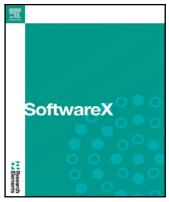
SoftwareX 24 (2023) 101569

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

WebCollectives: A light regular expression based web content extractor in Java

Hayri Volkan Agun

Bursa Technical University, Bursa, Turkey

ARTICLE INFO

Keywords:
Focused crawler
Web content extractor
HTML parsing
Regular expressions

ABSTRACT

Conventional web crawling methods typically involve a sequence of distinct steps for downloading and extracting web content. A noteworthy limitation of these conventional crawling approaches is their lack of a focus-based crawling strategy. The software introduced in this paper, known as WebCollectives, introduces a straightforward crawling approach by integrating content extraction into a hierarchical regular expression definition model. Furthermore, it streamlines the crawling process through a pipeline-oriented framework, emphasizing focus-based link extraction. This crawler employs either a configurable Selenium mechanism or a direct HTTP GET method to fetch web pages. Subsequently, it undergoes an extraction process based on hierarchical regular expressions. Notably, Selenium allows for adaptable JavaScript functions to navigate web pages effectively. The content extraction generates XML structures from diverse types of content. Comparative analysis with the standard DOM (Document Object Model) reveals that the proposed approach yields significant improvements in extraction efficiency and requires fewer lines of code. Specifically, it outperforms non-recursive standard DOM hierarchy definitions in terms of both extraction speed and code complexity.

Code metadata

Current code version	1.5
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00372
Permanent link to Reproducible Capsule	
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Java 11 and Scala 2.11
Compilation requirements, operating environments	Java 11, Maven, Linux, Windows
If available Link to developer documentation/manual	https://github.com/volkanagun/WebCollectives.git
Support email for questions	volkanagun@gmail.com

1. Motivation and significance

Text data is the primary input in text and data mining applications. In many applications, web provide the input source in the form of a web page [1]. Web pages contain diverse set of content as images, tables, and text and it is represented as hypertext markup language (HTML). HTML is a well-structured text content but it is not suitable directly to be used by sample based applications. The content must be scrapped according to the task. Using the web content directly from web requires two steps: download and extract. Web crawler is designed for downloading the HTML pages from web domains through traversing the links [2]. Downloading is done through HTTP Get or Selenium API. However, most of the crawlers do not decide on the links and they waste time through collecting advertisements. On the other hand, traditional web content extraction techniques are named as boilerplate removal [3], web scrapping [4], or content extraction [5]. Extraction is done through either document object model (DOM) based parsing [3], or light weight scrapping techniques [6]. DOM approach creates a structured hierarchical node view of the web page and can be queried.

E-mail address: hayri.agun@btu.edu.tr.
URL: https://sayfam.btu.edu.tr/en/hayri.agun.

https://doi.org/10.1016/j.softx.2023.101569
Received 8 June 2023; Received in revised form 14 October 2023; Accepted 17 October 2023
Available online 24 October 2023

However, it is slow because all the irrelevant hierarchical HTML tags including tag contents are retrieved from web during parsing. On the other hand scrapping approaches do not necessarily rely on DOM and are efficient.

WebCollectives does not use DOM to parse the HTML elements. Instead, it use HTML as a text source and process only the relevant HTML structure through scrapping. Traditional scrapping approaches are useful to scrap directly from HTML without DOM construction but they search flat content and not designed to capture the hierarchical structure. WebCollectives benefits the advantages of both DOM based models and scrapping. In this respect, a tree of regular expression structure is first defined where the parent and siblings, the matching regular expressions, the starting and ending regular expression patterns and if defined the nth item in the sub-tree are specified imperatively. All the defined conditions must be matched to extract the relevant content. The defined hierarchical extraction also recursively eliminates all the sub-steps and builds an XML structure based on the defined tags in the tree. In other words, it does not require a post-processing step to re-construct the XML from the parsed structure. Additionally, a Selenium browser can be embedded for navigating in web pages where is impossible to navigate by JavaScript function calls and also a JSON based content can be easily used in the same API model.

Traditional scrapping approaches are very simple for extracting structural components [6]. Each scrapping rule must specify the pattern of the exact target HTML element [7]. In WebCollectives, a hierarchical definition model is used to capture the target HTML element through both the relative positioning of the element to other defined patterns and the specified regular expression patterns. Thus, unlike simple scrapping approaches, it reduces the search space recursively navigating through defined pattern space and can use general pattern definitions. For example, a title of a web page can be defined by a h1 HTML markup in several positions of the HTML document. The correct title cannot be distinguished by a single rule without using the relative position. In WebCollectives, either a parent HTML element or a sibling HTML element can be used to define the relative position of the target HTML element. This approach is not only useful to position the elements but also is very easy for defining models that captures the correct HTML structure.

2. Definition and examples

Traditional crawling and content extraction approaches use HTML documents for searching the links and traversing the links in order and it repeats this process until no new links is extracted from the domain. This crawling methodology includes a shallow link extraction process. WebCollectives adopts a methodology closely resembling this traditional technique. Nevertheless, in WebCollectives, the focus shifts from link extraction to the definition of rule templates, which explicitly specify the desired content to be extracted. These rule templates consist of multiple rules structured according to a hierarchical orientation. Each rule is designed to match specific HTML elements such as links, tables, images, and the like. The rule matching adheres to the hierarchical pattern definition, which is manually crafted by a human expert. Within WebCollectives, a user-friendly structure for rule definition is proposed, allowing for the use of hierarchy and labels to format the extracted content into an XML document.

There are three main class definitions that is used in the crawling and extraction. First one is named as WebTemplate. It is container class for the link generator (WebSeedGenerator) and for the pattern extracter (LookupPattern). Download and extraction processes are combined together in WebTemplate definition. The WebTemplate is responsible to traverse the generated links by the WebSeedGenerator. The extraction template including the final format of the extracted content is defined by the LookupPattern. LookupPattern uses a hierarchical regular expression based template matching definition model.

A comprehensive overview of the entire crawling and extraction process is depicted in Fig. 1. The overview of WebCollectives can be explained in three steps. First, the generator creates the links, and the links are downloaded. Second, the downloaded HTML content is used to extract the relevant content by the rules defined as a single hierarchical LookupPattern. Third, the results are obtained in LookupResult and they are validated according to the required fields. If there are extracted links, they can be used to download new web pages.

A simple example for an extraction process is given in Fig. 2. A WebTemplate downloads a web page and constructs an HTML content. The parsing step uses the LookupPattern and extracts LookupResult. The skip labeled LookupPatterns are not extracted but used for parsing the HTML content. The result of this approach constructs an XML file.

The crawling process is defined in the WebFlow class. WebFlow contains a separate WebTemplate for each WebDomain. WebTemplate is a container class to control the extraction process. One of the main advantages of the WebTemplate, it interface all the crawling process in one single entry point. Thus, it is easy to combine the HTTP based download after a selenium based navigation. The WebSeedGenerator is a url generator class defined to generate several different urls from a fixed domain. WebSeedGenerator can be combined with a Selenium browser to navigate inside web pages through JavaScript calls. There are vast number of operations for Selenium based crawling. For clicks a css query can be used to simulate the button or href clicks for fixed number of times. This approach is very useful for closing advertisements, or navigating via next page JavaScript calls.

The regular expression based parsing is done recursively, and the output is constructed instantly in the same structure of the pattern tree. The labels in the pattern definition are used to create XML tags where as the functionality of the tag is provided in type. For example, in cases where the output content must be flattened, the SKIP type is used. A long with the SKIP type, its opposite type is called TEXT. In static string constructions with TEXT type, the static string value can be exported in the XML output without using pattern matching.

The process of crawling and extraction, realized in three straightforward steps, simplifies the tasks of web crawling and scraping. Illustrations of object constructions for WebTemplate, WebFunctionCall, and WebSeedGenerator are showcased in the subsequent code snippets. When instantiating a WebTemplate, it involves specifying the download directory, the prefix for the resulting XML file, and the associated domain. Furthermore, each WebTemplate instance is required to incorporate a LookupPattern for its operation.

```
WebTemplate articleTemplate = new
     WebTemplate(LookupOptions.TURKISHARTICLEDIRECTORY,
     "article-text", domain)
               .setType(LookupOptions.ARTICLEDOC)
               .setLookComplete(false)
               .setThreadSize(1)
               .setSleepTime(2000L)
               .setDoFast(false)
               .setDomain(domain)
               .setHtmlSaveFolder(LookupOptions.
                 HTMLDIRECTORY)
               .setMainPattern(articleLookup)
               .setForceWrite(false);
```

In the context of initializing the Selenium web browser, it is imperative to establish a WebFunctionCall instance. The Selenium browser relies on the Selenium API to navigate web pages and plays a fundamental role in web crawling and page retrieval processes, albeit without direct implications for the extraction phase. Notably, a variety of WebFunctionCall types are available for use. A commonly employed function call involves scrolling down the page, particularly in situations where page reloading hinges on a downward scroll of the browser. Additionally, there exist other WebFunctionCall types, such as button

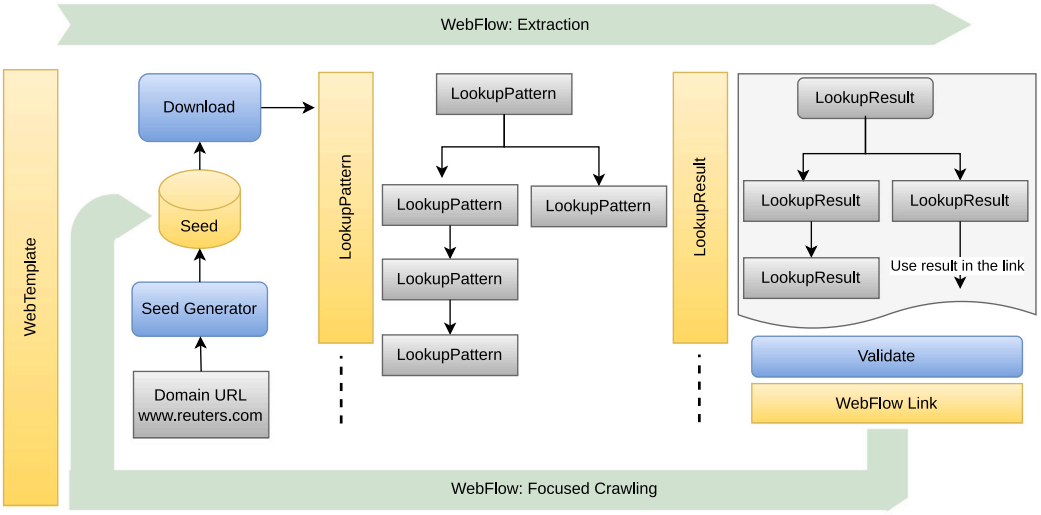


Fig. 1. A general overview of the crawling, and extraction processes for a single seed URL. Each WebTemplate has single LookupPattern. Multiple HTML contents can be feed into this LookupPattern. The results are stored in LookupResult.

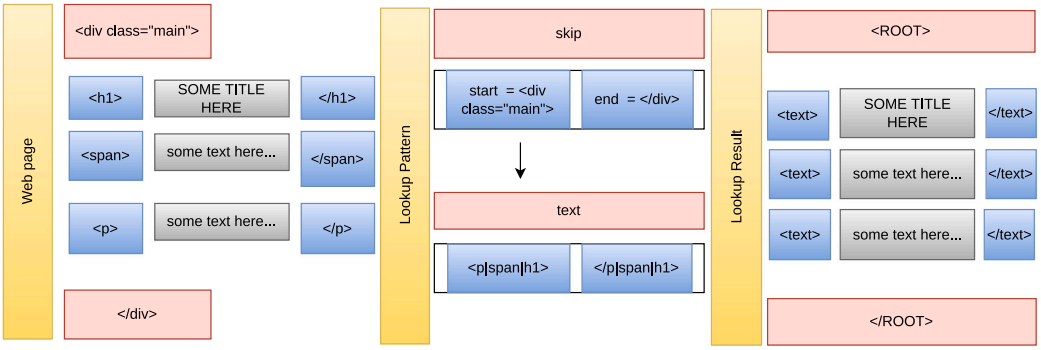


Fig. 2. A general overview of the content of the outputs in each step. Defined LookupPattern is used to extract an XML content from a web page. The skip label is only used to satisfy the three structure.

clicks. In cases necessitating the execution of multiple JavaScript calls, these can be sequentially combined through the utilization of the WebFunctionSequence type.

```
//Javascript function call
WebFunctionCall functionCall = new
     WebExecuteJS("loadMoreNews()")
      .setDoStopOnError(true).setWaitTime(2000);

//call scroll down 1 time
WebFunctionCall scrollCall = new
     WebFunctionScrollHeight(1)
               .setWaitTime(1500);

//Repeat the sequence call 5 times
int pageCount = 5;
// scroll and function JS calls
WebFunctionCall sequenceCall = new
     WebFunctionSequence(pageCount, scrollCall,
     functionCall)
               .setDoFirefox(true)
               .setWaitBetweenCalls(1000L)
               .initialize()
               .setWaitTime(1000);
```

For the generation of Uniform Resource Locators (URLs), various mechanisms can be employed, including date-based generators, prefix generators, and suffix generators. The code snippet provided offers an

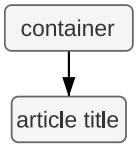


Fig. 3. An example pattern hierarchy.

illustration of a suffix generator known as 'WebCountGenerator.' This generator is designed to produce a series of suffixes for a given seed URL based on user-defined parameters, including a suffix string and a specified range. It is also possible to combine multiple suffix generators by utilizing a list collection.

```
//Several url generators can be combined.
List<WebSuffixGenerator> suffixGenerators = new
     ArrayList<>();
//This adds suffix in the form of KategoriNo=1..354
suffixGenerators.add(new WebCountGenerator(1, 354,
     "&KategoriNo="));
//This adds suffix in the form of &Page=1..5 to generated
     urls
//by the previous suffix definition
suffixGenerators.add(new WebCountGenerator(1, 5,
     "&Page="));
```

The WebTemplate serves as the central component for both web crawling and the extraction of web content. An example is given in the following code snippet. Within this class, various elements such as URL suffix generators, domain URLs, seed URLs, and JavaScript functions are initialized. Furthermore, the principal pattern is tasked with the responsibility of extracting content from the web pages that have been downloaded. By merging the processes of web crawling and web content extraction, it becomes feasible to tailor these procedures to suit the unique characteristics of a particular web domain.

```
//Previously defined function call
WebFunctionCall functionCall = function1();
//Previously defined seed generator
WebSuffixGenerator generator = generator1();
WebTemplate linkTemplate = new
    WebTemplate(LookupOptions.ARTICLEDIRECTORY,
    "article-links", "news.sky.com");
//Main seed to be used as input to seed generator
linkTemplate.addSeed("news.sky.com/world");
//Use Selenium API for function calls
linkTemplate.setFunctionCall(functionCall);
linkTemplate.setSuffixGenerator(generator);
```

By integrating the steps of WebTemplate initialization and LookupPattern definitions, a crawler downloads relevant content, parses the HTML data, and extracts the patterns in the structure of the LookupPattern. An example LookupPattern initialization for the HTML structure given in the following code snippet is demonstrated in Fig. 3.

```
//Pattern definition is used in extracting text. The
    parent pattern will not be produced in the output.
LookupPattern mainPattern = new
    LookupPattern(LookupOptions.SKIP,
    LookupOptions.CONTAINER, "<div
    id=\"mainbar\"(.*?)>","</div>")
//Search the HTML by using child div patterns
.setStartEndMarker("<div","</div>")
//Add a child pattern
.addPattern(new LookupPattern(LookupOptions.TEXT,
    LookupOptions.ARTICLETITLE,"<h3(.*?)>","</h3>")
//Clean the h3 content by removing all tags
.setRemoveTags(true));
```

In the preceding code snippet, the structure defined by the sub-elements is employed to extract the relevant content. The nested elements are analyzed for matching instances in an identical sequence and under the same parent–child relationship. The resulting extraction output is presented in the form of an XML file as depicted below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<RESULT TYPE="TEXT" LABEL="ARTICLETITLE">
Select intervals intersecting a given interval in
    O(log(N)) in SQLite
</RESULT>
<RESULT TYPE="TEXT" LABEL="ARTICLETITLE">
How to update current branch with changes from master
    using TFS?
</RESULT>
<RESULT TYPE="TEXT" LABEL="ARTICLETITLE">
IIS Express delivers old, expired SSL Cert
</RESULT>
<RESULT TYPE="TEXT" LABEL="ARTICLETITLE">
The class List doesnt have an unnamed constructor
</RESULT>
</ROOT>
```

Table 1
Comparison with a similar DOM based scrapping.

Web domain	Comparisons	WebCollectives	DOM Approach
Hurriyet	Efficiency	1.353	59.771
	Lines of code	19	27
DWNews	Efficiency	0.822	51.110
	Lines of code	9	16
Euronews	Efficiency	1.338	55.787
	Lines of code	14	28
Webrazzi	Efficiency	0.987	52.172
	Lines of code	7	24
BBCTurkish	Efficiency	0.755	51.789
	Lines of code	5	20
Engadet	Efficiency	1.197	58.733
	Lines of code	11	24
Boingboing	Efficiency	1.292	61.224
	Lines of code	13	28
Gizmodo	Efficiency	1.194	60.788
	Lines of code	15	24
HuffingtonPost	Efficiency	1.181	55.174
	Lines of code	10	28
Stackoverflow	Efficiency	0.827	60.214
	Lines of code	5	8

3. Evaluations

In the evaluations, the parsing and XML extraction is compared against a similar experimental DOM approach which is constructed by the Jsoup library [8]. A similar recursive definition approach over DOM elements is applied in the compared model. The comparisons are made based on the lines of code and the extraction efficiency for 10 web domains and 100 web pages for each domain. The results are given in Table 1. The efficiency scores are given in terms of milliseconds.[1]

WebCollectives and the DOM based approach is compared based on the efficiency and lines of code for web content extraction. The functionality of the compared DOM approach is very limited and only include the necessary steps of the extraction. WebCollectives includes an imperatively defined structure for the extraction where each pattern must follow either a parent or sibling rule. On the other hand DOM approach enforce such structural definitions based on label relations. Thus, the DOM must define parent and sibling relations explicitly. Because of this definition model in DOM approach, each pattern definition corresponds to 4 lines of code. The following code snippet shows the compared definition model.

```
//Create the hierarchy
DOMHierarchy text = new
    DOMHierarchy(LookupOptions.ARTICLETEXT);
//Create the pattern
text.add(DOMLabel.create(LookupOptions.ARTICLETEXT,
    "div", "class", "news-content(.*?)"));
//Relate with the hierarchy
article.addChild(text);
//Specify the output labels
domModel.structure(DOMModel.ROOT,
    LookupOptions.ARTICLETEXT);
```

The efficiency of WebCollectives is much higher than the DOM model. The efficiency of the proposed approach is dependent on the number of recursive patterns. Increased pattern size increases the extraction time logarithmically. On the other hand, DOM approach has a significant parsing time which is mainly affected by the number

[1] Experiments are conducted on Xeon E5-2670 v3 processor with single threaded settings.

of HTML elements inside the web page. The results indicate that the proposed regular expression based extraction is faster than the DOM approach in all compared domains.

4. Impact

There are various approaches for crawling and extracting the content from web pages [9]. These can be categorized based on the crawling methodology, and extraction techniques. Crawling can be done through mirroring the web site, searching inside the web site or using a focus based crawling approach for traversing certain links [1, 10]. On the other hand, extraction can be done through querying a DOM model [11], using a supervised machine learning approach [12] or semi-automated rules [4]. There are some scrapping applications that use DOM in the extraction process [13]. But recent approaches of web extraction use scrapping techniques without relying on DOM models [14]. Scrapping techniques obtain only the queried information through pattern matching. Scrapping have been successfully applied in obtaining prices from web [15], aggregating the news [16], extracting images from web [17] and phishing web page detection [18].

Web scrapping approaches not always rely on DOM parsing [4,17]. DOM parsing is a time consuming process when the web page contains many HTML elements. These approaches use text search methods for scrapping relevant content more efficiently [19]. However, current approaches do not apply rules for hierarchical template based text search. WebCollectives combines the advantages of Web scrapping and hierarchical parsing. It efficiently extracts the relevant content.

There are several software tools for both crawling and extraction [6, 20]. One of the drawbacks of these softwares is that they either require exact information about the target HTML elements or require a training dataset of the target domain for extraction through machine learning inference [21]. In these software applications, the task of selection of the exact target element or training a classifier is easy for updating the extraction software. However, when there are small changes in the web domain, the element selection or training must be repeated. In WebCollectives, the regular expression based patterns integrated inside tree structure give a template matching ability and prevent failure in small changes, thus they are more accurate in extraction than other rule based approaches.

5. Conclusions

Non-DOM-based hierarchical parsing offers several advantages. It effectively captures the hierarchical arrangement of the extraction space as defined within the HTML content, making it amenable to generalization based on the relative positions of sub-elements. Furthermore, its capacity for matching through regular expressions allows it to accommodate a diverse range of HTML elements, enhancing resilience to template modifications. In addition, it demonstrates superior speed compared to DOM-based approaches and can be applied to a variety of content types, including XML, JSON, programming languages, and textual documents.

A limitation of template-based matching and similar web scraping techniques is their restricted adaptability to alterations in web page structures. Whenever there are changes in the layout of a web page, the corresponding rules need to be revised accordingly. While WebCollectives offers a degree of generalization through techniques such as relative element positioning and generic regular expressions, complete transformations of the web template necessitate rule adjustments. Machine learning methods have been developed to forecast the location of relevant content amidst such alterations. These methods rely on domain-independent training data manually curated by human experts. It is worth noting that WebCollectives does not inherently possess this capability, but it can be seamlessly integrated as a component within the pipeline of these machine learning approaches.

Currently, WebCollectives is primarily employed in text mining applications for the acquisition of textual data. It has been used in three distinct research articles. These are about authorship attribution [22, 23], and regular expression inference [24]. Currently, WebCollectives is used to collect sentences from web domains for training a neural language model.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my code.

Acknowledgments

I want to thank to my Ph.D. supervisor Prof. Dr. Özgür Yılmazel, and my former colleague Prof. Dr. Erdinç Uzun for their motivations in this field.

References

[1] Khder MA. Web scraping or web crawling: State of art, techniques, approaches and application. Int J Adv Soft Comput Appl 2021;13(3).
[2] Kumar M, Bhatia R, Rattan D. A survey of web crawlers for information retrieval. Wiley Interdisc Rev: Data Min Knowl Discov 2017;7(6):e1218.
[3] Kohlschütter C, Fankhauser P, Nejdl W. Boilerplate detection using shallow text features. In: Proceedings of the third ACM international conference on web search and data mining. 2010, p. 441–50.
[4] Uzun E. A novel web scraping approach using the additional information obtained from web pages. IEEE Access 2020;8:61726–40. http://dx.doi.org/10.1109/ACCESS.2020.2984503.
[5] Uzun E, Serdar Güner E, Kılıçaslan Y, Yerlikaya T, Agun HV. An effective and efficient web content extractor for optimizing the crawling process. Softw - Pract Exp 2014;44(10):1181–99.
[6] Diouf R, Sarr EN, Sall O, Birregah B, Bousso M, Mbaye SN. Web scraping: State-of-the-art and areas of application. In: 2019 IEEE international conference on big data. IEEE; 2019, p. 6040–2.
[7] Glez-Peña D, Lourenço A, López-Fernández H, Reboiro-Jato M, Fdez-Riverola F. Web scraping technologies in an API world. Brief Bioinform 2013;15(5):788–97. http://dx.doi.org/10.1093/bib/bbt026, arXiv:https://academic.oup.com/bib/article-pdf/15/5/788/17488715/bbt026.pdf.
[8] Wang J, Yang S, Wang Y, Han C. The crawling and analysis of agricultural products big data based on jsoup. In: 2015 12th international conference on fuzzy systems and knowledge discovery. IEEE; 2015, p. 1197–202.
[9] Vogels T, Ganea OE, Eickhoff C. Web2Text: Deep structured boilerplate removal. Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol.10772 LNCS, Springer Verlag; 2018, p. 167–79. http://dx.doi.org/10.1007/978-3-319-76941-7_13.
[10] Gupta A, Anand P. Focused web crawlers and its approaches. In: 2015 1st international conference on futuristic trends in computational analysis and knowledge management. IEEE; 2015, p. 619–22. http://dx.doi.org/10.1109/ABLAZE.2015.7154936.
[11] Liu Q, Shao M, Wu L, Zhao G, Fan G, Li J. Main content extraction from web pages based on node characteristics. J Comput Sci Eng 2017;11(2):39–48.
[12] Velloso RP, Dorneles CF. Web page structured content detection using supervised machine learning. Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol.11496, Springer Verlag; 2019, p. 3–18. http://dx.doi.org/10.1007/978-3-030-19274-7, https://link.springer.com/chapter/10.1007/978-3-030-19274-7_1.
[13] Ruchitaa Raj N R, Nandhakumar Raj S, Vijayalakshmi M. Web scrapping tools and techniques: A brief survey. In: 2023 4th international conference on innovative trends in information technology. 2023, p. 1–4. http://dx.doi.org/10.1109/ICITIIT57246.2023.10068666.
[14] Zhao B. Web scraping. Encycl Big Data 2017;1–3.
[15] Bricongne J-C, Meunier B, Pouget S. Web-scraping housing prices in real-time: The Covid-19 crisis in the UK. J Hous Econ 2023;59:101906. http://dx.doi.org/10.1016/j.jhe.2022.101906, COVID-19's Impacts on Housing Markets, URL https://www.sciencedirect.com/science/article/pii/S105113772200078X.
[16] Bhujbal MM, Bibawanekar MB, Deshmukh DP. News aggregation using web scraping news portals. 2023, p. 2581–9429, ISSN (Online).

[17] Uzun E. Scraping relevant images from web pages without download. ACM Trans Web 2023. http://dx.doi.org/10.1145/3616849.
[18] Boyapati M, Aygun R. Phishing web page detection using web scraping. In: Southeastcon 2023. IEEE; 2023, p. 167–74.
[19] Bale AS, Ghorpade N, S R, Kamalesh S, R R, S RB. Web scraping approaches and their performance on modern websites. In: 2022 3rd international conference on electronics and sustainable communication systems. 2022, p. 956–9. http://dx.doi.org/10.1109/ICESC54411.2022.9885689.
[20] Lopez LA, Duerr R, Khalsa SJS. Optimizing apache nutch for domain specific crawling at large scale. In: 2015 IEEE international conference on big data. IEEE; 2015, p. 1967–71.
[21] Aslam N, Tahir B, Shafiq HM, Mehmood MA. Web-AM: An efficient boilerplate removal algorithm for web articles. In: 2019 international conference on frontiers of information technology. IEEE; 2019, p. 287–2875.
[22] Agun HV, Yilmazel O. Incorporating topic information in a global feature selection schema for authorship attribution. IEEE Access 2019;7:98522–9.
[23] Agun HV, Yilmazel O. Bucketed common vector scaling for authorship attribution in heterogeneous web collections: A scaling approach for authorship attribution. J Inf Sci 2020;46(5):683–95.
[24] Agun HV, Uzun E. An efficient regular expression inference approach for relevant image extraction. Appl Soft Comput 2023;110030.