Proceeding Number: 100/15

A Lightweight Parser for Extracting Useful Contents from Web Pages

Erdinç Uzun[1], Tarık Yerlikaya[2], Meltem Kurt[3]

[1] Computer Engineering Department / Namik Kemal University

[2, 3] Computer Engineering Department / Trakya University

[1] erdincuzun@nku.edu.tr, [2]tarikyer@trakya.edu.tr, [3]meltemkurt@trakya.edu.tr

Abstract. In many web content extraction applications, parsing is crucial issue for obtaining the necessary information from web pages. A DOM parser is preferably used in this task. However, major problems with using a DOM parser are time and memory consumption. This parser is an inefficient solution for applications which need web content extraction between tags. Therefore, we develop a lightweight parser, namely SET Parser, which is utilized regular expressions and string functions for extracting these tags. In experiments, 2000 web pages are crawled from Sabah and Milliyet for experimental setup. The average size of a DOM tree created from these web pages is as large as 9.58 times of the average size of the web pages. On the other hand, SET Parser in parsing contents is less time consuming than DOM. These experiments indicate that SET Parser is more useful for extracting contents from web pages.

Keywords: *Parsers, DOM, Content Extraction, Time and Memory Consumption*

1 Introduction

Document Object Model (DOM, www.w3.org/DOM) parsing is widely used a key operation for processing web pages. Especially, a DOM parser is the preferred way of extracting web contents [1], [2], [3], [4]. Web documents have a hierarchy of informational units called nodes; DOM is a way of describing those nodes and the relationships between them. In DOM, everything is an object that consists of element names and contents of them. A DOM parser presents a document as a tree-structure in memory. However, the size of DOM tree created from a web document is larger than the size of the original web document. This means that memory and time consumption will increase constantly. For exceptionally large documents, parsing and loading the entire document can be slow and resource intensive.

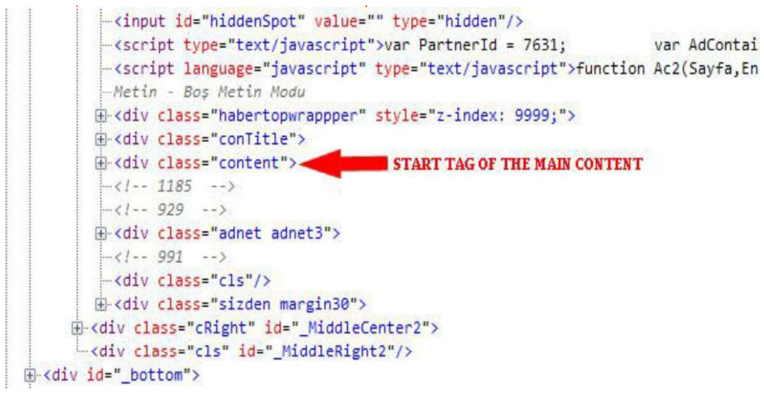
Web pages often contain irrelevant contents such as pop up ads, advertisements, banners, unnecessary images, extraneous links etc. around the body of an article that distracts users from actual content. There are lots of applications about extraction of useful and relevant content from web pages including text summarization, cell phone and pda browsing, speech rendering for the visually impaired [5], [6]. For extraction of relevant content,unnecessary tags and contents in Web pages should be removed. In general, extracting content of text in conventional techniques required knowledge on structure of the text [7]. Web pages are semi-structured documents mostly written in HTML that defines tags. These tags can be structured with DOM parser. However, Cai et al. [8] proposed that only some tags (<table>, <tbody>, <tr>, <td>, <p>, <ul> and <li>) are very important and commonly used in web pages. Yerlikaya and Uzun [9] use the layout html tags, <div>, <table> and <ul>, for obtaining relevant contents. In content extraction, because of A DOM parser takes all tags into account, it is not efficient.

In search engine for Turkish (SET[6]) Project, we developed an intelligent crawler that automatically detects irrelevant contents and extracts relevant contents. We realized that memory and time consumptions are important issue in this task. Therefore, we developed a lightweight parser (namely SET Parser) for reducing these consumptions. In this study, we describe this parser and examine the improvements in memory and time.

The paper is organized as follows: Section 2 gives general information about DOM, Regular Expressions and the use of them C# programming language. Section 3 introduces the SET parser which is utilized regular expressions and string functions for extracting contents. Section 4 examines memory and time consumptions in DOM and SET Parser. The paper ends with some concluding remarks.

2 DOM and Regular Expressions

The DOM is a W3C (World Wide Web Consortium) standard. The DOM defines a standard for accessing documents by using Javascript and Jscript that are client-side programming languages. Each document is treated as the Document Object; this object is queried, searched and updated. The following figure is an example Document Object for a web page of Milliyet.



```
      <input id="hiddenSpot" value="" type="hidden"/>
      <script type="text/javascript">var PartnerId = 7631;         var AdContai
      <script language="javascript" type="text/javascript">function Ac2(Sayfa,En
      Metin - Boş Metin Modu
    ⊞ <div class="habertopwrappper" style="z-index: 9999;">
    ⊞ <div class="conTitle">
    ⊞ <div class="content">
      <!-- 1185  -->
      <!-- 929  -->
    ⊞ <div class="adnet adnet3">
      <!-- 991  -->
      <div class="cls"/>
    ⊞ <div class="sizden margin30">
  ⊞ <div class="cRight" id="_MiddleCenter2">
    <div class="cls" id="_MiddleRight2"/>
⊞ <div id="_bottom">
```

Fig 1. En example DOM Object from Milliyet Web Pages

Fig 1. presents a DOM example (as a tree-structure) for a web page. DOM is a standard model and programming interface for HTML. Manyprogramming languages support DOM. For example, when extracting links or images from a web page, the following codes can be used in C#.

```
IHTMLDocument2 htmlDocument = new mshtml.HTMLDocumentClass();
htmlDocument.write(a web page);
IHTMLElementCollection Elements_links = htmlDocument.links;
IHTMLElementCollection Elements_images = htmlDocument.images;
```

To access the DOM programmatically, *mshtml* library is used. *write* function creates a DOM object for a web document. Nonetheless, while structuring a web document, a DOM object takes all tags into consideration. It is clear at this point that memory consumption and time consumption will increase. Moreover, regular expressions can be used for extracting links and images.

The origin of regular expressions lies in automata and formal language theory. Mathematician Stephen Cole Kleene described these models of using his mathematical notation called regular sets in the 1950s. Regular expression is a language that can be used to edit texts or obtain sub-texts that based on a set of defined rules.

[6]Classes of SET developed in C# programming language are open source and available via the web page http://bilgmuh.nku.edu.tr/SET/. Moreover, you can try all modules by using this web page.

Regular expressions can be used for obtaining relevant contents from web pages. For example, the following table indicates patterns that are used for extracting links and images from web pages.

Table 1. Regular expression patterns for extracting links and images

Links	(?<=href=\").*?(?=\")
Images	<\\s*img [^\\>]*src\\s*=\\s*([\"\\'])(.*?)>

Regular expressions are supported by many programming languages such as C, C++, .NET, Java, Perl, Ruby, Python to search and manipulate text based on patterns. For accessing .Net regular expression engine from C#, we import the namespace

“System.Text.RegularExpressions”.

Regular expressions can save you time when considering DOM. Therefore, we write two functions which are used regular expression for this task. However, regular expressions have been a bottleneck for the unknown nested structure that is a layout stored within the structure of another layout. Now, we describe the main function of SET Parser.

3 SET Parser

SET Parser has three functions. Two functions of SET Parser are used only regular expressions for extracting links and images.Nevertheless, unfortunately regular expressions don’t give accurate results in the nested structure of tags. Therefore, we write codes for obtaining inner contents from layout tags.

```
Function Find Contents of a given tag
input a given tag and a web page
prepare pattern of regular expression of a given tag
determine cnt_giventag = count of a given tag by using pattern in a web page
determine start_tag of a given tag
determine end_tag of given tag
output array results
for i=0 to cnt_giventag do
start_position of i. given tag in a web page
    NEXT: end_position of temp_tag in a web page
    content = extract string between start_position and end_position
    cnt_opentags= determine the count of start tags by using start_tag
    cnt_endtags = determine the count of end tags by using end_tag
    if(count of open tags != count of close tags)
        find next end_position;
        goto NEXT;
    end if
    else if
        results[i] = contents
    end else if
end for
return results
```

```
end function
```

For example, when using SET Parser in a web page for <div class="content"> tag, firstly regular expression pattern <div.class="content"> (by replacing blanks with a point) is prepared for calculating the count of this pattern in a web page. Then, start tag as "<div" and end tag </div are created as appropriate to the tag. Start position and end positions are found by using these tags. "indexof" string function can be used for this task. "substring" can be utilized for extracting a proposed content by using these positions. Afterwards, the count of start tag and end tag are calculated whether the proposed content is correct or not. If the count of start and end tags are equal, this content is added to the result array. While implementing this algorithm, it is clear that the last control operation increases the response time in contents which have a lot of inner tags. Therefore, we examine the effects of the count of inner tags in the experiments.

4 Experiments

In experiments, test collection obtained from Milliyet (www.milliyet.com.tr) and Sabah (www.sabah.com.tr) contains 2000 web pages. 1000 pages are crawled for each newspaper. The total sizes of these collections are around 160.47 MB in UTF-8 format. Table 2 gives more information about this collection.

Table 2. Information about Test Collection

	Document Size		Tag Count	
Information	**Milliyet**	**Sabah**	**Milliyet**	**Sabah**
Storage Size of all Documents	56.32 MB	104.15 MB	-	-
Median a Document	57.67 KB	106.65 KB	1103.41	1320.07
Minimum a Document	30.04 KB	36.7 KB	587	496
Maximum a Document	112.66 KB	140.39 KB	3730	1998

This collection contains tags that are used to design web pages and contents that present between tags. A web document in this collection may take approximately 82.16 KB. A crude observation on the text collections reveals that text collection of Sabah contains more unnecessary tags and contents than Milliyet when considering that the sizes of actual contents are approximately similar. While doing experiments, we also take into account the effect of file size and the number of used tags. In the first analysis, we compare the parsing time of links and images for parsers of DOM and SET. The following table gives more information about this comparison.

Table 3. Information about the average time and the average count of extracted links and images

		Average Time (ms)		Average Count	
		Links	**Images**	**Links**	**Images**
DOM	**Milliyet**	214.49	355.54	248.84	55.42
	Sabah	741.48	1098.93	272.75	171.99
SET Parser	**Milliyet**	21.05	4.68	248.84	55.42
	Sabah	29.91	7.83	272.75	171.99

This analysis indicates that DOM Parser needs longer time to parse when compared with SET Parser which is used regular expressions. Moreover, the parsing time of a DOM object for Sabah is very slow when compared with Milliyet because of the file sizes and the count of tags used in web pages. The use of Regular expressions is more suitable solution for extracting links and images than the use of DOM. The analysis show that the pattern used for extracting links is slow than the pattern of extracting images in terms of time efficiency since patterns used for extraction have an effect on the results. The second analysis is on examining the parsing time of layout tags which have nested structure. We select three layout tags for each newspaper. These tags for respectively Milliyet and Sabah:

1. Main layouts: <div class="content"> and <div id="container_body">
2. Menu layouts: <div class="menu"> and <div id="altmenu" class="altmenu_style altmenu_font_style">
3. Related news Layouts: <div class="haber"> and <div class="pad10lrb print_hidden" style="">

In this analysis, the parsing time of DOM and SET parsers is examined for better understanding the improvements. (See below Table 4)

Table 4. Analysis of the Parsing Time in the Nested Structure

	DOM Parser		**SET Parser**			
Tags	**Milliyet (ms)**	**Sabah (ms)**	**Milliyet (ms)**	**Count of the Inner Tags**	**Sabah (ms)**	**Count of the Inner Tags**
1	639.68	1729.51	3.15	17.88	228.02	424.24
2	636.15	1690.21	2.50	3.00	37.47	176.68
3	617.21	1637.98	1.79	1.00	5.54	3.00
Avg.	**631.01**	**1685.90**	**2.48**	**7.29**	**90.34**	**201.31**

Table 4 indicates similar results when compared with Table 2. However, the count of inner tags increases the parsing time of SET Parser. For example; the count of the inner tags in *Tag 3 of Sabah* is approximately 3.00 and the parsing time is 5.54 milliseconds. Nevertheless, the count of the inner tags in *Tag 1 of Sabah* goes up to 424.24 and also the parsing time increases to 228.02 milliseconds. But still, the SET Parser is more suitable solution for obtaining contents between tags. The average parsing time of DOM Parser is 254.4 and 18.7 times shorter than the average parsing time of DOM parser, respectively in Milliyet and Sabah. Third analysis is examined the impact of the size of web pages on parsing time. For this analysis, the main layouts of Milliyet and Sabah are selected. The following figure indicates the parsing time of DOM parser for tag <div class="content"> of Milliyet and tag <div id="container_body"> of Sabah. (See Fig 1. (1))

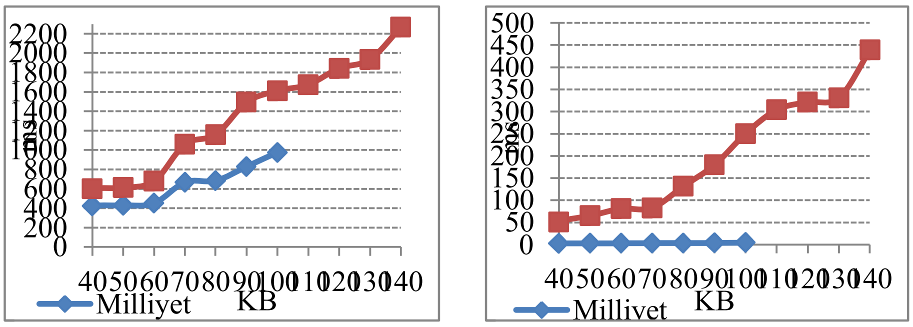


Fig 1. The parsing time for DOM Parser (1) and SET Parser (2) in different file sizes

The parsing time of Sabah is longer than the parsing time of Milliyet in similar file sizes because of the count of tags used in Sabah is more than the count of tags used in Milliyet. That is, the parsing time of DOM parser depends on the count of tags used in web pages. Moreover, Fig 1. (1) shows that the increasing size of web pages has a negative effect on parsing time.The Fig 1. (2) indicates the parsing time of SET Parser for same tags. The parsing time of Milliyet is lesser than the parsing time of Sabah. The parsing time of SET parser depends on the count of inner tags. In the other words, the Fig 1. (2) shows that Sabah contains more inner tags than Milliyet. The last analysis is for investigating the memory consumption of DOM Parser and SET Parser. (See below Table 5.)

Table 5. Analysis of Memory Consumption

	Average of File Sizes (KB)	**DOM Memory Usage (KB)**	**SET Memory Usage (KB)**	**Improvements (%)**
Milliyet	57.67	539.08	61.90	88.52
Sabah	106.66	1035.42	142.77	86.21
Avg.	82.16	787.25	102.34	87.00

DOM parser stores all contents and relations in memory. However, SET Parser only contains required part of contents and file size in memory. Therefore, SET Parser improves the memory usage with %87.00. The size of a DOM tree created from a document is as large as 9.58 times of the size of the original document. Wang et al. [10] claim that the size of a DOM tree crated from a document may be 10 times larger than the original size. However, this finding is not experimental but our study argues this claim for HTML DOM.

Conclusion

DOM tree constructed from a web page is widely used step for many content extraction algorithms. However, the use of DOM is not efficient on algorithms that are focused on specific tags. Because of DOM takes all tags into account in this task. In this study, the experiments show thatthe use of SET Parser is more suitable solution for extracting contents than the use of DOM. That is, a lightweight parser like the one in this study can be utilized for this task.

These analyze show that SET Parser was used for obtaining relevant contents of Turkish newspapers, namely Milliyet and Sabah. This method provides less parsing time and memory than the use of DOM. Some future research possibilities are adapting this parser to extract different web contents. Furthermore, the results of our search engine (SET) may be improved by using texts obtained from this parser.

References

1. Álvarez-Sabucedo L. M., Anido-Rifón L. E., Santos J. M. (2009). Reusing web contents: a DOM approach. Software: Practice and Experience 2009, 39(3): 299–314.
2. Kaasinen E., Aaltonen M., Kolari J., Melakoski S., Laakko T. (2000). Two Approaches to Bringing Internet Services to WAP Devices. WWW9; 231-246.
3. Wong W. and Fu A. W. (2000). Finding Structure and Characteristics of Web Documents for Classification. In ACM SIGMOD 2000; Dallas, TX., USA
4. Zheng Y., Cheng X., Chen K. (2008). Filtering noise in Web pages based on parsing tree. The Journal of China Universities of Posts and Telecommunications; 15, 46-50.
5. BuyukkoktenO., Garcia-MolinaH., Paepcke.A. (2001). "Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones". In Proc. of Conf. on Human Factors in Computing Systems (CHI'01)
6. Buyukkokten,O., Garcia-Molina, H., Paepcke.A. (2001). "Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices", In Proc. of 10th Int. World-Wide Web Conf..
7. Gupta S., Kaiser G. E., Peter G. (2005). Chiang M F, Starren J. Automating Content Extraction of HTML Documents. World Wide Web: Internet and Web Information Systems; 8, 179-224.
8. Cai D., Yu S., Wen J. R., Ma W. Y. (2003). Extracting content structure for web pages based on visual representation. APWeb'03 Proceedings of the 5th Asia-Pacific web conference on Web technologies and applications.

9. Yerlikaya T. and UZUN E. (2010) İnternet Sayfalarında Asıl İçeriği Gösterebilen Akıllı Bir Tarayıcı. ASYU-2010; 21-24 Haziran, Kayseri & Kapadokya, ISBN: 978-975-6478-60-8, 53-57
10. Wang, F., Li, J., Homayounfar, H. (2007). A space efficient XML DOM parser. Data Knowl. Eng.185-207

Biographies

Erdinç Uzun – received his BSc (2001), MSc (2003) and PhD (2007) degrees from Computer Engineering Department of Trakya University respectively. He was a research assistant in this university during this time. Then, he has begun to work in Computer Engineering Department of Namik Kemal University as assistant professor. He was the vice dean between 2007 and 2010 Faculty of Corlu Engineering. Since 2010, he is the vice chairman of Computer Engineering Department of this faculty. His research interests include information retrieval, machine learning, data mining, web mining and natural language processing. He is the developer of SET (Search Engine for Turkish - bilgmuh.nku.edu.tr/SET/) and Teaching (Lesson Content Management System - bilgmuh.nku.edu.tr/Teaching/)

Tarık Yerlikaya – received his BSc degree in Electronics & Communications Engineering Department of Yıldız Technical University (1999), MSc (2002) and PhD (2006) degrees in Computer Engineering Department of Trakya University. He was a research assistant between 1999 and 2006 in this university. Since 2006, he is the assistant professor in same university. Between 2006 and 2008, he was the head of the Division of Computer Hardware in this department. Since 2009, he is the chairman of Computer Technologies and Information Systems Department in Keşan Yusuf Çapraz School of Applied Sciences of Trakya University. His research interests are cryptology, security, web mining and intelligent systems.

Meltem Kurt – received her BSc degree in Computer Engineering Department of Trakya University in 2010. Now, she is the MSc student in same department. Her research interests are cryptology, web mining and information retrieval.