**©Journal of the Technical University Sofia, branch Plovdiv**
**"Fundamental Sciences and Applications", Vol. 16, 2011**
**International Conference Engineering, Technologies and Systems**
**TechSys 2011**
**BULGARIA**

# REDUCING COMPUTATIONAL COMPLEXITY BY RESTRICTING THE SIZE OF COMPARED WEB CONTENTS

ERDINC UZUN, TARIK YERLIKAYA, MELTEM KURT

***Abstract.*** *Extracting the relevant contents on web pages is an important issue for researches on information retrieval, data mining and natural language processing. In this issue, contents of tags in same domain web pages can be used to discover unnecessary contents. However, little changes in tag contents of web pages can cause problems in extraction. Therefore, we have adapted levenshtein distance algorithm to overcome these problems. Nevertheless, tag contents that may contain too many characters, have a negative impact on computational complexity. Hence, a solution, which reduces this complexity by comparing only a few characters, is proposed. In experiments, this solution gives a significant improvement (with 84.37%) in the performance of the use of levenshtein distance algorithm to find irrelevant contents.*

***Key words:*** *Reducing Complexity, Levenshtein Distance Algorithm, Parsing HTML*

## 1. Introduction

The web is an invaluable source of data for researches on subjects of information retrieval, natural language processing and data mining. However, web pages have contained unnecessary contents in the recent years. These contents can be filtered by determining whether contents of tags in web pages are necessary or not. Tags can be grouped by using tag properties such as id, attribute and style. Nonetheless, contents of these tags may be little changes between web pages. In this study, we ignore these little changes by utilizing levenshtein distance algorithm to this task.

The Levenshtein distance algorithm is widely used to determine how similar two strings are. Therefore, it can be utilized several different researches such as spell checking [1], speech recognition [2], DNA analysis [3] and plagiarism detection [4]. In this study, we adapt this algorithm to solve the problem of repetition in the contents of tags in web pages.

In general, extracting content of text in conventional techniques required knowledge on structure of the text. Web pages are semi-structured documents mostly written in HTML that defines tags. These tags can be parsed with different parsers such as DOM and SAX. Then, these parsed tags can be used for determining whether contents of tags are relevant or not. Web pages contain unnecessary contents such as advertisements, banners, navigation panels, news categories and comments of users [5, 6, 7, 8]. These contents also have negative effects in storing, searching and indexing. Therefore, we develop an intelligent crawler that is namely ICrawler. ICrawler is a module of SET (Search Engine for Turkish) project that has a search engine, an evaluator module and a stemming module. Classes of SET, are open source, are available via the web page http://bilgmuh.nku.edu.tr/SET/. This crawler automatically detects irrelevant contents and extracts relevant contents. We choose the layout tags (<div>, <table> and <ul>) that are mostly used in design of web pages. We also design a simple content extractor that utilizes regular expressions and string functions. However, the simple content extractor encounter with a problem in comparing contents of two tags. This study is described the solution about this comparison problem.

## 2. Structure of HTML

HTML(Hyper Text Markup Language) used for data sharing on the internet is the predominant markup language for web pages. HTML is written in the form of HTML elements consisting of tags, enclosed in angle brackets (like <div>), within the web page content. HTML tags normally come in pairs like <div> (opening tags) and </div> (closing tags). In the layout design of a web page, layout tags are commonly used <div> and <table>. Therefore, we have used these tags in filtering unnecessary contents. Table 1 indicates an example about the use of <div>.

**Table 1.** Example about tag of <div>

```
<div id="_top" class="header">
Main part of contents (starting…)
        <div class="head">
          Inner part of contents (1)
        </div>
        <div class="logo">
        Inner part of contents (2)
        </div>
 Main part of contents (end…)
 </div>
```

<div> tag has id, class and style features such as width, height, top, bottom and margin. In general, these attributes can be used to separate a tag from the others. In this study, we have utilized these attributes by grouping contents of layout tags. After grouping these tags, we compare the equality of the contents of layout tags for determining to extract unnecessary contents. However, little changes in html pages make it harder to compare these contents. Table 2. shows the little changes in web pages.

**Table 2.** Similar Parts from two web pages

```
<div class="menu">
<b>Main Menu</b><br/>
Publications<br/>
Links<br/>
About us<br/>
<div/>
```

```
<div class="menu">
Main Menu<br/>
<b>Publications</b><br/>
Links<br/>
About us <br/>
</div>
```

Normally, these two web pages are similar. But the contents of these two pages are not equal because of string "Main Menu" in web page 1 is taken bold tags (<b>…</b>), in web page 2 string "Publication" is bold. Hence, we compare the contents of two <div> tags with equality; the result is that two web pages are different. Instead of this method, we have used levenshtein distance algorithm to analyze the contents and decided to accept that the pages on a certain threshold value are the same. As expected, computational complexity has increased. In addition, we have observed that the contents of <div> tags have too many characters for comparison. The next aim is to reduce computational complexity and solve problem by providing less comparison between contents of layout tags. In this study, we also have tested the results of the methods and techniques are described below.

At this point, firstly we have compared by taking a certain number of characters from initial of contents. (See Algorithm 1)

Algorithm 1
Similarity control: starting from initial character to 200 characters

```
IF (length of string1 > 200)
string1 = string1.Substring (0, 200)
IF (length of string2 > 200)
string2 =string.Substring(0, 200)

IF (levenshtein (string1, string2) > 80%)
RETURN String1 is equal to String2
ELSE
RETURN String1 is not equal to String2
```

The substring() method, which is used in algorithm, extracts the characters from a string, between two specified indices, and returns the new sub string.
But we have seen that, initial parts of the web pages are very similar which make it appear like both are the same contents. Analysis shows that main difference between contents of tags is in the middle of them. Therefore, we have devised an algorithm for solving this problem. (See Algorithm 2)

Algorithm 2
Similarity control: starting from finding position to 200 characters

```
m = string1.Length;

IF (string2.Length > m)
m = string2.Length;

m = m / 2;
m = m - 100;

IF (length of string1 > 200)
String1 = string1.Substring(m, 200)
IF (length of string2 > 200)
String2 = string2.Substring(m, 200)

IF (levenshtein (string1, string2) > 80%)
RETURN String1 is equal to String2
ELSE
RETURN String1 is not equal to String2
```

Mid-point of the contents of tags is selected, and then a certain number of characters left and right of this point are taken into account to extract new strings. Our simple analysis indicates that this solution works efficiently on different web domains. In web pages, the actual content is usually viewed in the central part of HTML. Therefore, our solution argues that the use of central contents is appropriate for comparing substrings of tag contents.

## 3. Levenshtein Algorithm

The latest word processor programs are capable of suggesting a replacement for a mistyped word. Spelling checkers know how to evaluated distance between a misspelled word and the words in its files. Words whose evaluated distance is the smallest are offered as candidates for replacement.

Levenshtein distance is named after Vladimir Levenshtein, who considered this distance in 1965. It is useful in applications that need to determine how similar two strings are, such as spell checkers, speech recognition, DNA analysis and plagiarism detection. It's defined for strings of arbitrary length. It counts the differences between two strings (actually words, sentences), where we would count a difference not only when strings have different characters but also when one has a character whereas the other does not. Simply, this algorithm keeps the value of insertion operation, deletion operation and substitution operation.
For example; the Levenshtein distance between "running" and "sunday" is 5,since these five edits change one into the other, and there is no way to do it with fewer than five edits.

Table 2. En example for matrix that is used in Levenshtein distance algorithm

| | | r | u | n | n | i | n | g |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| s | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| u | 2 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| n | 3 | 3 | 2 | 1 | 2 | 3 | 4 | 5 |
| d | 4 | 4 | 3 | 2 | 2 | 3 | 4 | 5 |
| a | 5 | 5 | 4 | 3 | 3 | 3 | 4 | 5 |
| y | 6 | 6 | 5 | 4 | 4 | 4 | 4 | **5** |

1. mai**n**→mai**l**(substitution of 'n' with 'l' ,distance is 1)
2. dat**e**→dat**a** (substitution of 'e' with 'a' ,distance is 1)
3.film→film**y** (insert 'y' at the end ,distance is 1 )

While calculating the distance between two strings with this algorithm, matrix is created. Nevertheless, as the number of compared string increases, matrix size expands exponentially. When this algorithm is applied to similarity control of the contents of layout tags, processing time decreases because of exponential matrix size. Therefore, we have used certain number of characters for reducing matrix size.

## 4. Experiments

In experiments, 1000 pages are downloaded from popular Turkish daily newspaper that is namely Milliyet. The Figure 1 indicates that the count of obtaining contents from layout tags for some intervals.

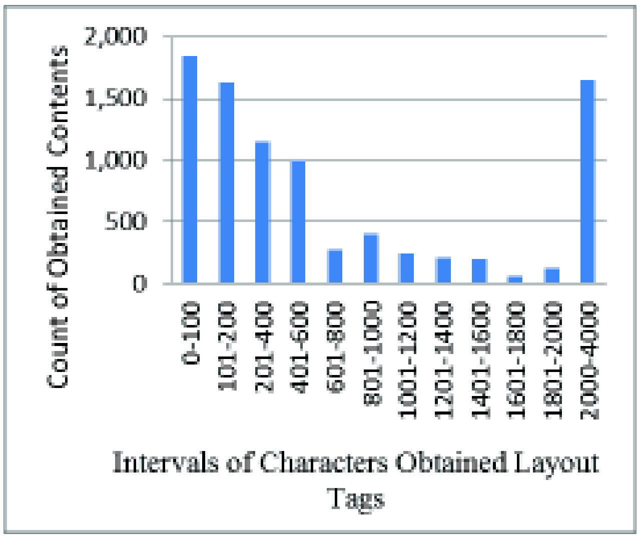

***Fig. 1.*** The Count of Obtained Contents from Layout Tags

For example, layout tags which contain fewer than 100 characters have 1,842 contents of tag. Our Crawler creates approximately 100x100 matrixes for these contents. Matrix size expands exponentially for different intervals so that the complexity and cost of system increase. This increasing has negative effects on system. Fig. 2. shows that there is an exponential growth with the increase in the number of characters (Normal Case).

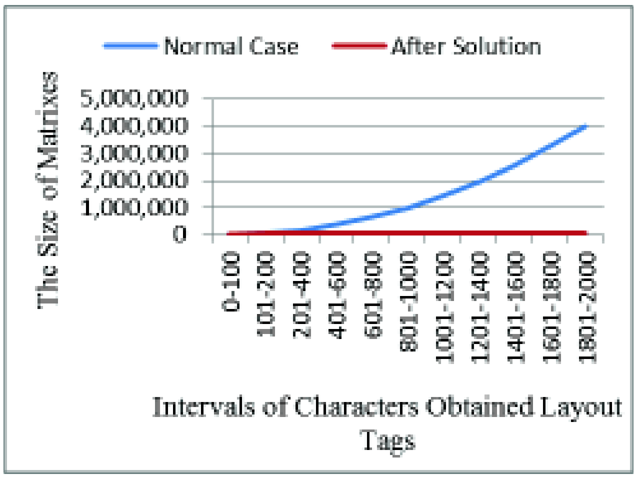

***Fig. 2.*** Expansion of Matrix Size

8,799 different contents are created for our testing data and 60.56% of these values are greater than 200. For instance, there are 1,000,000 comparisons for matrix size of 1000x1000. However, after applying our solution to Levenshtein Distance Algorithm, the number of comparison reduces to 40,000 for values that are greater than 200. In normal case, there are 10,078,600 comparisons for testing data of Milliyet, but applied by our solution has only 1,575,600 comparisons. That is, 84.37% improvements are obtained in complexity and costs by using our algorithm.

## 5. Conclusion

Matrix sizes are crucial issue for finding similarity of two data because of complexity and cost on researches such as spell checking, speech recognition, DNA analysis and plagiarism detection. Complexity and cost of algorithms can be reduced by using correct parts of data. In this study, we have used only the central parts of tags contents for maintain it. Because, according to experiments, changing parts of tags highly occur in the middle of them. Firstly, necessary contents have been found and then we have compared similarity between the contents of tags with adapting levenshtein distance algorithm. Thanks to our algorithm, not only matrix size decreased but also complexity reduced by comparing only a few characters. Therefore, using our algorithm, instead of using levenshtein algorithm to find unnecessary contents, provides (with 84.37%) improvement in the performance.

## References

**1. G. Navarro** A guided tour to approximate string matching. ACM Computing Surveys (CSUR) archive, 33(1), pp. 31-88, 2001.

**2. J. Fiscus, J. Ajot, N. Radde, C. Laprun** Multiple Dimension Levenshtein Edit Distance Calculations for Evaluating Automatic Speech Recognition Systems During Simultaneous Speech, Proceedings of Language Resources and Evaluation (LREC), Genoa, Italy, May 2006.

**3. X. Wu, F. Lü, B. Wang, J. Cheng** Analysis of DNA Sequence Pattern Using Probabilistic Neural Network Model, Journal of Research and Practice in Information Technology 37(4), 2005.

**4. Z. Su, B. Ahn, K. Eom, M. Kang, J. Kim, M. Kim** Plagiarism Detection Using the Levenshtein Distance and Smith-Waterman Algorithm, Proceedings of the 2008 3rd International Conference on Innovative Computing Information and Control, 2008.

**5. Yerlikaya T., Uzun E** İnternet Sayfalarında Asıl İçeriği Gösterebilen Akıllı Bir Tarayıcı. *Akıllı Sistemlerde Yenilikler ve Uygulamaları Sempozyumu (ASYU-2010) 2010*; 21-24 Haziran, Kayseri & Kapadokya, ISBN: 978-975-6478-60-8, 53-57.

**6. L. M. Álvarez-Sabucedo, L. E. Anido-Rifón, J. M. Santos** Reusing web con-tents: a DOM approach. Software: Practice and Experience 2009, 39(3): 299–314, 2009.

**7. Y. Zheng, X. Cheng, K.** Chen Filtering noise in Web pages based on parsing tree. The Journal of China Universities of Posts and Telecommunications; 15, 46-50.

**8. S. Gupta, G. E. Kaiser, G. Peter** Chiang M F, Starren J. Automating Content Extraction of HTML Documents. World Wide Web: Internet and Web Information Systems; 8, 179-224, 2005.

Department of Computer Engineering, Namik Kemal University, Corlu Engineering Faculty, Corlu / Tekirdag / Turkey
E-mail: erdincuzun@nku.edu.tr

Department of Computer Engineering, Trakya University, Ahmet Karadeniz Yerleskesi, Edirne / Turkey
E-mail: tarikyer@trakya.edu.tr
E-mail: meltemkurt@trakya.edu.tr