IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 3, 2020, accepted March 25, 2020, date of publication March 31, 2020, date of current version April 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2984503

A Novel Web Scraping Approach Using the Additional Information Obtained From Web Pages

ERDINÇ UZUN
Department of Computer Engineering, Çorlu Faculty of Engineering, Tekirda Namık Kemal University, 59860 Tekirda , Turkey
e-mail: erdincuzun@nku.edu.tr

ABSTRACT Web scraping is a process of extracting valuable and interesting text information from web pages. Most of the current studies targeting this task are mostly about automated web data extraction. In the extraction process, these studies first create a DOM tree and then access the necessary data through this tree. The construction process of this tree increases the time cost depending on the data structure of the DOM Tree. In the current web scraping literature, it is observed that time efficiency is ignored. This study proposes a novel approach, namely UzunExt, which extracts content quickly using the string methods and additional information without creating a DOM Tree. The string methods consist of the following consecutive steps: searching for a given pattern, then calculating the number of closing HTML elements for this pattern, and finally extracting content for the pattern. In the crawling process, our approach collects the additional information, including the starting position for enhancing the searching process, the number of inner tag for improving the extraction process, and tag repetition for terminating the extraction process. The string methods of this novel approach are about 60 times faster than extracting with the DOM-based method. Moreover, using these additional information improves extraction time by 2.35 times compared to using only the string methods. Furthermore, this approach can easily be adapted to other DOM-based studies/parsers in this task to enhance their time efficiencies.

INDEX TERMS Computational efficiency, algorithm design and analysis, web crawling and scraping, document object model.

I. INTRODUCTION

A web page contains unnecessary content such as menus, advertisements, banners, footers, sitemaps and necessary content such as title, summary, main text, price, and description in terms of information that users need. With the increase of unnecessary content on the web pages, it has become essential to eliminate unnecessary content and to extract necessary content that can be used by the text processing applications such as search engines, question-answering systems, recommendation systems, trend detection/monitoring, sentiment analysis, and e-commerce market monitoring. Many studies [1] in this task focus on determining the extraction of the data/pattern automatically. However, the time efficiency of this process is not taken into consideration in these studies. Therefore, this study deals with the acceleration of the extraction process rather than deciding the extraction pattern. And this study introduces a novel approach, namely UzunExt, which accomplishes the extraction process in a shorter time and uses fewer resources. Moreover, how this approach can be adapted to other studies is explained thoroughly.

Web pages are made of HTML elements and data between these elements. Web scraping is a process of extracting specific data between these elements for providing data for other applications such as online price change monitoring, product review analyzing, weather data monitoring, tracking online presence, gathering articles and so on. The internet is a rich source of "big data" for these applications. Most of the studies [2] in "big data" concentrate on the time efficiency of deep learning models. However, to increase the time efficiency of obtaining data is an important issue. The UzunExt approach consists of two main components: crawling web pages and extracting data from them. It uses the additional information obtained from web pages during the crawling process for increasing the extraction time efficiency.

Most of the academic studies described in Section II for this task are based on creating a DOM tree that is a tree-based hierarchy for representing a web document.

The associate editor coordinating the review of this manuscript and approving it for publication was Dongxiao Yu.

Moreover, HTML parsers that can be used for web scraping are also based on the DOM tree. The following list presents several parsers for four different programming languages.

- .Net: HAP, AngleSharp, Microsoft HtmlDocument
- Java: jsoup, Lagarto, HtmlCleaner
- Python: HTML Parser of The Standard Library, html5lib, html5-parser, lxml, AdvancedHTMLParser
- Node.js: Cheerio, Jsdom, Htmlparser2, Parse5

In the DOM tree construction, all HTML elements in a web document are taken into consideration. However, constructing a DOM Tree is not convenient for efficient web scraping. A few studies, which do not require creating a DOM Tree, have been proposed in the literature. Uzun *et al.* [3], [4] and Uçar *et al.* [5] eliminate the construction of the DOM tree in their algorithms. Wu [6] divides a news page into text fragments, prepares a set of content features and layout features from these fragments and finally uses a Bayesian algorithm to find appropriate content for online newspapers. All approaches described above make a generalization for a web document. These studies are mostly about automatically extracting the desired data from web pages and generating an extraction pattern for this data rather than making an effective extraction. This study is based on developing effective web scraper. Three additional information obtained from the extraction process of web pages is used to improve the following extraction processes. Additionally, this approach can also be adapted to the methods of previous studies as a module.

Different HTML parsers in programming languages can be used for creating a DOM tree and extracting content from this tree. Uzun *et al.* [7] try three different well-known parsers in.Net. In our study, we use these parsers for understanding the time consumption of DOM creation and extraction of content. However, our expectation is that a simple text processing on the required elements can speed up the extraction process. The most important issue in text processing is the string matching step of an extraction algorithm. Bulu *et al.* [8][1] compared 31 different matching algorithms for searching HTML elements in web pages. Their experiments show that Skip Search algorithm is the best algorithm in this task. Our study takes the best search results of these algorithms into account and additionally uses the IndexOf method in string Class of the.Net. One of the additional information of our approach is to predict the starting position information of the searched element on a web page of a website for accelerating the extraction process. Moreover, Grubbs' test [9] is utilized to eliminate the invalid position information obtained from web pages for a website. The second additional information is the number of inner tag in the desired element for finishing the extraction process in a shorter time. Finally, the last additional information is the repetition of the desired HTML element in a web document. Taking this repetition into account, a part of the document instead of the whole document can be used for terminating the extraction process.

Web pages are collected via a web crawler that discovers and retrieves all website pages automatically. Some information obtained from the web pages can be used for improving the crawling process [10], [11]. A focused crawler downloads web pages that are relevant by analyzing the data on the pages [12]. There are many studies on this task that uses information in web pages to improve this task [13]–[17]. Web pages of a website contain similar elements and have a similar hierarchy. Uzun *et al.* [4] improve the crawling process by predicting these elements. In our study, the extraction process is enhanced by analyzing three additional information obtained from previous web pages of a website.

The rest of the study is organized as follows. The second section gives information about the literature on web scraping. The third section introduces the details of the problem. The fourth section presents web content extraction methods using DOM Tree. The fifth section covers the novel approach used for this task. The sixth section is dedicated to the experiments for comparing our approach with other methods and assessing the effectiveness of the algorithm. The last section provides our conclusion and future studies.

II. STUDIES IN WEB SCAPING

The most studied issue for web scraping is to determine the extraction patterns automatically. On the other hand, these patterns are created manually by developers/experts in the implementation of web scrapping tools that are used HTML parsers. Determining the extraction pattern manually is explained in detail in Section III.

Automated techniques can be grouped into supervised, unsupervised and hybrid approaches. In a supervised approach, extraction patterns are learned from labeled data. Early studies for this approach are mainly based on wrapper induction methods [18]–[21]. Recent studies are about machine learning techniques [22]–[25] that focus on predictions based on a learning model obtained from the training dataset. While researchers in this field try to experiment with different machine learning algorithms and to obtain new features for improving their learning models.

An unsupervised approach automatically finds extraction patterns from single or multiple web pages and eliminates the manual labelling task. This approach can be classified into three groups: statistical-based, template detection, and assumption techniques. Statistical-based techniques utilize statistical information including content ratio [26], tag ratio [27], link density [28], [29], quantity of parent nodes [30], quantity of HTML/word node tokens [31], link/text percentage of nodes [32], ratio of non-hyperlink character number to character number that hyperlinks contain [33] and ratio of words/leafs [34]. Template detection techniques [35]–[38] find the main content blocks in all blocks. A template is a master web page or shell document that is used as a basis for creating new web pages. Blocks are parts of a template including main content blocks, menus,

[1] All algorithms are open-source and available via the web page. https://github.com/erdincuzun/SMA.NET.

advertisements, banners, footers and so on. Assumption techniques [39]–[47] use the structure of the web document, content nodes, main content block, some elements, style information, etc. to extract relevant content.

A hybrid approach tries to exploit the advantages of both approaches. Uzun *et al.* [3], [4] propose a hybrid approach that has two steps: the first step uses a DOM (Document Object Model) tree for learning extraction patterns and the second step eliminates DOM Tree creation for efficient extraction. The first step has three procedures including creating a DOM Tree, preparing features and predicting appropriate elements with an appropriate machine learning method. The second step uses a simple string matching algorithm to extract content by employing an extraction pattern that can be used on other web pages of the related website. This second step eliminates the first step and gives time efficiency to the algorithm. These two studies are convenient for extracting online newspapers, one-page blogs, and programming solution/example descriptions. Uçar *et al.* [5] developed a similar algorithm for extracting review parts of a web document. The second step of this study is utilized as one of the baselines in our experiments.

There are many studies in determining the extraction pattern automatically. Besides, developers/experts prepare this pattern manually in web scraping tools. All of the techniques, approaches, and tools are DOM-based and ignore time efficiency in the extraction process. The main contribution of this study is to increase the time efficiency of the extraction process. Moreover, our approach collects three additional information obtained from web pages of a web site and improves time efficiency in the extraction process.

III. PROBLEM DEFINITION

A web browser renders a display from a web document. In the background of this rendering process, there are three additional operations including construction of a DOM Tree, processing of CSS (Cascading Style Sheets) and execution of JavaScript. Fig. 1 shows a simple web page and the DOM Tree constructed from a document.

A web document, D, contains characters that is CH = $\{ch_1, ch_2, \ldots, ch_n\}$. HTML parsers find HTML elements, E = $\{e_1, e_2, \ldots, e_n\}$ from C. An HTML element is typically made up of two tags: an opening tag (OT) and closing tag (CT). Content, C, is written between these tags. C may also contain inner HTML elements, Ee. All tags, T = $\{t_1, t_2, \ldots, t_n\}$, are determined by W3C (World Wide Web Consortium) that is an international community responsible for developing the Web standards. The web page has tags such as <html>, <head>, <title>, <body>, <div>, <h1>, <h2>, and <p> that are given in Fig. 1. Moreover, many different tags can be utilized in web pages. The opening tag of an element (OT_e) may have attributes and their values that are defined as A = $\{a_1 = v_1, a_2 = v_2, \ldots, a_n = v_n\}$. Attributes such as id, class, and itemprop that are shown in Fig. 1 help to provide extra information about elements. This extra information can be used for formatting the desired elements in a web document. Some start tags such as <html>, <head>, <body> and <title> in Fig. 1 may not contain any attributes that are represented as A = { }. On the other hand, a tag may contain one or more attributes like <div> tag in Fig. 1. For example, the first <div> tag has the attribute that contains id as the attribute name and "bigcontainer" as attribute value that is case-insensitive and separated by an equal sign. As a result, an HTML element can be expressed as follows:

$$e = (OT_e(t_e, A_e), (C_e, E_e), CT_e) \tag{1}$$

A DOM tree is utilized to represent HTML elements and their hierarchies. In web scraping, OT_e can be used for preparing an extraction pattern to find C by searching on D. The desired elements are determined by an expert programmer as needed. For example, the following opening tags can be employed to eliminate the unnecessary content and to extract necessary content in Fig. 1.

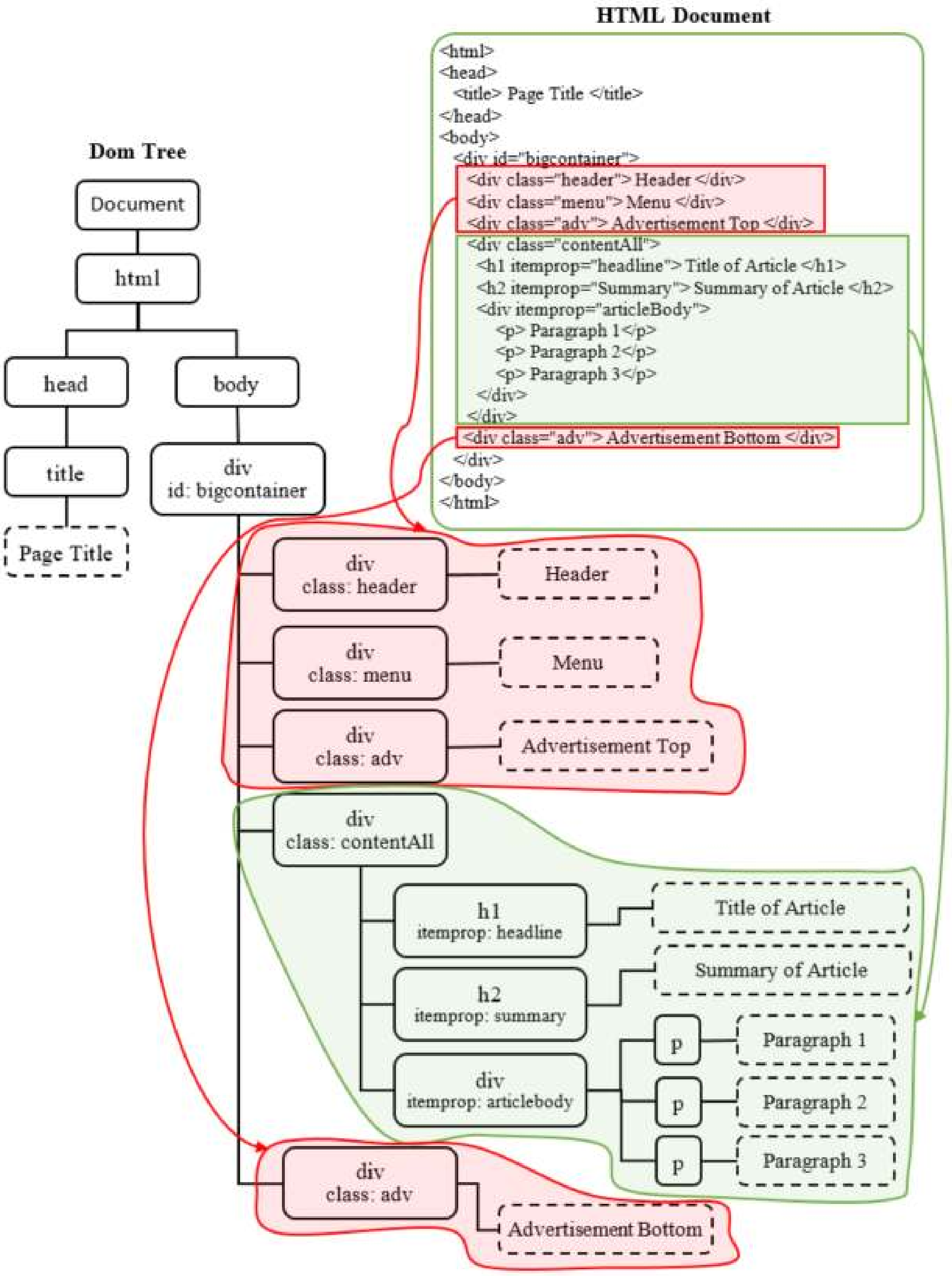
- <div class = "contentAll">: All necessary content
- <h1 itemprop = "headline">: Title of the web page
- <h2 itemprop = "Summary">: Summary of the web page
- <div itemprop = "articleBody">: Main text of the web page
- <p>: Inner texts of the web page

A web developer can prepare an extraction pattern by using the tag of an element (t_e) and specifying desired attributes (A_e) in OT_e. Moreover, this pattern is used to produce different web pages for a website. Therefore, when required patterns are resolved for a website, the extraction process can be easily applied by using resolved elements. A software developer may choose one or more than one of these extraction patterns for her / his purpose. For instance, if a developer is developing a search engine project, instead of storing the entire HTML document, a developer can store the text in a particular element such as <div class = "contentAll">. On the other hand, if there is not enough storage space for this task, you can store only texts of two elements including <h1 itemprop = "headline"> and <h2 itemprop = "Summary">.

IV. EXTRACTION METHODS

Extraction methods in web scraping can be classified as DOM-based and string-based methods. DOM-based methods are based on creating a DOM-tree for a web page and searching a specific element in this tree. String-based methods are focused on directly extracting content from CH.

In a web browser, a web developer can access, add, alter and delete all elements in a web page by using methods of JavaScript-based on the DOM tree. Parsers of programming languages in this task are usually built on this structure. For example, these parsers have similar methods that create a DOM tree and access the elements of this tree. Extraction methods mostly used to perform web scraping by traversing the DOM tree.



HTML Document

```
<html>
<head>
  <title> Page Title </title>
</head>
<body>
  <div id="bigcontainer">
    <div class="header"> Header </div>
    <div class="menu"> Menu </div>
    <div class="adv"> Advertisement Top </div>
    <div class="contentAll">
      <h1 itemprop="headline"> Title of Article </h1>
      <h2 itemprop="Summary"> Summary of Article </h2>
      <div itemprop="articleBody">
        <p> Paragraph 1</p>
        <p> Paragraph 2</p>
        <p> Paragraph 3</p>
      </div>
    </div>
    <div class="adv"> Advertisement Bottom </div>
  </div>
</body>
</html>
```

FIGURE 1. An example of an HTML document and Dom Tree of this document.

In JavaScript, there are some methods including getElementsByTagName, getElementByID, getElementsByClassName, and getElementsByName for traversing a DOM tree and extracting content (C_e) using a tag (t_e) or attribute of id, class, and name (A_e) in the OTe, respectively. However, these methods are not appropriate for accessing attribute of itemprop. querySelector() method returns the first element that matches a specified CSS selector(s) in an HTML document. For example, querySelector("[itemprop = headline]") method can be used to access the element of <h1 itemprop = "headline">. In this study, MS_HtmlDocument in.Net has similar to methods of JavaScript therefore it is selected for comparison with other methods. Although such methods are available for traversing the DOM tree, the W3C, which sets official Web standards, has introduced new standard XPath for navigating through elements and attributes in the DOM tree. XPath uses path expressions to access an element. These path expressions look very much like the path expressions in computer file systems. For selecting <h1 itemprop = "headline"> element, XPath expression is '/html/body/div/div/h1[@itemprop = "headline"]'. XPath expressions can also be utilized in JavaScript, Java, Python,

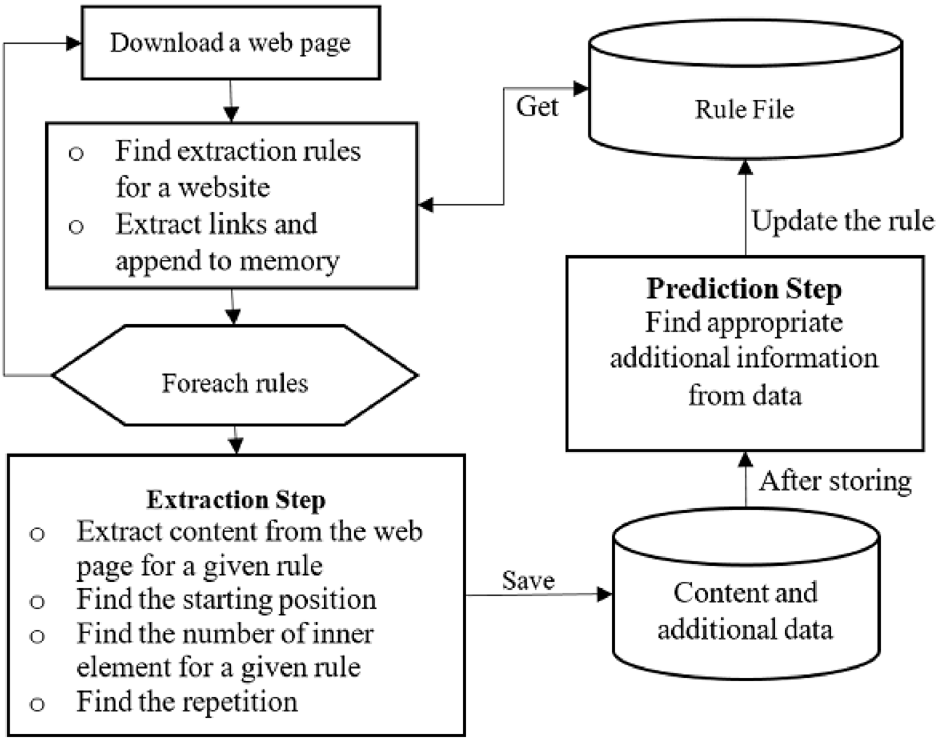

Download a web page
o Find extraction rules for a website
o Extract links and append to memory
Foreach rules
Extraction Step
o Extract content from the web page for a given rule
o Find the starting position
o Find the number of inner element for a given rule
o Find the repetition
Get
Rule File
Update the rule
Prediction Step
Find appropriate additional information from data
After storing
Save
Content and additional data

FIGURE 2. **The flowchart of the UzunExt.**

Ruby, C# and many other languages. In this study, the .Net parser HAP[2] is preferred to implement XPath expressions.

Some programming languages have general query facilities for applying to all sources of information. In .Net, LINQ (Language-Integrated Query) supports these facilities. In this study, the AngleSharp parser[3] that allows using of LINQ is selected.

Three methods described above are based on the DOM tree that is a convenient structure for manipulating all elements. However, an expert programmer only needs a certain number of elements for web scraping. In this situation, regular expressions, the well-known text processing technique, can be considered as a solution for obtaining the content of certain elements. For example, the following expressions can be used for extraction:

- <h1.itemprop =.headline.>(.*?)</h1>
- <h2.itemprop =.Summary.>(.*?)</h2>
- <p>(.*?)</p>

The form of <h1>, <h2> and <p> elements are suitable for a regular language that is formal language [48]. A regular language that can be expressed using a regular expression. However, the form of <div> element is not appropriate for using regular expressions because this element can be represented in a nested form. That is, the form of <div> element is context-free because the number of inner <div> elements is not known. If the number of inner <div> elements are known, the problem can be expressed with a regular expression. Moreover, the time complexity can be improved because the regular languages can be recognized by a final state machine of m-states where m is a finite number. This means it requires O(m) space and O(n) time, where n is the length of the data. On the other hand, the context-free languages can be recognized by a non-deterministic pushdown automaton-like CYK algorithm [49]. The time complexity of this algorithm is $O(n^3 * G)$, where n is the length of the data and the number of rules. This study aims to try to improve the time efficiency of web scraping by converting it to a regular language level.

There are few studies that have attempted to improve the time efficiency of the web scraping process. Uzun *et al.* [50] propose a simple parser, namely SET Parser, using basic string matching for efficient extraction. This parser has been used by some studies [3]–[5] for eliminating DOM tree construction. This study introduces a much more efficient approach using the additional information obtained from a crawling process.

V. APPROACH OVERVIEW

Our approach, namely UzunExt, is two-fold: extracting content from a web page and predicting suitable values of the stored additional data. Fig. 2 presents the flowchart of the approach. According to Fig. 2, after a web page is downloaded, extraction patterns are obtained from the rule file for

[2] HAP (HTML Agility Pack) is an agile HTML parser that builds a DOM tree and allows to parse this DOM Tree. For more information http://html-agility-pack.net/.

[3] The ultimate angle brackets parser library (AngleSharp) constructs a DOM tree based on the official W3C specifications. For more information https://github.com/AngleSharp/AngleSharp.

TABLE 1. An example extraction pattern file holding the OT$_e$, type of content and additional information.

			Additional information		
Website	OT$_e$	Type	Starting Position	The number of Inner	Tag Repetition
Acsh	<h1 class="page-header">	Title	15576	1	False
Acsh	<div class="content">	Main Text	18356	-1	False
Adidas	<div class="p-title">	Title	26487	1	False
Adidas	<div class="p-price">	Price	26731	1	False
Adidas	<div class="product-features">	Desc.	36428	-1	False

a website and hyperlinks are parsed from this page. The rule file holds rules as given Table 1 for each website. A hyperlink starts with an anchor opening tag <a, and includes a hyperlink reference href = "URL for the page". The following regular expression <a.href..(.*?).>.*?</a> can easily be used to extract all hyperlinks in a web page.

In Table 1, the OT$_e$ and type of content are determined by an expert web programmer. The OT$_e$ is a pattern that is utilized for searching in the downloaded web page. The additional information is estimated during a crawling process for speeding up search and extraction operations. A conventional search algorithm starts at the beginning of the document. However, the search algorithm in our approach uses the starting position information that means the initial index. The number of inner tag information holds the number of tag occurrences for the content of the element. A value of −1 in the number of inner tag information indicates that the number of tag occurrences in the element is not fixed. If this value is fixed, this situation reduces the duration time of the extraction process because of time complexity. The repetition information is used for breaking the extraction process after the first extraction. In this case, the rest of the document is not checked.

After obtaining rules, the extraction step is applied to each rule for a web page. This process uses additional information loaded from the rule file. It extracts the content of a web page and finds the starting position, the number of inner element and the repetition as data. The content of a web page and additional data stores into the storage. After storing the process, data is evaluated in the prediction step for finding values of the starting position, the number of inner element and the repetition. Finally, these values are saved to the rule file as additional information.

The rest of this section gives more information about the approach. Firstly, the extraction step is described through an algorithm. The second subsection covers the prediction step of the additional information. Thirdly, the outputs of these steps are examined by an example. The second subsection is about searching algorithms that can be used in the extraction step. The final subsection describes how can easily be adapted to the algorithms of other studies to improve their time efficiencies.

Algorithm 1 Initialization of the Extraction Step

```
   Data: OTe: searching pattern, source(CH): the
         source of a web page, startingPosition,
         innerTagCount, repetition, sa: a search
         algorithm
   Result: List: a string list that holds the result of
           extraction, data_startingPosition: holds the
           first index value of searching
1  start_tn = resolveStartTag(OTe);
2  end_tn = resolveEndTag(OTe);
3  start_index = Search(OTe, source, startingPosition,
    sa);
4  if start_index == −1 then
5      start_index = Search(OTe, source, 0, sa);
6  end
7  data_startingPosition = start_index;
8  if start_index == −1 then
9      end_index = Search(end_tn, source, start_index,
        sa) - start_index+ end_tn.Length;
10     start_tag = 0;
11     if innerTagCount > −1 then
12         start_tag = innerTagCount;
13     end
14     end_tag = 0;
15     result_temp = "";
16     call Algorithm 2;
17 end
18 return List, data_startingPosition
```

A. EXTRACTION STEP

The main algorithm of the UzunExt extracts web content from web pages by using string methods. It has six parameters as input, including the opening tag (pattern - OT$_e$), source, starting position (startingPosition), number of nested tag name (innerTagCount), repetition of tag name (repetition) and a search algorithm (sa). The OT$_e$ determines the pattern to be searched in the source text (CH). The starting position specifies an index that is the starting position to search in the CH. The number of nested tag name parameter defines the formal language whether is a regular language or context-free language. When the nested parameter value is "−1", it means that content-free language is selected. On the other hand, if this parameter is greater than zero, it means regular language. The repetition of an element name is Boolean value to determine whether the element name has one item or more than one item in the source. The starting position, number of nested tag name and repetition parameters are taken from the rule file. The search algorithm allows the use of different search algorithms. The initialization of the Extraction step is given in Algorithm 1.

In Algorithm 1, the tag name (start_tn) and the closing tag name (end_tn) are parsed for the OTe (Line 01, 02). For instance, when the OT$_e$ is <div class = "contentAll">, the start_tn is <div and the end_tn is </div>. The Search

Algorithm 2 The Extraction Step of UzunExt

```
1  while True do
2      sub_temp = source.Substring(start_index,
       end_index);
3      result_temp + = sub_temp;
4      if innerTagCount == −1 then
5          start_tag + = Count(start_tn, sub_temp, sa);
6      end
7      end_tag++;
8      if start_tag ! = end_tag then
9          start_index+ = end_index;
10         end_index = Search(end_tn, source,
           start_index, sa) - start_index +
           end_tn.Length;
11     else
12         add result_temp to List;
13         if repetition == false then
14             break;
15         end
16         start_index = Search(OTe, source,
           start_index+ end_index, sa);
17         if start_index == −1 then
18             break;
19         end
20         end_index = Search(end_tn, source,
           start_index, sa) - start_index +
           end_tn.Length;
21         result_temp = "";
22         if innerTagCount > −1 then
23             start_tag = innerTagCount;
24         else
25             start_tag = 0;
26         end
27         end_tag = 0;
28     end
29 end
```

method returns the index of the first occurrence (start_index) of the OT_e in a source (Line 3). If the appropriate value for the initial index (startingPosition) is known, the number of comparisons in the loop of the search algorithm is reduced, thus making the search more efficient. Values of data_startingPosition obtained from web pages are stored to be used in order to the prediction step of our approach. However, if the startingPosition is not selected correctly, the algorithm continues searching until the end of the CH, and the OT_e cannot be found. Although this situation is not desirable, the algorithm also searches for the text before the starting position (Line 5). The search method is utilized to find both the start_index and the end_index. After the values are found in the initial process, the extraction step is started as given Algorithm 2.

In Algorithm 2, Substring method (Line 2) extracts the characters from the CH, between two specified indices (start_index, end_index) determined in Algorithm 1, and returns the new substring. This substring is appended to result_temp (Line 3). If the value of innerTagCount is −1, it means that innerTagCount is not known. In this case, the Count method (Line 5) calculates and returns the number of the start_tn for a given string (sub_temp), and the algorithm works like a context-free language. On the other side, this method is not needed if the algorithm knows the number of occurrences of the start_tn in a source. In this case, the algorithm works like regular language and the start_tag sets with a fixed value that is taken from innerTagCount. The element <div class = "contentAll"> in Figure 1 has two elements including itself and the element <div itemprop = "articleBody">. If the algorithm knows the innerTagCount as two, it does not need using the Count method. In other cases, the algorithm calculates the number of start_tn and updates the start_tag (Line 5).

The last additional information in Algorithm 2 is the repetition. If the value of repetition is False, the algorithm breaks the loop after the first extraction (Line 17). However, when a web page can have multiple content like the elements of <div class = "adv"> and <p> in Figure 1, the value of repetition must be True for obtaining all the results. Otherwise, the algorithm continues to search the other OT_e in the CH (Line 02). If the search result is "−1" (Line 17), the algorithm breaks the loop. If the search result is not −1, the end_index is found by the Search method (Line 20). Then, the variables are updated and the loop continues to extract the other content (goto Line 2).

Algorithm 1 returns values including a List that is a string list of extraction results and data_startingPosition that is the index of the first occurrence of OTe in the CH. Values of data_startingPosition are collected in order to utilize in the prediction step.

B. PREDICTION STEP

In this section, the prediction of the startingPosition, innerTagCount, and repetition are separately examined.

1) PREDICTION OF THE startingPosition

Each extraction pattern has own value for the startingPosition as given in Table 1. For finding this value, Algorithm 1 stores all extraction positions to the storage. After storing process, the minimum value of these values can be selected as the starting position, to speed up the search process. The value of starting position can be discovered while crawling web pages belonging to a website. Algorithm 3 indicates the prediction of the StartingPosition.

In Algorithm 3, the Min method returns the smallest value in a list. The Len method returns the length (the number of items) of a list. The UzunExt approach uses Grubbs' test to assess whether one of the values in the startingPosition_List is a significant outlier from the rest before returning the minimum value. However, at least three values are needed for this test to be applied (Line 02). There are two versions as maximum and minimum of this test. Our algorithm uses

```
Algorithm 3 Finding an Appropriate Value for the StartingPosition
   Data: List_startingPosition: all data obtained from
         Algorithm 1 for each rule
   Result: min: an appropriate value for using as the
           StartingPosition
1  min = Min(List_startingPosition);
2  if $Len(List\_startingPosition) >= 3$ then
3      mean = Calculate_Mean(List_startingPosition);
4      sd = Calculate_SD(List_startingPosition);
5      if $sd == 0$ then
6          return List_startingPosition[0];
7      min = List_startingPosition[0];
8      foreach value in List_startingPosition do
9          G = |mean - value| / sd;
10         if $G < CriticalZ[Len(List\_startingPosition)]$
           and $min > value$ then
11             min = value;
12 return min;
```

to test whether the minimum value is an outlier as one of the following one-sided tests:

$$G = \frac{\overline{Y} * Y_{min}}{s} \tag{2}$$

with $\overline{Y}$, s, and Y_{min} denoting the sample mean (Line 02), standard deviation (Line 03: sd), and the minimum value of the List_startingPosition, respectively. This test detects outliers from normal distributions. The value of G compares with the Grubbs' critical value table (Line 09: CriticalZ) that is used to determine whether the minimum value is the outliner. Thanks to Grubbs' test, the values furthest away from the mean is ignored in order to estimate the close value for the mean.

C. PREDICTION OF THE innerTagCount AND REPETITION

In the prediction step, the innerTagCount and repetition can be determined by examining at least two web pages of a website. For the opening tag <div id = "bigcontainer"> in Fig. 1, the number of inner tags is six. If these numbers are equal in two web pages, this value can be used as the innerTagCount. If the number of repetition is more than one for two web pages, it means that the value of the repetition is true. For the opening tag <div class = "adv"> in Fig. 1, the value of repetition is true. On the other hand, the value of repetition is false for elements <div id = "bigcontainer">, <div class = "header">, <div class = "menu">, <div class = "contentAll">, <h1 itemprop = "headline">, and so on.

Two web pages are enough for most websites, but some websites and some extraction patterns need more than two web pages. For example, the number of inner tag in the opening tag <div id = "bigcontainer"> is six in the first five pages, but this value may change on other web pages. Therefore, simple tests are done for every website in order to find out how many pages are enough for the prediction step. Simple heuristic tests show that six/seven web pages are enough for prediction of the innerTagCount and repetition. In the study, ten web pages are selected for the prediction step by considering the number of errors that may occur on other websites. For these two parameters, these values should be rechecked at a certain time because a web designer can make changes in the structural design of his/her website.

TABLE 2. The outputs of the algorithms of the approach with an example.

	The extraction step				The prediction step		
WPN	sP	iTC	r	SS	sP	iTC	r
1	22968	5	False		0	-1	True
2	18364	5	False	*	22968	-1	True
3	23516	5	False		18364	-1	True
4	17160	5	False	*	18364	-1	True
5	17901	5	False		17160	-1	True
6	24459	5	False		17160	-1	True
7	21957	5	False		17160	-1	True
8	23378	5	False		17160	-1	True
9	23649	5	False		17160	-1	True
10	20679	5	False		17160	-1	True
11	24301	5	False		17160	5	False
12	20859	5	False		17160	5	False
13	23814	5	False		17160	5	False
14	16543	5	False	*	17160	5	False
15	17961	5	False		16543	5	False

WPN: Web Page Number, sP: startingPosition
iTC: innerTagCount, r: repetition, SS: Second Search

D. OUTPUTS OF THE ALGORITHMS

In this section, the outputs returned from 15 web pages of an example website are examined to understand the approach better. Table 2 indicates the outputs obtained from steps of the extraction and prediction.

In Table 2, the algorithm of the extraction step takes the additional information of startingPosition, innerTagCount, and repetition from the prediction step. In the first web page, these information are 0, −1, and True, respectively. The value of 0 in startingPosition means that the search process starts from the beginning of the web page. The prediction step finds the most appropriate value for startingPosition by looking at the previous data obtained from the extraction step. If the value of the prediction step is not appropriate, the search process starts again at the beginning of the web page. This situation can be considered as a second search (SS) that occurred three times in Table 2.

The value of −1 in innerTagCount indicates that the number of the nested tag should be calculated. The value of True in repetition means that the extraction process continues until the end of the source of a web page. These values are calculated throughout the first ten web pages. After ten web pages, the prediction step checks the data obtained from the extraction step for determining the additional information on innerTagCount and repetition. The data of ITC in the first ten web pages are 5 and equal for all extraction processes. Therefore, ITC information sets as 5. On the other hand, the data of repetition for the first web pages is False and repetition information sets as False. The effect of each additional information is examined in the experiment section.

Algorithm 4 Discovering an Appropriate Element for Efficient Extraction

```
  Data: a web page: a html document, a specific element:
        the resulting element from any extraction
        algorithm
  Result: theElement: The opening tag
1 HTMLElements = CreateDOM(a web page);
2 theElement = HTMLElements.FindElement(a specific
  element);
3 while true do
4     if theElement = Body or theElement.Unique then
5         return theElement;
6     else
7         theElement = theElement. parentElement;
```

E. SEARCHING ALGORITHMS

One of the most crucial issues in web data extraction is the searching process. There are many string searching algorithms, sometimes called string matching algorithms, in the literature. Moreover, popular programming languages such as Java, C#, Python, Ruby, and JavaScript have indexOf methods that return the position of the first occurrence of a specified value in a string. Bulu *et al.* [8] try to find suitable pattern matching algorithms in web pages. In this study, seven of these algorithms and indexOf methods are used in the experiments. Moreover, the Count method in Algorithm 2 makes use of the Search method.

F. APPLYING THE UzunExt APPROACH TO OTHER STUDIES

According to the experiments, it is necessary to use a unique opening tag for efficient extraction. Moreover, this unique tag improves the time efficiency of the UzunExt approach. The methods applied to most of the other published studies may recommend for any tag in this task. However, there is no effective extraction over repetitive tags, and it can also cause errors. For example in Fig. 1, although the main text can be obtained via the <p> tag, the same text can be acquired through the <div class = "contentAll"> tag. Since the <p> tag can be repeated throughout a web page, the entire web page is scanned. That is, the UzunExt performs an extraction process for each <p> tag. Additionally, the <p> tag can also contain an unnecessary content of a web page. Therefore, Algorithm 4 that finds a unique opening tag for the appropriate extraction pattern is developed. Algorithm 4 contains pseudocode that can easily be adapted to previous studies in this task and allows for faster extraction of web content.

In Algorithm 4, a DOM tree including all the HTML elements and their hierarchy is created (Line 01). The location of a specific HTML element on the DOM tree is found via FindElement method (Line 02). If the element found is a <body> element or unique element, the algorithm does not need to traverse the other elements (Line 05). All elements which have the same attributes in the DOM tree are searched to understand whether the element is unique. Otherwise, the parent of the element is the new element for the loop (Line 07).

TABLE 3. Information about dataset.

Category	Number of Website	Average Web Page Size (KB)	Average Number of Characters	Average Number of Elements in the DOM tree
Article	15	71.89	73477.60	667.67
Dictionary	9	89.64	91318.11	720.11
Health	5	69.14	70768.00	470.00
Movie	10	108.29	110818.40	1058.90
Newspaper	35	131.64	131555.54	807.43
Shopping	15	289.53	291876.87	1686.80
Trip	11	397.43	404687.36	2853.55
	100	166.36	168202.09	1143.86

VI. EXPERIMENTS

In this section, firstly information about the dataset and extraction patterns prepared for this task is given. The second subsection covers the average extraction time of three DOM-based parsers and SET Parser. In the last section, the time efficiency of the UzunExt approach is examined from several different perspectives. All experiments are carried out on an Intel Core i5-3.2Ghz 8 GB RAM computer with Windows 10 operating system.

A. DATASET AND EXTRACTION PATTERNS

The approach presented in this study requires more than one web page per website. Unfortunately, since we do not have an appropriate dataset for our requests, a simple crawler has been developed to create this dataset. This crawler was used to download 30 web pages for 100 websites containing the text of different languages including English, Russian, German, French, Italian, Spanish, Turkish, Chinese, and Japanese. The datasets of previous studies focus only on a single category such as main content, comment content and repetitive content. This study contains web pages in many different categories including article, dictionary, health, movie, newspaper, shopping, and trip. After downloading 3000 web pages, 2-4 extraction patterns were determined for each website. The statistical results obtained from web pages and extraction patterns are given in Table 3 and Table 4, respectively.

In Table 3, websites are divided into seven different categories in order to examine the differences between websites. Table 3 indicates the correlation between the average web page sizes, the average number of characters in web pages and the average number of elements in the DOM tree for different categories in our dataset. The average size of web pages in the seven categories is different from each other. When the web pages in the categories of articles, dictionary, and health are examined in detail, it appears that these web pages contain fewer advertisements and hyperlinks than other web pages. Therefore, the average web page sizes of these categories are smaller than the other web pages. On the other hand, the web

TABLE 4. Information about extraction patterns.

Tag Names	Num. of the OT$_e$	Average number of characters	ID	Class	Other Attributes	Fixed Inner Elements		Repetitive	
						Yes	No	Yes	No
ASTFM	14	31.50	4	14	1	12	2	-	14
div	151	33.20	34	137	12	77	74	2	149
H	65	24.00	4	46	12	65	-	3	62
p, span, ul	17	28.35	1	16	2	12	5	2	15
	247	30.63	43	213	27	166	81	7	240

Similar tags grouped together.
Abbreviations:
ASTFM : <article>, <section>, <table>, <fieldset>, <main>
H : <h1>, <h2>, <h3>, <header>

pages in the categories of movie, newspaper, shopping, and trips contain more additional information, linearly. In the following subsections, the correlation between the effects on the average response time of the algorithms and the increase of the average file size, the number of characters, the number of elements is examined.

Table 4 gives the statistical information about 247 extraction patterns. These rules include 13 different tags and may include one or more attributes such as ID, class, and others. Some elements have a fixed number of inner tags in web pages of a website. However, 49% of <div> tag is not fixed. Determining the number of the inner tag can speed up the extraction process of the UzunExt approach. For some extraction patterns, after completing the first extraction, there is no need to extract the other extraction. That is, some extraction patterns are not repetitive for a web page. Only 3% of elements are repetitive in the prepared rules. In the following sections, the effect of these situations on the UzunExt is analyzed.

B. EXTRACTION WITH THE DOM-BASED ALGORITHMS AND SET PARSER

Most programming languages have several parsers for DOM-based operations. In this study, three different parsers including MS_HtmlDocument, AngleSharp, and HAP are used to create a DOM Tree and find a specific element on this tree. In the DOM tree, finding the specific element is enough to obtain the content. Moreover, SET Parser, which does not need to create a DOM tree, performs the extraction process using regular expressions and string methods. Table 5 indicates the average duration time obtained from the dataset and extraction patterns for these three parsers and SET Parser.

TABLE 5. Information about extraction patterns.

Category	MS_Html Document		AngleSharp		HAP		SET Parser	
	C (ms)	F (ms)	C (ms)	F (ms)	C (ms)	F (ms)	C (ms)	E (ms)
Article	10.245	4.515	3.153	0.594	3.751	0.252	-	2.379
Dictionary	8.315	1.054	3.305	0.574	5.018	0.239	-	1.865
Health	8.261	1.002	2.391	0.437	2.864	0.258	-	2.551
Movie	15.512	1.649	5.190	0.856	5.543	0.375	-	2.828
Newspaper	14.223	1.847	4.388	0.650	7.009	0.264	-	2.925
Shopping	22.682	3.598	10.184	1.414	17.611	0.651	-	6.988
Trip	29.543	2.701	18.365	2.860	26.282	1.408	-	8.411
	16.461	2.542	6.921	1.064	10.377	0.486	-	4.150

Similar tags grouped together.
Abbreviations:
ASTFM : <article>, <section>, <table>, <fieldset>, <main>
H : <h1>, <h2>, <h3>, <header>

In Table 5, the DOM-based parsers including MS_HtmlDocument, AngleSharp, and HAP take an average creation time of 16.461 ms, 6.921 ms, and 10.377 ms, respectively to create a DOM tree of all web pages. Additionally, these parsers give an average finding time of 2.542 ms, 1.064, and 0.486 ms, respectively in order to reach the content of the opening tag. Finally, the average extraction time reusults of these algorithms are about 19.004 ms, 7.985 ms, and 10.863 ms, respectively. The MS_HtmlDocument parser gives the worst result in both creating and finding. The average duration of the AngleSharp parser in the creation of a DOM tree is better than the use of the HAP parser. On the other hand, the average finding time of the HAP parser is better than the average finding time of other DOM-based parsers. Although the HAP parser has a longer average time for creating a DOM tree, it gives faster results in the finding of the element. The AngleSharp parser can be selected if there is only one search on a DOM tree. On the other hand, the HAP parser is suitable if more than one search is performed for this task. SET Parser has an average extraction time of 4.150 ms. In the DOM-based algorithms, the number of elements is an important issue, but in SET Parser, the web page size and the number of characters are crucial issues for the extraction process. SET Parser searches in a text of a web page without performing the creating process. Table 5 indicates that SET Parser allows for fast extraction when compared to the DOM-based parsers. The UzunExt approach is intended to make this extraction faster than other extraction parsers.

C. EXTRACTION WITH THE UzunExt APPROACH

In this section, firstly the determination of the most suitable search algorithm for the search part of the UzunExt approach is investigated. Then, the effects of different extraction ways, the effects of the additional information, efficiency of different ways during the crawling process and the effects of tags are examined for a better understanding of the UzunExt approach.

TABLE 6. The average duration time of searching algorithms.

	Index = 0			Appropriate index		
Searching Algorithm	P (ms)	S (ms)	T (ms)	P (ms)	S (ms)	T (ms)
Skip Search	0.001	0.090	0.091	0.001	0.009	0.010
Tuned Boyer Moore	0.162	0.044	0.206	0.160	0.004	0.164
Horspool	0.159	0.041	0.200	0.157	0.004	0.161
Backward Nondeterministic Dawg Matching	0.129	0.079	0.208	0.128	0.023	0.151
Optimal Mismatch	0.162	0.047	0.209	0.161	0.005	0.166
Boyer Moore	0.162	0.052	0.214	0.157	0.005	0.162
Raita	0.174	0.044	0.219	0.173	0.004	0.177
Indexof_Ordinal	-	0.039	0.039	-	0.004	0.004
Indexof	-	3.538	3.538	-	0.338	0.338

P: Processing step, S: Searching step, T: Total

TABLE 7. The average extraction time of three different ways in the UzunExt for each category.

	(1)	(2)			(3)	
Category	ms	Before (ms)	After (ms)	Avg.	(ms)	NCE
Article	0.117	0.052	0.110	0.091	0.083	4530.13
Dictionary	0.054	0.051	0.020	0.030	0.024	3013.13
Health	0.166	0.348	0.050	0.149	0.133	7511.51
Movie	0.101	0.146	0.016	0.059	0.045	2951.16
Newspaper	0.090	0.077	0.034	0.048	0.034	4516.15
Shopping	0.192	0.149	0.084	0.106	0.075	4352.16
Trip	0.225	0.212	0.064	0.116	0.057	9366.17
	0.130	0.119	0.055	0.077	0.056	4921.98

NCE: Number of Characters after Extraction

1) SEARCHING ALGORITHMS

Searching algorithms, which is one of the crucial parts of the UzunExt approach, has been studied many times [51]. Bulu *et al.* [8] have observed the performances of 31 different searching algorithms on web pages. Seven of these algorithms including Skip Search [52], Tuned Hume and Sunday [53], Horspool [54], Backward Nondeterministic Dawg Matching [55], Optimal Mismatch [56], Boyer and Moore [57] and Raita [58] are used in our study. Moreover, IndexOf method, the .NET Approach's base class library, is compared with these algorithms. This method performs a culture-sensitive search. For optimizing string performance, the culture-sensitive search is ignored with StringComparison.Ordinal (Indexof_Ordinal) that is one of the enumeration values in order to specify the patterns for the search.

A search algorithm starts from the beginning of the text and continues until the pattern is found in this text. That is, the starting index of a search algorithm is zero, and the ending index is the length of the web page. Determining the starting index can speed up the search by reducing the number of comparisons in the search. However, determining this index is a difficult task. The UzunExt approach selects and uses the appropriate index for a website and its extraction patterns. The appropriate index is the minimum index obtained from 30 web pages in the experiments for a website and its extraction patterns. Table 6 shows the average duration time of the search algorithms and the effects of selecting the appropriate index for these algorithms.

Most search algorithms have two main phases: the preprocessing phase and the searching phase. The preprocessing phase of an algorithm consists of collecting some information like statistical data or character ordering about a pattern and making a model to search. The searching phase finds the index of a pattern in a text by using information obtained from the preprocessing phase. Nevertheless, some algorithms do not need a preprocessing phase. In Table 6, the Indexof_Ordinal method, which does not require a preprocessing phase, gives the best result with 0.039 ms. The Indexof_Ordinal method is much more effective when compared to the Indexof method. The algorithm that makes the pattern searchable as soon as possible is the Skip Search algorithm with a preprocessing time of 0.001 ms. The most appropriate starting index for each website has been selected for understanding the effect of starting from a forward index (see the appropriate index in Table 6). According to Table 6, using the appropriate index has been a positive influence on the searching phase. After determining the appropriate index, all the average searching time are improved. Table 5 indicates that the algorithms, which spend much time on preprocessing, are not suitable for this task. In this study, the Indexof_Ordinal method is used for the extraction step of the approach.

2) EFFECTS OF DIFFERENT WAYS OF THE UzunExt APPROACH

The UzunExt approach can be adjusted in three different ways. (1) The approach can extract relevant content without using any additional information. In this case, the search is performed from the top of the document, our algorithm extracts each element found by the searching process, and the algorithm continues this search until the end index of a document. This way is the simple extraction of the approach. (2) The second way, which is the default way of the approach, predicts the additional information including startingPosition, innerTagCount, and repetition during the crawling process. (3) The third way, which is created to understand the effectiveness of the default way better, uses predefined constants for additional information. These constants can be set manually by an expert programmer. The third way can be considered as a gold standard for the approach. In this study, the optimal values obtained from 30 web pages of each website rules are utilized as constants. Table 7 indicates the average extraction time of three different ways and the number of average characters obtained from extraction results for different categories.

The average extraction time of the first way, which does not use any additional information, is about 0.130 ms. The AngleSharp parser, which has the best average extraction time, is about 7.985 ms. That is, it means that the first way provides an improvement of 98.37% compared to the AngleSharp parser. Moreover, the average extraction time of SET Parser is about 4.150 ms. The first way provides an improvement of 48.03% compared to SET Parser. In the second way,

TABLE 8. The average extraction time of three different ways in the UzunExt for each category.

	AE	Improv.	AETI	AETR	Improv.
Number of extraction tasks	7410	-	5010	6570	-
Simple Extraction	0.130	-	0.088	0.139	-
Only startingPosition	0.095	27.01%	-	-	
Only innerTagCount	0.129	1.42%	0.085	-	3.08%
Only repetition	0.093	28.58%	-	0.097	30.32%

AE: All Extractions
Improv. : Improvement
AETI : Appropriate extraction tasks for innerTagCount
AETR : Appropriate extraction tasks for repetition

the startingPosition information starts to be learned from the first web page of a website. Information of InnerTagCount and repetition are decided after obtaining ten web pages, and these information are used in the following web pages. In Table 7, the average extraction time with only the position information is 0.119 ms on the first ten web pages. On the next twenty web pages, the other two information improves the average time with 0.055 ms. The average duration time of the extraction process influences the length of the extraction results as well as the size of the web page. As a result, the second way provides significant improvement according to the first way. The third way uses the most appropriate values as additional information. This way yields the best average extraction time of 0.056 ms. However, this technique needs to downloaded web pages for obtaining these additional information. But the second way continues to improve the time efficiency of the extraction step during the crawling process. The first way and the second way provide very close results after the first ten web pages.

3) EFFECTS OF THE ADDITIONAL INFORMATION IN THE UzunExt APPROACH

In this experiment, the effects of each information on the approach are examined separately for understanding improvements. startingPosition information is useful for all extraction tasks. Nevertheless, the information of innerTagCount and repetition are not appropriate for all extractions. Therefore, the contribution of these information is also examined concerning the number of appropriate extraction tasks. Table 8 shows the effects of these information for all extractions and appropriate extractions.

There are 247 extraction patterns, and 30 web pages for a website are utilized in the experiments. In other words, 7410 extraction tasks have been completed. The startingPosition information improves the average extraction time from 0.130 ms to 0.095 ms. Thus, there is a significant improvement with about 27.01% when using only the startingPosition information. The average extraction time in the innerTagCount fell from 0.130 ms to 0.129 ms. This information provides a slight improvement with about 1.42%. When using only the repetition information in the extraction task, the average extraction time enhances from 0.130 ms to 0.093. This improvement is about 28.58%. The number of appropriate extraction tasks for the information of innerTagCount and repetition is 5010 and 6570, respectively. That is, these information can be useful in these extraction tasks. The innerTagCount information enhances a little improvement with about 3.08% in the 5010 extraction tasks. The repetition information provides a significant improvement with about 30.32% in the 6570 extraction tasks. Consequently, the information of startingPosition and repetition yield a significantly better extraction time that the innerTagCount information.

4) COMPARISON OF THREE DIFFERENT WAYS DURING THE CRAWLING PROCESS

Figure 3 indicates the average extraction time for three different ways adjusted in the UzunExt approach. The average extraction time for each web page of a website is considered in this experiment. In three different ways, the average minimum extraction times are 0.099 ms, 0.023 ms, and 0.026 ms, respectively. On the other hand, the average maximum extraction time is 0.343 ms, 0.257 ms, and 0.259 ms, respectively. Depending on the file size and the number of characters resulting from extractions, all of the ways are affected similarly. The contribution of the additional information in Figure 3 is noticeable. The average time results of the second and third ways are very close, especially after the 10th web page.

5) EFFECTS OF TAGS FOR THE UzunExt APPROACH

In this experiment, the third way of the UzunExt approach, which has the most appropriate additional information, is used for understanding the effects of the tags. Table 9 indicates the effects of tags on the average extraction time, the average number of characters for content obtained from the extraction step of the UzunExt approach and the number of extraction tasks for tags.

In Table 9, the best average extraction time is 0.008 ms with the <h1>, <h2>, <h3>, and <header> (H). The average number of characters for content in these tags is about 173.26. This value is significantly less than the average number of characters of other tags. These tags allow for faster extraction as expected because they contain fewer characters. When the number of characters increases, the extraction time slows down as seen in the <p>, <span>, and <ul> tags. These tags have an average extraction time of 0.028 ms and the content of these tags contains an average of 173.26 characters. These results show the correlation between the extraction time and the number of characters. However, the tags of <article>, <section>, <table>, <field>, <main> (ASTFM) and the <div> tag don't support the expected findings. The average extraction time of the ASTFM tags is about 0.041 ms, and the number of characters in the content of these tags is 17391.97. The number of characters in the content of the <div> tag is less than the number of characters in the content of the ASTFM tags. However, the average extraction time of the <div> tag is about 0.081 ms which is slower than the average

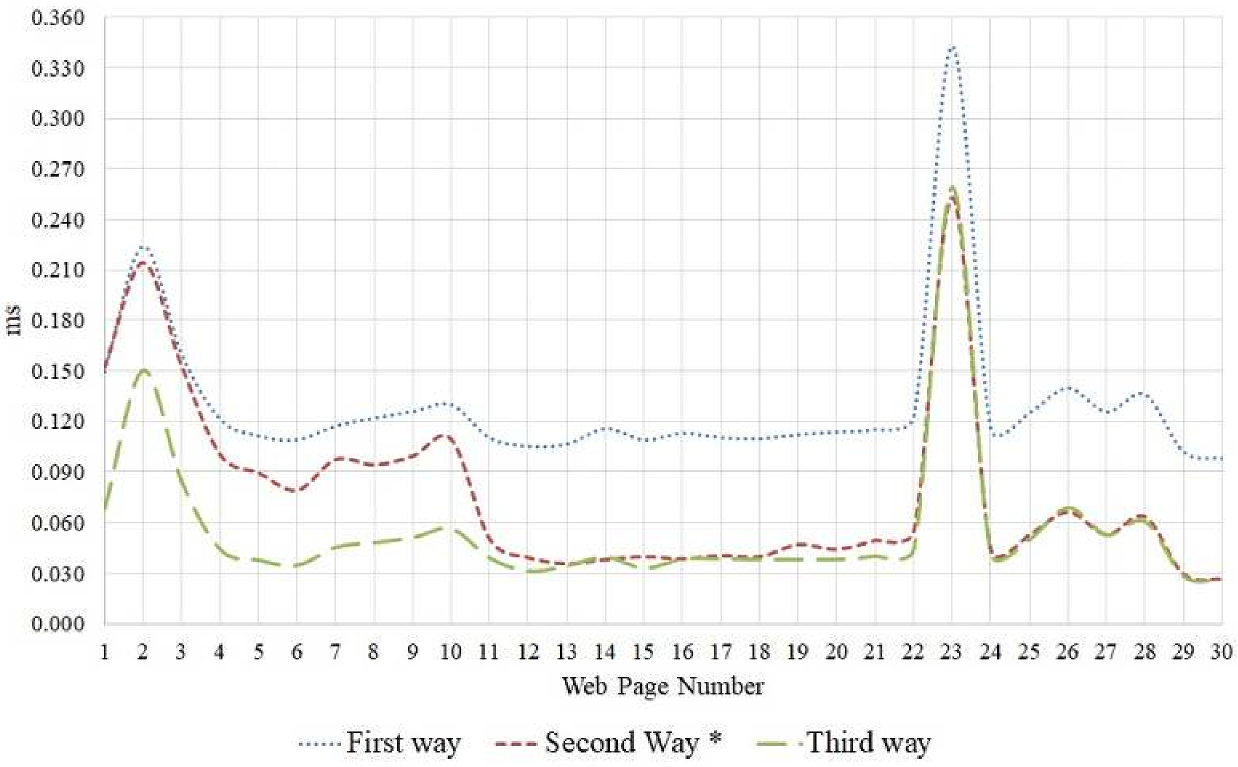


FIGURE 3. Three different ways of the UzunExt approach.

TABLE 9. Effects of tags.

		innerTagCount	
	All Results	Appropriate	Not
	Extraction Time (ms)		
ASTFM	0.041	0.010	0.224
div	0.081	0.012	0.151
H	0.008	0.008	-
p, span, ul	0.028	0.028	0.029
	0.056	**0.012**	**0.147**
	Avg. Number of characters		
ASTFM	17391.97	4556.87	94402.58
div 6068.05			
1952.48 10350.47			
H	173.26	173.26	-
p, span, ul	2305.32	1047.96	6391.72
	4899.66	**1376.70**	**12253.84**
	Avg. Number of characters		
ASTFM	420	360	60
div	4530	2310	2220
H	1950	1950	-
p, span, ul	510	390	120
	7410	**5010**	**2400**

extraction time of the ASTFM. Because of this slowness, a <div> tag can contain a number of other <div> tags. That is, this tag is a nested structure. Therefore, the number of inner tag name (innerTagCount) is also examined in Table 9.

The <div> tag is trendy to layout a web page, so the number of extraction tasks is 4530 for this tag. Knowing this number can provide efficiency in the extraction step. When the appropriate number is known, the average extraction time is only 0.012 ms for 2310 extraction tasks. On the other hand, the average extraction time increases to 0.151 ms for 2220 extraction tasks. However, another reason for this increase is the number of characters in the content. The number of characters is about 1952.48 and 10350.47 for the average extraction time of 0.012 ms and 0.151 ms, respectively. The <div> tag is mostly used nowadays, but we expect that the usage rate of the tags of <article>, <section>, and <main> will increase in the future and these elements can take the place of the <div> tag. These tags are new tags that are introduced in HTML5 so the usage rate of these elements on web pages is low.

The following suggestions may be offered for the programmer who prepares the extraction patterns for web data extraction. The H tags are especially useful for obtaining the title of a web page. The p, span, and ul tags can be used to extract content containing few characters. The tags of <article>, <section>, and <main> can be suggested to get the main content. However, the <div> tag is highly preferred by web designers. According to the experiments, we do not particularly recommend a <div> tag that contains too many inner <div> tags as an extraction pattern. Moreover, an extraction pattern, which has a unique value, may be preferred in terms of fast extraction. The general decision is more appropriate tags as an extraction pattern that does not contain inner tags, is non-repetitive, and has fewer characters in their content.

VII. CONCLUSION

Most of the previous studies on web scraping are about automatically extracting web content by ignoring time efficiency. These studies use traditional extraction methods based on a DOM tree which takes all elements in a web page into account. However, such an approach increases the time cost in the extraction step. In this study, a novel approach, which provides time efficiency by employing the string methods and additional information obtained from web pages of a website during a crawling process, is introduced. The average extraction time of the simple way in this approach gives better efficiency than the average extraction time of AngleSharp parser with approximately 60 times. Moreover, this approach provides additional information including starting position, the number of the nested tag, and repetition of the tag. Thanks to these information, the average extraction time is about 2.35 times better than the average extraction time of the simple way of the approach. Simply, while an extraction process is completed in about 140 days with the DOM-based parser, this process can be finished in approximately one day by use of the UzunExt approach. From another perspective, if one wishes to benefit from parallel computing to gain time, a cluster of 140 cores and an enormous amount of memory will be needed, while the same process can be done with only a single, moderately equipped computer using the proposed approach. Notably, the additional information of starting position and repetition in this approach dramatically improves the time efficiency of the extraction step.

A web designer/developer determines elements, attributes for these elements, and a hierarchy between these elements for creating a website. A researcher tries to discover the desired ones from these elements by developing algorithms, techniques, methods, and approaches. Recently, the determination stage of the attribute in these elements has gained importance. There is a community activity, schema.org, to determine the attributes of elements for the use of web designers. This community offers vocabularies covered entities, relationships between entities and actions for different fields. Many applications from Google, Microsoft, Pinterest, Yandex, and over 10 million websites use these vocabularies to identify their elements. Besides, these companies support the creation of these vocabularies. When elements and their attributes are known, it will be much easier to prepare an extraction pattern, especially for the UzunExt approach.

The UzunExt approach is an open-source project and available via the web page https://github.com/erdincuzun/UzunExt. It can be easily integrated into other projects that need web content. Moreover, it can be used to enhance the time efficiency of automatic web content extraction studies by determining an appropriate element. In particular, it can be adapted to crawler applications in order to increase the time efficiency of the content extraction step.

In this study, we deal with the text of a web page. However, this text can be changed with Ajax requests. Traditional solutions [59] use an embedded browser to render web pages and execute these requests. However, these solutions are expensive, in terms of time and network traffic. In the future, we will try to detect these requests without rendering a web page. Moreover, we also plan to develop an effective and efficient web scraper that can create datasets automatically for different purposes. Besides, we will also aim to develop an algorithm that automatically obtains the desired element without creating a DOM tree.

ACKNOWLEDGMENT

The author would like to offer his special thanks to Assist. Prof. Dr. Heysem Kaya for comments, support, and encouragement.

REFERENCES

[1] E. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner, "Web data extraction, applications and techniques: A survey," *Knowl.-Based Syst.*, vol. 70, pp. 301–323, Nov. 2014.

[2] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *J. Big Data*, vol. 2, no. 1, p. 1, Dec. 2015.

[3] E. Uzun, H. V. Agun, and T. Yerlikaya, "A hybrid approach for extracting informative content from Web pages," *Inf. Process. Manage.*, vol. 49, no. 4, pp. 928–944, Jul. 2013.

[4] E. Uzun, E. S. Güner, Y. Kılıçaslan, T. Yerlikaya, and H. V. Agun, "An effective and efficient Web content extractor for optimizing the crawling process," *Softw., Pract. Exper.*, vol. 44, no. 10, pp. 1181–1199, Oct. 2014.

[5] E. Uçar, E. Uzun, and P. Tüfekci, "A novel algorithm for extracting the user reviews from Web pages," *J. Inf. Sci.*, vol. 43, no. 5, pp. 696–712, Oct. 2017.

[6] Y.-C. Wu, "Language independent Web news extraction system based on text detection framework," *Inf. Sci.*, vol. 342, pp. 132–149, May 2016.

[7] E. Uzun, H. N. Bulu , A. Doruk, and E. Özhan, "Evaluation of Hap, AngleSharp and HtmlDocument in Web content extraction," in *Proc. Int. Sci. Conf. (UNITECH)*, vol. 2, 2017, pp. 275–278.

[8] H. N. Bulu , E. Uzun, and A. Doruk, "Comparison of string matching algorithms in Web documents," in *Proc. Int. Sci. Conf. (UNITECH)*, vol. 2, 2017, pp. 279–282.

[9] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 1–21, Feb. 1969.

[10] S. Chakrabarti, M. van den Berg, and B. Dom, "Focused crawling: A new approach to topic-specific Web resource discovery," *Comput. Netw.*, vol. 31, nos. 11–16, pp. 1623–1640, May 1999.

[11] K. Shchekotykhin, D. Jannach, and G. Friedrich, "XCrawl: A high-recall crawling method for Web mining," *Knowl. Inf. Syst.*, vol. 25, no. 2, pp. 303–326, Nov. 2010.

[12] B. Pinkerton, "Finding what people want: Experiences with the WebCrawler," in *Proc. 2nd Int. World Wide Web Conf.*, vol. 94, 1994, pp. 17–20.

[13] S. Chakrabarti, K. Punera, and M. Subramanyam, "Accelerated focused crawling through online relevance feedback," in *Proc. 11th Int. Conf. World Wide Web (WWW)*, 2002, pp. 148–159.

[14] M. Diligentit, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, "Focused crawling using context graphs," in *Proc. 26th Int. Conf. Very Large Data Bases (VLDB)*, 2000, pp. 527–534.

[15] H. Dong and F. K. Hussain, "SOF: A semi-supervised ontology-learning-based focused crawler," *Concurrency Comput., Pract. Exper.*, vol. 25, no. 12, pp. 1755–1770, Aug. 2013.

[16] M. Kumar, R. Bhatia, and D. Rattan, "A survey of Web crawlers for information retrieval," *WIREs Data Mining Knowl. Discovery*, vol. 7, no. 6, p. e1218, Nov. 2017.

[17] F. Menczer, G. Pant, and P. Srinivasan, "Topical Web crawlers: Evaluating adaptive algorithms," *ACM Trans. Internet Technol.*, vol. 4, no. 4, pp. 378–419, Nov. 2004.

[18] N. Kushmerick, *Wrapper Induction for Information Extraction*. Seattle, Washington: Univ. of Washington, 1997.

[19] L. Liu, C. Pu, and W. Han, "XWRAP: An XML-enabled wrapper construction system for Web information sources," in *Proc. 16th Int. Conf. Data Eng.*, 2000, pp. 611–621.

[20] R. Das and I. Turkoglu, "Creating meaningful data from Web logs for improving the impressiveness of a Website by using path analysis method," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 6635–6644, Apr. 2009.

[21] B. Fazzinga, S. Flesca, and A. Tagarelli, "Schema-based Web wrapping," *Knowl. Inf. Syst.*, vol. 26, no. 1, pp. 127–173, Jan. 2011.

[22] H.-Y. Kao, S.-H. Lin, J.-M. Ho, and M.-S. Chen, "Mining Web informative structures and contents based on entropy analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 1, pp. 41–55, Jan. 2004.

[23] M. Zachariasova, R. Hudec, M. Benco, and P. Kamencay, "Automatic extraction of non-textual information in Web document and their classification," in *Proc. 35th Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2012, pp. 753–757.

[24] Z. Li, W. K. Ng, and A. Sun, "Web data extraction based on structural similarity," *Knowl. Inf. Syst.*, vol. 8, no. 4, pp. 438–461, Nov. 2005.

[25] H. S. Maghdid, "Web news mining using new features: A comparative study," *IEEE Access*, vol. 7, pp. 5626–5641, 2019.

[26] T. Gottron, "Content code blurring: A new approach to content extraction," in *Proc. 19th Int. Conf. Database Expert Syst. Appl. (DEXA)*, Sep. 2008, pp. 29–33.

[27] T. Weninger, W. H. Hsu, and J. Han, "CETR: Content extraction via tag ratios," in *Proc. 19th Int. Conf. World wide Web (WWW)*, 2010, pp. 971–980.

[28] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm, "DOM-based content extraction of HTML documents," in *Proc. 12th Int. Conf. World Wide Web*, 2003, pp. 207–214

[29] P. A. R. Qureshi and N. Memon, "Hybrid model of content extraction," *J. Comput. Syst. Sci.*, vol. 78, no. 4, pp. 1248–1257, Jul. 2012.

[30] C. Mantratzis, M. Orgun, and S. Cassidy, "Separating XHTML content from navigation clutter using DOM-structure block analysis," in *Proc. 16th ACM Conf. Hypertext Hypermedia (HYPERTEXT)*, Jan. 2005, pp. 145–147.

[31] A. Finn, N. Kushmerick, and B. Smyth, "Fact or fiction: Content classification for digital libraries," in *Proc. Joint DELOS-NSF Workshop, Personalization Recommender Syst. Digit. Libraries*, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/citations;jsessionid=8E0FC70BEE7 DFA696487A2F7C6B622FA?doi=10.1.1.21.3834

[32] G. Adam, C. Bouras, and V. Poulopoulos, "CUTER: An efficient useful text extraction mechanism," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshops (AINA)*, May 2009, pp. 703–708.

[33] R. Gunasundari, "A study of content extraction from Web pages based on links," *Int. J. Data Mining Knowl. Manage. Process*, vol. 2, no. 3, pp. 230–236, May 2012.

[34] D. Insa, J. Silva, and S. Tamarit, "Using the words/leafs ratio in the DOM tree for content extraction," *J. Log. Algebr. Program.*, vol. 82, no. 8, pp. 311–325, Nov. 2013.

[35] Z. Bar-Yossef and S. Rajagopalan, "Template detection via data mining and its applications," in *Proc. 11th Int. Conf. World Wide Web (WWW)*, 2002, pp. 580–591.

[36] D. Chakrabarti, R. Kumar, and K. Punera, "Page-level template detection via isotonic smoothing," in *Proc. 16th Int. Conf. World Wide Web (WWW)*, 2007, p. 61.

[37] L. Fu, Y. Meng, Y. Xia, and H. Yu, "Web content extraction based on Webpage layout analysis," in *Proc. 2nd Int. Conf. Inf. Technol. Comput. Sci.*, Jul. 2010, pp. 40–43.

[38] Y. Wang, B. Fang, X. Cheng, L. Guo, and H. Xu, "Incremental Web page template detection by text segments," in *Proc. IEEE Int. Workshop Semantic Comput. Syst. (WSCS)*, Jul. 2008, pp. 174–180.

[39] D. Cai, S. Yu, J. R. Wen, and W. Y. Ma, "Extracting content structure for Web pages based on visual representation," in *Web Technologies and Applications* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence: Lecture Notes in Bioinformatics), vol. 2642. Berlin, Germany: Springer, 2003, pp. 406–417.

[40] J. L. Hong, E.-G. Siew, and S. Egerton, "ViWER-data extraction for search engine results pages using visual cue and DOM tree," in *Proc. Int. Conf. Inf. Retr. Knowl. Manage. (CAMP)*, Mar. 2010, pp. 167–172.

[41] D. Yang and J. Song, "Web content information extraction approach based on removing noise and content-features," in *Proc. Int. Conf. Web Inf. Syst. Mining (WISM)*, vol. 1, Oct. 2010, pp. 246–249.

[42] B. Zhang and X.-F. Wang, "Content extraction from chinese Web page based on title and content dependency tree," *J. China Universities Posts Telecommun.*, vol. 19, pp. 147–189, Oct. 2012.

[43] L. N. L. Figueiredo, G. T. de Assis, and A. A. Ferreira, "DERIN: A data extraction method based on rendering information and n-gram," *Inf. Process. Manage.*, vol. 53, no. 5, pp. 1120–1138, Sep. 2017.

[44] J. Zeleny, R. Burget, and J. Zendulka, "Box clustering segmentation: A new method for vision-based Web page preprocessing," *Inf. Process. Manage.*, vol. 53, no. 3, pp. 735–750, May 2017.

[45] D. Song, F. Sun, and L. Liao, "A hybrid approach for content extraction with text density and visual importance of DOM nodes," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 75–96, Jan. 2015.

[46] Y. Fang, X. Xie, X. Zhang, R. Cheng, and Z. Zhang, "STEM: A suffix tree-based method for Web data records extraction," *Knowl. Inf. Syst.*, vol. 55, no. 2, pp. 305–331, May 2018.

[47] Z. Tan, C. He, Y. Fang, B. Ge, and W. Xiao, "Title-based extraction of news contents for text mining," *IEEE Access*, vol. 6, pp. 64085–64095, 2018.

[48] N. Chomsky, "Three models for the description of language," *IEEE Trans. Inf. Theory*, vol. IT-2, no. 3, pp. 113–124, Sep. 1956.

[49] M. Sipser, "Introduction to the theory of computation," *ACM Sigact News*, vol. 27, no. 1, pp. 27–29, 1996.

[50] E. Uzun, T. Yerlikaya, and M. Kurt, "A lightweight parser for extracting useful contents from Web pages," in *Proc. 2nd Int. Symp. Comput. Sci. Eng. (ISCSE)*, Ku adasi, Turkey, 2011, pp. 67–73.

[51] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[52] C. Charras, T. Lecro, and J. D. Pehoushek, "A very fast string matching algorithm for small alphabets and long patterns," in *Combinatorial Pattern Matching* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence: Lecture Notes in Bioinformatics), vol. 1448. Berlin, Germany: Springer, 1998, pp. 55–64.

[53] A. Hume and D. Sunday, "Fast string searching," *Softw., Pract. Exper.*, vol. 21, no. 11, pp. 1221–1248, Nov. 1991.

[54] R. N. Horspool, "Practical fast searching in strings," *Softw., Pract. Exper.*, vol. 10, no. 6, pp. 501–506, Jun. 1980.

[55] G. Navarro and M. Raffinot, "A bit-parallel approach to suffix automata: Fast extended string matching," *Combinatorial Pattern Matching* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence: Lecture Notes in Bioinformatics), vol. 1448. Berlin, Germany: Springer, 1998, pp. 14–33.

[56] D. M. Sunday, "A very fast substring search algorithm," *Commun. ACM*, vol. 33, no. 8, pp. 132–142, Aug. 1990.

[57] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, Oct. 1977.

[58] T. Raita, "Tuning the boyer-Moore-horspool string searching algorithm," *Softw., Pract. Exper.*, vol. 22, no. 10, pp. 879–884, Oct. 1992.

[59] R. R. Fayzrakhmanov, E. Sallinger, B. Spencer, T. Furche, and G. Gottlob, "Browserless Web data extraction," in *Proc. World Wide Web Conf. World Wide Web (WWW)*, 2018, pp. 1095–1104.

ERDINÇ UZUN received the bachelor's, master's, and Ph.D. degrees from the Department of Computer Engineering, Trakya University, Edirne, Turkey, in 2001, 2003, and 2007, respectively.

He is currently supervising five graduate students. He is interested in developing a career that combines teaching and research while maintaining his interest in the field of information retrieval, data mining, and natural language processing. After graduating in 2001, he started his academic career at the Department of Computer Engineering, Trakya University, in 2001, where he worked as a Research Assistant, from 2001 to 2007, for seven years. In 2007, he completed his doctoral thesis on the development of a web-based system that can automatically learn subcategorization frames. It is a thesis that combines the fundamental fields of computer science such as information retrieval, machine learning, and natural language processing. Later in 2007, he started his career at the Department of Computer Engineering, Tekirda Namık Kemal University. He was the vice dean in his faculty, from 2008 to 2010. After 2007, he worked in web search, web mining, and web content extraction. His work, "A hybrid approach for extracting informative content from web pages," in 2013, was one of the most cited works in the field of web content extraction with over 50 citations. He supports education not only with courses but also with lecture notes and blog posts shared on erdincuzun.com. He has more than 30 publications and more than 150 citations. He has also been a member of the Board of Directors of the Faculty, since 2017. In 2019, he started to work as the Director of the Çerkezköy Vocational School of his University. He worked as a referee and panelist in various TÜB TAK programs.