Extracting the Latent Hierarchical Structure of Web Documents

Michael A. El-Shayeb[Ψ], Samhaa R. El-Beltagy[Ψ] and Ahmed Rafea[λ]

[Ψ]Computer Science Department, Faculty of Computers and Information, Cairo University, 5 Tharwat Street, Orman, Giza 12613, Egypt
[λ]Computer Science Department, American University in Cairo, Cairo, Egypt

mikeazmy@yahoo.com, samhaa@computer.org, rafea@aucegypt.edu

Abstract. The hierarchical structure of a document plays an important role in understanding the relationships between its contents. However, such a structure is not always explicitly represented in web documents through available html hierarchical tags. Headings however, are usually differentiated from 'normal' text in a document in terms of presentation thus providing an implicit structure discernable by a human reader. As such, an important pre-processing step for applications that need to operate on the hierarchical level is to extract the implicitly represented hierarchal structure. In this paper, an algorithm for heading detection and heading level detection which makes use of various visual presentations is presented. Results of evaluating this algorithm are also reported.

Keywords: heading detection, heading level detection, document structure.

1 Introduction

The hierarchal or logical structure of a document plays an important role in many applications. For example, work presented in [1] exploits the hierarchal structure of a document to carry out anaphora resolution. In [2], the logical structure is used to segment a web document and perform passage retrieval. Other applications that can make use of a document's hierarchal structure include browsers designed for cell phones, PDAs and PCs with non-PC terminals as well as text summarization and data mining applications. However, the hierarchal structure of web documents is not always explicitly represented. Many web designers prefer to use their own styles to represent headings than to use the html heading tags meant to convey a document's logical structure. The work in this paper aims to overcome this limitation by presenting a heading detection algorithm and a heading level detection algorithm through which a document's hierarchical structure can be extracted.

The paper is organized as follows: section 2 presents related work, section 3 details the developed heading detection algorithms; experimental results are reported in section 4 and finally section 5 concludes the paper and presents future research directions.

2 Related Work

Extracting the hierarchal structure of a web document does not seem to be an area where much work has been carried out. In [3], an approach is presented for detecting headings in a document to solve the usability problem of browsing web pages designed for PCs with non-PC terminals but the approach does not address the determination of the levels of the detected headings. However, since detecting headings and heading levels can help in segmenting a document to smaller semantically independent units, work carried out in the area of web page segmentation becomes relevant.

A variety of approaches have been suggested for segmenting web documents. Most of these approaches try to make use of the nature of the web documents as a sequence of HTML tag delimited data elements. Many researchers have considered dividing the web document based on the type of the tags encountered in these documents. Examples of the most used tags include the <P> (paragraph), <TABLE> (table), <UL> (list), and <H1>~<H6> (heading) tags. In [4] a web query processing system uses such a tagged based approach to present to the user segments of documents that correspond to his/her query. The Vision-based Page Segmentation (VIPS) algorithm [5] uses spatial and visual cues to segment a web page in an effort to simulate the way a user mentally divides a page when presented to him/her based on his/her visual perception. It utilizes the facts that semantically related contents are often grouped together and that different regions are separated with implicit or explicit visual separators like images, lines, font sizes, blank areas, etc. The assumption of VIPS that semantically related contents are grouped together in a web page is not always true as sometimes semantically related segments may be distributed across a web page. In [6] a pre-trained Naïve Bayes classifier is used to help in the composition and decomposition of the semantically coherent segments resulting from the VIPS algorithm. In [7] an approach whereby template-based HTML web documents are transformed into their corresponding semantic partition trees is presented. This is achieved through the discovery of repeated sequential patterns on the sequence of HTML tags (with their attribute values) in a DOM Tree [8, 9].

Work presented in [9] argues that one of the best ways to view a web page on a small screen device is through thumbnail images that can aid the user to navigate the page by zooming in and out of regions of interest. Web page segmentation is an important part of achieving this effect and the work addresses it from a machine learning perspective as entropy reduction and decision tree learning are used to indicate divides within the page.

3 Heading Detection Algorithms

To infer the hierarchal structure of a document, identification of headings and their relationship with each other must take place. Two phases are employed for carrying out this task:

1. The heading detection phase (identification of headings)
2. The heading level detection phase (identification of relationships between headings)

In the first phase, all document text portions that are recognized as headings are collected and stored along with their features (font size, font weight, color, etc). In the second, a level for each of the identified headings is assigned. Each of these phases is described in details in the following two subsections. The third subsection, explains how results produced by these algorithms can be fine tuned.

3.1 The heading detection phase

The basic idea behind the presented approach is that headings are usually differentiated from other text in the document through the placement of some sort of emphasis on them. So, an important part of the heading detection algorithm is to scan the document whose structure is to be extracted to identify its general features (or what constitutes normal text within it). This step is important in order to understand the significance of various features within various documents as general features can differ from document to document. For example, in a document where most or all of the text is written in bold, having the feature of being bold, will not carry much weight, while it will in another document where the bold feature is rarely used. This step is referred to as the document's 'general feature extraction' step. Generated rules are then applied to text chunks that result from breaking down the document into blocks that share similar features or styles and which are easily detected by a DOM tree [8, 9]. An example of a DOM tree fragment is given in (Fig. 1). Whether or not a chunk/block is considered as a candidate heading is determined through considering whether or not it:

1. has a h1-h6 tag name
2. has font size > general document font size
3. has a font weight > general document font weight
4. has a font color not equal to the general document font color
5. has italic, bold, or underline styles that are not part of the document's general features
6. appears within some text
7. contains block-level elements
8. contains BR or HR tags
9. satisfies a maximum length threshold has words that are all capitalized

The basic algorithm for heading detection is provided in Fig. 2. The algorithm carries out its task through six main steps, two of which have just been described. First, the HTML of a web document is parsed to generate its HTML DOM tree (line 1). Second, different HTML tags and attributes that have the same visual effect are transformed to a standard representation for comparison. For example, the font weight can be represented by the <B> tag, <STRONG> tag or by using "font-weight = 700" attribute. Also the font size can be represented using different measures like pixels and points. So a transformation between the different tags attributes and measures that have the same effect is done by the transform-features function in line 2. Third, the document's general features (such as font size, weight, etc) are identified (line 3). To identify the general features of a document, the system first traverses the HTML DOM tree, and for each node in the DOM tree, it extracts the node's features, and de-

termines the total number of words that makeup the node's content. Features with the highest number of occurrence (based on the number of words) are chosen by the system to be general features.

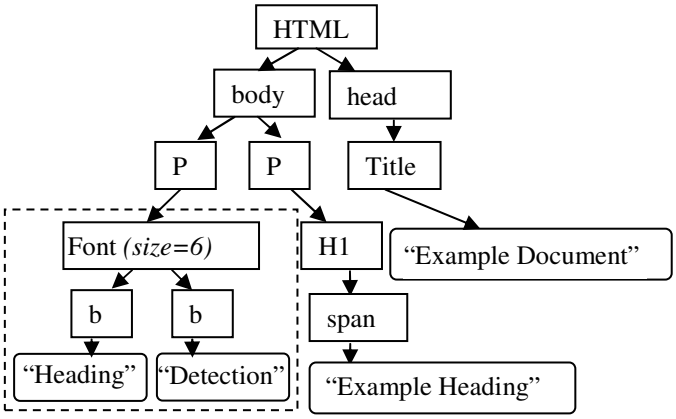

Fig. 1. Example of a DOM tree fragment

Fourth, heading detection rules are built based on the identified general features of the document (line 4).

Fifth, the generated heading detection rules are applied to the DOM tree nodes to generate a list of candidate headings. This task is sometimes complicated by the fact that a single heading can be split across multiple tags. For example in the DOM shown in Fig. 1 the heading "Heading Detection" is represented with the sub tree rooted at a font node (shown within the dashed rectangle in Figure 2).

```
PROCEDURE Detect_Candidate_Headings(Document Doc) {
  DOMTree = parse-html-doc (Doc)                          1
  transform-features(DOMTree)                             2
  docGenFetures=extract-doc-general-features(DOMTree)     3
  Rules = build-heading-rules(docGenFetures)              4
  Apply-Rules(DOMTree, Rules)                             5
}
PROCEDURE Apply-Rules (Tree DOMTree, RuleList rules)
  FOR EACH n ∈ DOMTree DO                                  6
    IF (match(n, Rules)) THEN {                            7
      headingContent = collect-heading-content(n)          8
      IF (validate-heading(headingContent) ) THEN          9
        docHeadings = docHeadings ∪ headingContent        10
      ELSE Apply-Rules(n, Rules)                          11
    }
}
```

Fig. 2. Pseudo-code for the heading detection algorithm

For this reason, rules are first applied to the root nodes of the DOM tree (lines 6-7). If a root node is found to be a candidate heading, then an attempt is made to collect its

content from its children nodes through the collect-heading-content function in line 8. If the root node does not qualify as a candidate heading, its children are then traversed in search for headings using the same rules (line 11). Finally, validation rules on the content of extracted candidate headings are applied and a final list of headings is generated. The validation step serves as a final filter for candidate headings. In this step rules such as those that check for a length threshold are applied. Also extracted candidate headings are validated for a manually entered list of text that can not be considered as a heading as part of results fine tuning detailed in section 3.3.

3.2 Heading Level Detection

After the detection of all possible headings in the document, the sequence of features of each heading is then used to detect its level. For example the sequence of features for the "Heading Detection" heading (shown in Fig. 1) is "<font size = 6>, <b>". A hierarchical tree is then built based on the relationship between heading levels. The main root of this tree is the title of the document. The nodes of this tree are the headings of the document. The children of a node are those headings with levels smaller than the heading level of a parent node. The algorithm for heading level detection is given in Fig. 3.

To detect heading levels, we assume that the first heading that occurs after the title of the document has the highest level or a level of 1 (line 1). A comparison is then made between each heading and its preceding heading (lines 4-6). If the features of the current heading are evaluated to be equal to the features of the previous heading, then the level of the current heading is assigned the same value as that of its preceding heading (lines 8-9). When comparing between the features of two headings, weight is given to the features in this order: font-size, font-weight, underlines, and finally italics.

```
PROCEDURE Heading-Level-Detection(HeadingsFeatures
Headings)
  /*first heading is the one with the highest level */
    Headings (0).Level = 1                              1
    CurHeading = 1                                      2
    For each curH ∈ Headings DO {                       3
      curH = Headings (CurHeading)                      4
      prevH = Headings(CurHeading -1)                   5
      Assign-Level(curH, prevH)                         6
      CurHeading = CurHeading +1                        7
    }
}
PROCEDURE Assign-Level (headingFeatures curH, heading-
Features prevH)
  IF curH.isEqualTo( prevH )THEN                        8
    curH.Level = prevH.Level                            9
  ELSEIF curH.isLessThan(prevH) THEN                   10
    curH.Level = prevH.Level + 1                       11
  ELSEIF curH.isGreaterThan(prevH )THEN   {            12
    /* if prevH is not the first heading in the heading
list */
      IF (prevH != 0)     THEN                         13
```

```
        Assign-Level(curH, prevH-1)                    14
    /* current heading does not have a parent, so it
will be assigned the highest level */
      ELSE curH.Level = 1                              15
  }
}
```

Fig. 3. Pseudo-code for the heading level detection algorithm

If however, the features of the current heading are less than the features of the previous heading, then the level of the current heading is smaller than the level of the previous heading (lines 10-11). Finally if the features of the current heading are greater than the features of the previous heading, then the parent of the current heading is searched for recursively (lines 13-14). If no parent can be found for this heading, then it is assigned a level of 1 (line 15).

3.3 Results Fine Tuning

Occasionally, web document authors assign headings to text portions just to achieve a certain effect without considering how this will affect the structure of the document. This is particularly common when the author of the document is not a professional web designer. For example, organization or document author names may sometimes appear as headings just because using a heading tag achieves the desired emphasis effect easily. Arbitrary text of certain importance may also sometimes be emphasized by denoting it as a heading. In such cases the results produced by the heading detection algorithm will not be 100% accurate. To detect errors like these, human intervention is required. So, an *option* is provided within a heading detection tool built around the presented algorithms, that enables users from viewing and editing extracted headings and heading levels. Users of the heading detection tool are also provided with the means for entering a list of terms that are often emphasized within documents as headings when they are not in fact that (like organization names). This list can be thought of as a heading stop list. Additionally, more advanced users are given access to dynamically generated heading detection rules, and are allowed to fine tune those.

4 Evaluation

In order to access the overall performance of the developed algorithm, it was applied to a dataset containing a total of 542 headings, of which only 5.3% was represented using html heading tags. Only documents that were unique in the way they represent headings were included in our data set as one of the main goals of carrying out this experiment was to find out how well the system can cope with different presentation styles for headings. As a result, a large number of documents collected from different sites were excluded, as the heading style employed in one document from those sites, was consistent across all documents collected from each. It is important to note that 299 of the headings used in this experiment were written by multiple authors none of which has any understanding of html. These documents were authored using Micro-

soft word or Front page with the main concern of users being achieving certain effects, which they did in a variety of inconsistent ways. It is also important to note, that in this experiment, no fine tuning was allowed to take place, which means that the results obtained are those of fully automated heading detection and heading level detection.

The standard information retrieval measures of precision and recall were used to evaluate the success of the algorithm in achieving its task hence forth, the total number of correctly extracted headings will be referred to as true positives (TP), while the total number of incorrectly extracted headings will be referred to as false positives (FP). Given these definitions, in the context of this work:

Precision = TP / TP+FP

Recall = TP/total number of headings in the document set.

The obtained results were also compared to an algorithm that we've devised based on VIPS[5]. The VIPS algorithm, which is a powerful and widely used segmentation algorithm, provides for each segment it extracts, a depth, which means that segments extracted by VIPS are represented in a kind of hierarchy. So, a layer was added on top of VIPS with the objective making this hierarchical representation explicit by assigning a level number to each extracted segment so as to compare the results with our algorithm and to see whether it would be better to go down the path of using VIPS. Basically, in the VIPS based algorithm, if a heading is represented in a segment on its own, then the algorithm is considered to have detected the heading correctly. Heading level detection is then calculated depending on how accurately segments are nested with respect to their parent headings. The results of applying our algorithm as well as the VIPS based one are shown in tables 1 and 2.

Table 1. Results for the heading detection task

	Total Headings	**Our Algorithm**				**An algorithm based on VIPS**			
		Recall	**Precision**	**TP**	**FP**	**Recall**	**Precision**	**TP**	**FP**
Total	542	92.8%	96.4%	503	19	55.5%	96.5%	301	11

Table 2. Results broken down by level for the heading *level* detection task

Heading level	**Total**	**Our Algorithm**				**An algorithm based on VIPS**			
		Recall	**Precision**	**TP**	**FP**	**Recall**	**Precision**	**TP**	**FP**
Level1	0	87.3%	76.2%	96	30	58.2%	64.7%	64	35
Level2	168	73.2%	69.1%	123	55	32.7%	55%	55	45
Level3	204	75%	81.4%	153	35	11.7%	28.9%	24	59
Level4	45	22.2%	33.3%	10	20	24.4%	42.7%	11	15
Level5	9	0%	0%	0	0	11.1%	25%	1	3
Level6	6	0%	0%	0	0	0%	0%	0	0
Total	542	70.5%	73.2%	382	140	28.6%	49.7%	155	157

In table 1 . (TP) or true positives indicate the number of correctly detected headings, while (FP) or false positives indicate the number of wrongly detected headings. In table 2, (TP) indicates the number of headings which were mapped to their levels

correctly while (FP) indicates the number of headings which were mapped to their levels inaccurately. These results seem to indicate that in most cases our proposed algorithm extracted headings accurately. They also show that the proposed algorithm outperforms the algorithm that we've based on VIPS. Cases in which the system falsely detected a heading were mainly due to the misuse of heading tags or of placing emphasis on text in a very similar way to representing a heading. Cases in which headings were not detected, were largely due to headings being represented as ordinary text in the document and cases when heading numbering was used as the only indicator of some text being a heading. The latter case is one which was not handled as focus was placed on presentation features of headings, but is also one that can be easily addressed by a simple improvement in the heading detection rules.

When analyzing the results of the heading level detection algorithm, it was found that undetected as well as wrongly detected headings from the heading detection phase seem to have a major effect in the process of level assignment. This seems logical, as an undetected or wrongly detected intermediate-level headings can propagate incorrect level assignments to detected headings that follow it.

5 Conclusion and Future work

This paper has presented a novel approach for transforming the implicitly represented hierarchal structure of a web document to an explicitly represented one using algorithms for heading detection and heading level detection. The algorithm presented is simple and straightforward, but the experiments conducted to evaluate this work show that it produces reasonable results. Future work will focus on trying to improve the results of the heading level detection algorithm, and the utilization of text segmentation and similarity techniques along side an ontology to assign headings to segments and infer a document's hierarchy when segment headings are not included explicitly or implicitly in a document.

Acknowledgments. Work presented in this paper has been supported by the Center of Excellence for Data Mining and Computer modeling within the Egyptian Ministry of Communication and Information (MCIT).

6 References

1. D. Goecke, and A. Witt. "Exploiting logical document structure for anaphora resolution", in Proceedings of LREC 2006 Conference, Genoa, Italy, 2006.
2. S.El-Beltagy, A. Rafea, and Y. Abdelhamid, "Using Dynamically Acquired Background Knowledge For Information Extraction And Intelligent Search". In Intelligent Agents for Data Mining and Information Retrieval, Ed. M. Mohammadian. Hershey, PA, USA: Idea Group Publishing, 2004, pp. 196-207.
3. Y. Tatsumi, T. Asahi, "Analyzing web page headings considering various presentations", In Proceedings of the 14th international conference on World Wide Web, Chiba, Japan, 2005.

4. Y. Diao, H. Lu, S. Chen, and Z. Tian, “Toward Learning Based Web Query Processing” In Proceedings of International Conference on Very Large Databases, Cairo, Egypt, 2000, pp. 317-328.
5. D. Cai, S. Yu, J.R. Wen, and W.Y. Ma, “Extracting content structure for web pages based on visual representation”, In Proceedings of the 5th Asia Pacific Web Conf, Xi'an, China, 2003.
6. R. Rupesh, Mehta, P. Mitra, and H. Karnick, “Extracting Semantic Structure of Web Documents Using Content and Visual Information”, In Proceedings of the 14th international conference on World Wide Web, Chiba, Japan, 2005.
7. S. Mukherjee, G. Yang, I. V. Ramakrishnan, “Automatic annotation of content-rich HTML documents: Structural and semantic analysis”, In Proceedings of the 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, USA, LNCS 2870,Springer-Verlag, 2003, pp.533-549.
8. HTML 4.01 Specification, http://www.w3.org/TR/REC-html40/
9. http://www.w3.org/DOM/
10. S. Baluja, “Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework”, in Proceedings of the 15th international conference on World Wide Web, Edinburgh, Scotland, 2006