. *IEEE Intelligent Systems*, 19(6), pp. 56-64.
- [6] Chang, C.-H., & Lui, S.-C. (2001). IEPAD: Information Extraction based on Pattern Discovery. *Proceedings of the Tenth International Conference on World Wide Web (WWW)*, Hong-Kong, pp. 223-231.
- [7] Crescenzi, V., Mecca, G., & Merialdo, P. (2002). Roadrunner: Automatic Data Extraction from Data-Intensive Websites. *SIGMOD*, pp. 624-624.
- [8] Crescenzi, V., Merialdo, P., & Qiu, D., (2013). A Framework for Learning Web Wrappers from the Crowd. *WWW'13 Proceedings of the 22nd international conference on World Wide Web*, pp. 261-272.
- [9] Dönz, B., Bruckner, D., (2013). Extracting and Integrating Structured Information from Web Databases Using Rule-Based Semantic Annotations. *Industrial Electronics Society IECON 2013-39th Annual Conference of the IEEE*, pp. 4470-4475.
- [10] Dönz, B., Boley, H., (2014). Extracting Data from the Deep Web with Global-as-View Mediators Using Rule-Enriched Semantic Annotations. *Proceedings of the RuleML 2014 Challenge and the RuleML 2014 Doctoral Consortium hosted by the 8th International Web Rule Symposium*, vol. 1211, pp. 1-15.
- [11] Elsevier Journals. Accessed May 18, 2016 from <https://www.elsevier.com/journals/title/a>
- [12] Furche, A., Gottlob, T., Grasso, G., Orsi, G., Schallhart, G., Wang, C., (2012). AMBER: Automatic Supervision for Multi-Attribute Extraction. *CoRR abs/1210.5984 2012*.
- [13] Hammer, J., McHugh, J., & Gracia-Molina, H. (1997). Semistructured data: The TSIMMIS experience. *Proceedings of the First East-European Symposium on Advances in Databases and Information Systems (St. Petersburg, Russia)*, pp. 1-8.
- [14] Hogue, A., & Karger, D. (2005). Thresher: Automating the Unwrapping of Semantic Content from the World Wide. *Proceedings of the 14th International Conference on World Wide Web (WWW)*, Japan, pp. 86-95.
- [15] Janosi-Rancz, K.-T., Lajos, A. (2015). Semantic Data Extraction. *Elsevier Procedia Technology*, Vol. 19, pp. 827-834.
- [16] Jaunt API. Accessed May 18, 2016 from <http://jaunt-api.com/>
- [17] Jer Lang Hong (2011). Data Extraction for Deep Web using WordNet. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, Vol. 41, No. 6, pp. 854 – 868.
- [18] JQuery selector, Accessed May 18, 2016 from <https://api.jquery.com/category/selectors/>
- [19] JSOUP API. Accessed May 18, 2016 from <https://jsoup.org/>
- [20] Kayed, M. & Chang, C.-H. (2010). FiVaTech: Page-level web data extraction from template pages. *IEEE Transactions on Knowledge and Data Engineering*, 22(2), pp. 249-263.
- [21] KDNuggets. Accessed May 18, 2016 from <http://www.kdnuggets.com/>
- [22] Knuth, Donald E., James H. Morris, Jr, and Vaughan R. Pratt. (1977). Fast pattern matching in strings. *SIAM journal on computing* 6.2 pp. 323-350.
- [23] Kushmerick, N., Weld, D., & Doorenbos, R. (1997). Wrapper Induction for Information Extraction. *Proceedings of the Fifteenth International Conference on Artificial Intelligence (IJCAI)*, pp. 729-735.
- [24] Lixto. Accessed May 18, 2016 from <http://www.lixt.com/>
- [25] Long Nguyen Hung, Thuy Nguyen Thi Thu, Giap Cu Nguyen (2015). An Efficient Algorithm in Mining

- Frequent Itemsets with Weights over Data Stream Using Tree Data Structure, *Int'l Jour. of Intelligent Systems and Applications*, Vol. 7, No. 12, pp. 23-31.
- [26] Manoj, D.-S., Sonune, G., Meshram, B.-B. (2013). Understanding the Technique of Data Extraction from Deep Web. (*IJCSIT*) *International Journal of Computer Science and Information Technologies*, Vol. 4 (3), pp. 533-537.
- [27] Michael Bolin. (2005) End-user programming for the web. Master's thesis, Massachusetts Institute of Technology, May 2005.
- [28] Mozenda. Retrieved 18 May, 2016 from <http://mozenda.com/>
- [29] Nature Journal. Accessed May 18, 2016 from <http://www.nature.com/siteindex/>
- [30] Oxford Journal. Accessed April 17, 2017 from https://academic.oup.com/journals/pages/journals_a_to_z
- [31] Pavai, G., Geetha, T.-V., (2013). A Unified Architecture for Surfacing the Content of Deep Web Databases. *Proc. of Int. Conf. on Advances in Communication, Network, and Computing* pp. 35 – 38.
- [32] Sahuguet, A., Azavant, F. (2001). Building Intelligent Web Applications using Lightweight Wrappers. *IEEE Transactions on Data and Knowledge Engineering*, 36(3), pp. 283-316.
- [33] Singhal, A., & Srivastava, J. (2013). Data Extract: Mining Context from the Web for Dataset Extraction. *International Journal of Machine Learning and Computing*, Vol. 3, No. 2, pp 219 – 223.
- [34] Sleiman, H.-A., Corchuelo, R. (2014). Trinity: On Using Trinary Trees for Unsupervised Web Data Extraction. *IEEE Transactions on Knowledge and Data Engineering*, 26(6), pp. 1544-1556.
- [35] Springer Journals. Accessed on April 17, 2017 from <https://www.springeropen.com/journals-a-z#A>
- [36] Thamviset, W. & Wongthanavasut, S. (2014). Information Extraction for Deep Web using Repetitive Subject Pattern. *World Wide Web* (2014) 17: 1109. doi:10.1007/s11280-013-0248-y
- [37] Wang, J., & Lochovsky, F. - H. (2003). Data extraction and Label Assignment for Web databases. *Proceedings of the Twelfth International Conference on World Wide Web (WWW)*, Budapest, Hungary, pp. 187-196.
- [38] Wiley Journals. Accessed on May 18, 2016 from <http://onlinelibrary.wiley.com/browse/publications>
- [39] XPATH, Accessed May 18, 2016 from <https://www.w3.org/TR/xpath/>
- [40] Zhai, Y., & Liu, B. (2005). Web Data Extraction Based on Partial Tree Alignment. *Proceedings of the 14th International Conference on World Wide Web (WWW)*, Japan, pp. 76-85.
- [41] Zhou, S., Zhang, S., & Karypis, G., (2012). Automated Web Data Mining Using Semantic Analysis. *ADMA 2012*, LNAI 7713, pp. 539–551.

University in 2010, and currently pursuing Ph.D. degree in Computer Science and Engineering in Pondicherry Engineering College respectively. Her research interests include web mining, web data extraction, information security and sentiment analysis. She is a member of IAENG.



Kalpana Ramanujam is currently working as Professor in the Department of Computer Science and Engineering at Pondicherry Engineering College, Puducherry, India. She received her B.Tech. degree in Computer Science and Engineering from Pondicherry University, Puducherry, India in the year 1996 and M. Tech. degree in Computer Science and Engineering from Pondicherry University, Puducherry in 1998. She completed her Ph.D in Computer Science & Engineering in the year 2013 in the field of Parallel Computing Systems. She joined as Lecturer in Department of Computer Science & Engineering, Pondicherry Engineering College, Puducherry in the year 2000. Subsequently she was promoted as Assistant Professor in the Department of Computer Science & Engineering, Pondicherry Engineering College, Puducherry in the year 2007 and elevated as Associate Professor in the year 2010. She is presently holding the post of Professor. Her areas of interest include Parallel Computing Systems, High Performance Computing, Web services and Distributed Computing. She has published more than 30 research papers in International Journals / Conferences. She is also a member of ISTE.

How to cite this paper: Umamageswari Kumaresan, Kalpana Ramanujam, "Web Data Extraction from Scientific Publishers' Website Using Heuristic Algorithm", *International Journal of Intelligent Systems and Applications(IJISA)*, Vol.9, No.10, pp.31-39, 2017. DOI: 10.5815/ijisa.2017.10.04

Authors' Profiles



Umamageswari Kumaresan is currently working as assistant professor in the Dept. of IT, at New Prince Shri Bhavani College of Engineering and Technology. She received her B.Tech in Computer Science and Engineering from Pondicherry Engineering College in 2005, her M.Tech in Computer Science and Engineering from Bharath