

Extracting Logical Hierarchical Structure of HTML Documents Based on Headings

Tomohiro Manabe^{*}
Graduate School of Informatics
Kyoto University
Sakyo, Kyoto 606-8501 Japan
manabe@dl.kuis.kyoto-u.ac.jp

Keishi Tajima
Graduate School of Informatics
Kyoto University
Sakyo, Kyoto 606-8501 Japan
tajima@i.kyoto-u.ac.jp

ABSTRACT

We propose a method for extracting logical hierarchical structure of HTML documents. Because mark-up structure in HTML documents does not necessarily coincide with logical hierarchical structure, it is not trivial how to extract logical structure of HTML documents. Human readers, however, easily understand their logical structure. The key information used by them is headings in the documents. Human readers exploit the following properties of headings: (1) headings appear at the beginning of the corresponding blocks, (2) headings are given prominent visual styles, (3) headings of the same level share the same visual style, and (4) headings of higher levels are given more prominent visual styles. Our method also exploits these properties for extracting hierarchical headings and their associated blocks. Our experiment shows that our method outperforms existing methods. In addition, our method extracts not only hierarchical blocks but also their associated headings.

1. INTRODUCTION

Since the wide-spread of the Internet, a huge amount of data have been accumulated on the Web in the form of HTML documents. In order to take full advantage of this huge valuable asset, information extraction from these Web pages has become an important research topic. There has also been extensive research on querying, ranking, summarizing, and efficiently browsing these Web pages.

Because HTML documents are not plain text data, automatic understanding of the structure within them is important for improving these HTML processing tasks. There has been much research on the extraction of various types of structure in Web pages, such as list or table structure [3, 23, 27, 30] and layout structure consisting of a main body [16, 22, 28], side menus [19], and so on.

However, there remains one of the most prevalent types of structure in Web pages: *logical hierarchical structure within their main bodies*. Logical hierarchical structure is important for correctly understanding documents. For example, suppose we want to extract temporal information from some corpus, which includes a Web

page shown in Figure 1. If we ignore the hierarchical structure of this document and simply regard the document as a sequence of words, the line for July 2010 can be mistaken as describing events in July 2012 because the line is equally close to the word 2010 and 2012. For the same reason, the hierarchical structure of this document is also important when we rank this page for the query “2012 Jul construction”. These are examples of HTML processing tasks where recognition of hierarchical document structure is important.

However, there has not been sufficient research on the extraction of logical hierarchical structure of HTML documents. The problem may not seem difficult as HTML documents include explicit nested mark-ups. These mark-ups, however, often describe physical layouts or visual appearances of text data, and their nested structure does not necessarily coincide with logical hierarchical structure. For example, the mark-up structure of a HTML document consisting of sections and subsections usually has no hierarchical structure corresponding to the inclusion relationship between sections and subsections. It instead includes many nested tags corresponding to visual decoration of text fragments, and also includes even wrong or abused usage of tags. Because of this discrepancy between mark-up structure and logical hierarchical structure in HTML documents, it is not trivial how to automatically extract the latter. Extraction of logical hierarchical structure is thus a crucial yet non-trivial step commonly required in various HTML processing tasks.

On the other hand, human readers easily recognize logical hierarchical structure of these HTML documents. The key information used by human readers is hierarchical headings in the documents. For example, a search result of Google shown in Figure 2 includes three types of items: ordinary Web pages, images, and videos. Google search results may also include news, maps, recipes, Web page with sitelinks, and so on, and they have heterogeneous structures. Human readers, however, can parse such a completely heterogeneous list simply because the items in the list have only one common component: the headings. Human readers can also recognize the hierarchical structure of sections and subsections in the document in Figure 1 because of their hierarchical headings.

Based on this observation, we develop a method that extracts logical hierarchical structure of HTML documents by exploiting hierarchical headings in them. This approach is expected to work even if the document has many noisy or abused tags as long as it is appropriately designed by the author so that human readers can recognize its hierarchical structure by its headings.

However, identification of hierarchical headings in HTML documents is also a non-trivial task. According to our survey on our data set (which will be explained in Section 5.1), only less than 1/3 of headings are marked up with the proper HTML tags for headings, namely H1 to H6 and DT (definition term). The ratio may

^{*}Research Fellow of Japan Society for the Promotion of Science

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/3.0/>. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.

Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

Kyoto Aquarium
 Kyoto Aquarium is an aquarium in Kyoto, Japan.

Overview
 Kyoto Aquarium is one of the largest inland aquariums. It is exhibiting about 15,000 animals of about 250 species.

Information
 See also: [disclaimers](#)

Holidays
 Open throughout the year except for occasional holidays.

Opening Hours
 From 9 a.m. to 5 p.m. Reception closes at 4 p.m.

History
 See also: [History of Kyoto Aquarium](#) for details.

2010

- **Jul.** Construction started.

2012

- **Feb.** Construction finished.
- **Mar.** Opened just as planned.
- **Jul.** Welcomed the one-millionth visitor.

Figure 1: Example Web page with heading structure

depend on how we define “headings.” Our definition (which will be explained in Section 3) is relatively loose (i.e., defined in a wider sense), and if we adopt a tighter definition, the ratio may be higher than 1/3. Our loose definition is, however, preferred for our purpose, i.e., heading-based document segmentation, and as long as this usage of headings is concerned, the naive method based on tag names can extract only 1/3 of what we want.

In addition, H1–H6 or DT tags do not always represent headings. According to our same survey, only about 2/3 of these tags really represent headings even in our loose definition. There are many pages where these tags are used for marking up some metadata (e.g., author names and timestamps). There are also many pages where text nodes marked up with these tags are not visually prominent, which means they are not recognized as headings by human readers. We suspect the main reason of such usage of heading tags is search engine optimization (SEO); some search engines assign heavier weights to text between heading tags. There are also pages where heading tags are used to merely change visual styles.

Our method identifies hierarchical headings based on our assumptions on the visual design of headings. When users read a document with headings, they use each heading to determine whether its associated block is relevant to their needs. If they determine that the block is irrelevant, they skip it and jump to the next heading of the same level. If they determine that it is relevant, they look into it, and if it includes lower-level headings, they recursively repeat the same process in order to further narrow down blocks to read.

In order to help this process, the authors design headings in a special way: (1) they insert headings at the beginning of the corresponding blocks, (2) they give headings prominent visual styles, (3) headings of the same level are given the same visual style, and (4) headings at higher levels are given more prominent visual styles than headings at lower levels. We explain the details in Section 3.2.

In this paper, we propose a method that extracts hierarchical headings in HTML documents based on these properties of the headings, and extracts logical hierarchical structure based on the extracted headings. Our experiment shows that our method outperforms existing Web page segmentation methods.


In addition, our method can identify not only hierarchical blocks but also their headings. Heading extraction is important in its own right because they are important for understanding the information

About 597,000 results (0.37 seconds)

List of F5 and EF5 tornadoes - Wikipedia, the free ...
 en.wikipedia.org/wiki/List_of_F5_and_EF5_tornadoes - Wikipedia
 Among the most violent known meteorological events are tornadoes. Each year, more than 2,000 tornadoes occur worldwide, with the vast majority occurring in ...
 2013 El Reno tornado · 2011 Joplin tornado · 2013 Moore tornado · TORRO scale

Category:F5 tornadoes - Wikipedia, the free encyclopedia
 en.wikipedia.org/wiki/Category:F5_tornadoes - Wikipedia
 Wikimedia Commons has media related to F5 tornadoes. These tornado outbreaks had their strongest tornado rate as an F5 on the Fujita scale or an EF5 on the ...

Images for F5 tornado Report images



More images for F5 tornado

MASSIVE F5 TORNADO CAUGHT ON CAMERA! - YouTube
 www.youtube.com/watch?v=oaDmpcG0Ww0
 Apr 29, 2012 · Uploaded by ben turnford
 STARTING TODAY 50 MILLION FARMERS, HILLBILLIES, AND REDNECKS WILL FACE TORNAOES. I CALL

F5 Tuscaloosa tornado - YouTube
 www.youtube.com/watch?v=Tx26R6pCk
 Apr 29, 2011 · Uploaded by Ryne Chandler
 Nate Hughett and Ryne Chandler chasing the F5 tornado in Tuscaloosa AL. This storm was like nothing else ...

Figure 2: Example of heterogeneous search results by Google (<http://www.google.com>)

in their associated blocks. For example, we developed a block-based Web search system upon our block extraction method, and our experiment shows that we can significantly improve the ranking of blocks by complementing blocks with words in their ancestor headings. It is because words that have already appeared in some ancestor headings are often omitted in a block. The details are beyond the scope of this paper, and will be reported in another paper.

Heading extraction is also useful for document summarization and intra-document browsing support. For example, in the Accordion Summarization method [4], each block in a Web page is first represented only by its first line, and the whole block is shown upon the users’ click on it. With our heading extraction method, we can show its heading instead of its first line.

Our method relies on only a few properties specific to HTML, and it can easily be adapted to other types of marked-up documents as long as their headings are designed on the same principle. We use the following HTML-specific knowledge: the knowledge on which CSS properties affect visual styles of HTML nodes and the knowledge that IMG elements represent images.

The remainder of the paper is organized as follows. In the next section, we survey related work. In Section 3, we give our definitions of blocks and headings, and explain our assumptions on properties and structure of them. We then explain our structure extraction method in Section 4. In Section 5, we explain the result of our experiment. Finally, we conclude the paper in Section 6.

2. RELATED WORK

HTML5 has tags for marking-up hierarchical block structure. However, methods that rely on them cannot be applied to the huge asset of existing Web pages that are not in HTML5 format. In addition, given that the ratio of headings described by proper tags is not high, we cannot expect that most Web pages in future would be marked-up with proper HTML5 tags for hierarchical structure.

Hierarchical structure extraction is a kind of document segmentation. There have been several studies on Web page segmentation. Cai et al. [5] proposed a method called VIPS, which recursively divides a page mainly based on the margins between blocks in the rendered page. Their method works well for typical top-level layout structure of pages composed of main bodies, menu blocks, sidebars, and so on, but it does not work well for the hierarchical structure inside main text bodies. For example, it cannot detect blocks delimited by headings in bold fonts without any special margins.

Kohlschutter and Nejdil [20] proposed a method based on text-density. Their method can distinguish main bodies filled with full sentences from menu blocks including only short phrases, and also from advertisements mostly composed of images. Hattori et al. [17] proposed a method that divides a page at points where the HTML mark-up structure largely changes. Their method also works well only for the top-level layout structure of Web pages.

Chakrabarti et al. [7] formulated the page segmentation problem as an optimization problem on weighted graphs and showed how to solve the problem by learning weights from manually labeled data. Their method is also focused only on the top-level structure of pages, and it is not clear how to adapt their labeling and learning schemes to nested structure in the main bodies.

Debnath et al. [10] proposed a method of detecting main blocks in pages, but they focused on how to choose important blocks, and page segmentation is simply done by a predefined set of HTML tags; they first segment a page by TABLE tags then by TR tags, and so on. Lin and Ho [22] also proposed a method of estimating the importance of blocks based on tabular tags. Song et al. [28] used VIPS [5], explained above, for main block detection. Gupta et al. [16] also proposed a method of main block detection, and Keller and Nussbaumer [19] proposed a method of extracting side menus.

In summary, these page segmentation methods and main block detection methods focus on the top-level page layout, while we focus on structure inside the main body. These methods and our method are, therefore, complementary to each other. Another difference between these methods and ours is that our method extracts not only hierarchical blocks but also headings associated with them.

The SphereSearch engine [15] converts heterogeneous HTML documents into XML documents for unified ranking, and their conversion includes the detection of headings and their corresponding blocks based on simple heuristic rules. Their rules, however, depend on specific tags, such as H1 and B, and can work only for well-structured HTML documents described by these tags.

El-shayeb et al. [11] proposed a method that extracts hierarchical document structure by using the same information as ours: visually prominent headings. Their method, however, examines each node separately and determine if its visual style is likely to be a heading. They do not use information on other nodes sharing the same visual style, which is useful as explained in Section 5.4.5. Tatsumi and Asahi [29] proposed a method that detects headings based on their visual styles, but they also examine each node separately, and they do not extract hierarchical relationship among them.

There are also heading extraction methods based on machine learning [24, 26, 25]. These learning-based methods, however, can detect a heading only when the training data set includes positive examples of headings sharing the same features (e.g., visual style and partial tree structure). In [24] and [26], the evaluations were based on cross-validation, in which the training and test data sets may share headings in the same format from the same page. In [25], their method was evaluated on a data set consisting of top-ranked pages by Google, which must contain many pages that are from the same domain (e.g., Wikipedia) and have the same format.

There has been extensive research on the extraction of structured data, such as lists and tables, from Web pages [3, 23, 27, 30, 1] and from plain text [9]. All these methods exploit repetitive occurrences of tree patterns, tag sequences, word patterns, or visual features. This strategy, however, cannot extract a list of items with completely heterogeneous structures. As explained before, a result page of Google in Figure 2 includes items with heterogeneous structures. Three top-level sections of the document in Figure 1 also have heterogeneous structures, i.e., “Overview” has no internal structure, “Information” has two subsections, and “History” con-

tains a list of years “2010” and “2012”. Human readers can recognize these lists because they have only one common component: headings with the same visual style. Our method can extract such a heterogeneous list by detecting its homogeneous headings.

There have also been research on structure extraction from PDF documents. Gao et al. [14] has proposed a method of extracting hierarchical structure from the table of contents (ToC) of PDF documents. If a document does not have a ToC, they apply the same method to the list of headings extracted from the main body [13]. Their method, however, assumes that the level of a heading is either one-level lower, the same, or higher than the level of the immediately preceding heading. It holds for headings of sections, but does not hold for headings in a wider sense. For example, an enumerated list with headings appears at any level in section structure. If a document includes at least one such heading, their method fails.

Anjewierden [2] has also proposed a method of extracting logical structure from PDF documents. Their approach is to define domain-specific grammars for parsing documents. It is not applicable to a general Web corpus including diverse structure and design.

Chao and Fan [8] proposed a method of extracting text blocks from PDF documents by merging neighboring lines with similar styles. Their method, however, extracts disjoint blocks without hierarchical relationship.

3. LOGICAL DOCUMENT STRUCTURE

In this section, we give our definitions of blocks and headings. We then explain our assumptions on properties of headings, and we also discuss the correspondence between logical hierarchical structures and nested mark-up structure in HTML documents.

3.1 Definitions of Blocks and Headings

We first need to define blocks in order to clarify what we want to extract, but there is no consensus on what are logical blocks in documents. For example, a sentence and paragraph should or should not be a block depending on the application. Our main applications are information extraction and information retrieval. In these applications, we want to extract a block that has different topic from its neighboring blocks. In other words, we want to divide a document at points where the topic changes [6].

For example, in Figure 1, the lines for 2010 and those for 2012 have different topics: year 2010 and 2012. Therefore, they should be segmented into different blocks so that the lines for 2010 July would not be mistaken as describing 2012 July.

The definition above, however, is still ambiguous. Topics have hierarchical subtopics, and it is unclear what level of topic change it means. In order to make it less subjective, we only consider topic changes that are substantial enough that the author inserts a heading in order to explicitly describe a new topic. Based on this discussion, we define logical blocks in documents as follows.

DEFINITION 1. *Blocks:* A block is a coherent segment of a document that has its own heading describing its topic.

We then need to define what are headings. We define it as follows.

DEFINITION 2. *Headings:* A heading is a visually prominent segment of a document describing the topic of another segment.

We define headings as shown above because we made the following observations on blocks and headings in documents.

OBSERVATION 1. When a document includes a block with its own topic and the author inserts a heading describing the topic, the author makes the heading visually prominent so that (1) readers can easily distinguish the block from its neighbors, understand its

topic, and correctly interpret the information in it, and (2) readers can locate the information on that topic by scanning only headings.

For example, each block in the page shown in Figure 1 has its heading describing what topic of the block, and the user can locate relevant blocks by scanning only these headings.

Many pages have hierarchical topics. We also made the following observation on headings in such pages.

OBSERVATION 2. *When a page has hierarchical topics, its contents are organized into hierarchical blocks with hierarchical headings so that readers can locate information by a top-down recursive scan of its hierarchical headings, and can understand the topic of a block by its and its ancestors' headings.*

For example, suppose a user wants to find the information on the opening hours of the Kyoto Aquarium in the document in Figure 1. He/she first scans the top-level headings "Overview", "Information" and "History" then reads the block associated with "Information". Next, he/she scans the next-level headings "Holidays" and "Opening Hours" within the block then reads the block associated with "Opening Hours." As shown in this example, each heading corresponds to some significance level, and the scan starts from the most significant headings and proceed to lower ones.

We regard an entire document as a block with regarding the document title as its heading. Thus, all blocks in a document form a hierarchy rooted by a block corresponding to the entire document.

3.2 Properties of Headings

Based on the observations above, we assume that appropriately designed headings have the following properties. We will discuss the validity of these assumptions through the analysis of our HTML data set later in Section 5.4.5.

ASSUMPTION 1. Positions of Headings: *Each heading appears at the beginning of its corresponding block.*

We assume that page authors insert headings at the beginning of the corresponding blocks so that readers can start reading the corresponding block soon after finding a relevant heading.

ASSUMPTION 2. Visual Styles of Headings: *Headings and non-heading segments never have the same visual style.*

It is to help readers easily scan only headings without reading the other parts of a document. A visual style consists of various attributes, such as font size and font weight.

ASSUMPTION 3. Visual Styles of Heading Lists: *Two headings in hierarchical headings have the same visual style if and only if they are at the same significance level.*

It is to help readers easily jump from a heading to the next heading of the same level in each step in a recursive scan.

ASSUMPTION 4. Visual Styles of Hierarchical Headings: *Headings at higher levels are given more prominent visual styles than headings at lower levels.*

It is to help top-down recursive scan of hierarchical headings.

We call a sequence of headings with the same visual style (and therefore at the same significance level) a *heading list*. We also call a sequence of blocks corresponding to a heading list a *block list*. A block list in our definition has a broader sense than a list in prior research on list extraction. A block list may consist of heterogeneous items as long as their headings share the same visual style. For

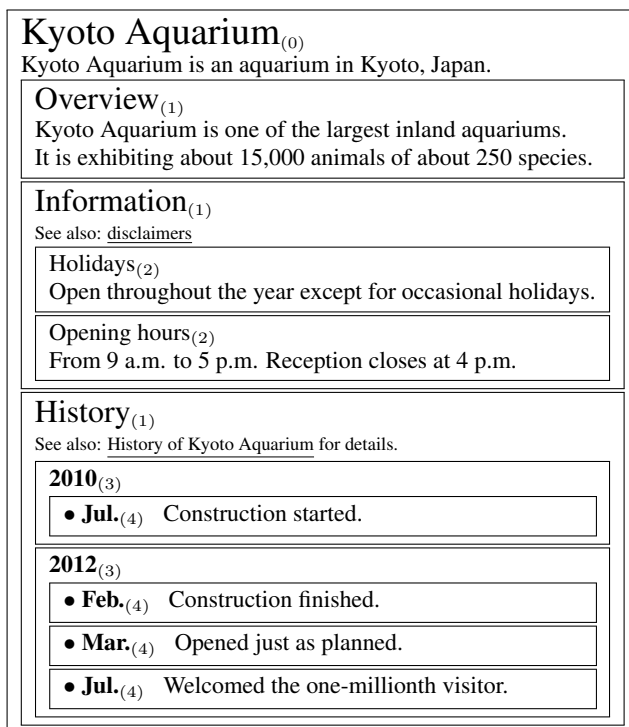


Figure 3: Heading structure of document in Figure 1. Subscripts (0) to (4) of headings corresponds to five heading lists.

example, items in the page in Figure 2, and the three top-level sections in the document in Figure 1 form block lists in our definition although they have completely different internal structures.

The hierarchical structure of the document in Figure 1 is shown in Figure 3. Each rectangle represents a block, and a subscript of a heading represents which heading list it belongs to. Note that a block list in our definition may be interleaved by other components. For example, four blocks corresponding to one event in 2010 and three events in 2012 are not siblings and are interleaved by the higher-level heading "2012," but they form a block list in our definition. Such an interleaved list is split into multiple sets of sibling blocks when we later segment the page into hierarchical blocks.

We define the logical *depth* of a heading by the depth of its associated block in the logical block hierarchy. Note that headings of the same significance level may not have the same logical depth. For example, a subsection occurring inside some section and a subsection occurring immediately under a chapter without a section have the same significance level but have different logical depths.

3.3 Correspondence between Logical Structure and Mark-up Structure

Nested mark-ups in a HTML document form a tree called a DOM tree, but it does not coincide with the logical hierarchy as mentioned before. We discuss correspondence between them below.

3.3.1 Headings and Nodes

We first discuss the correspondence between headings and DOM trees. Although HTML has tags for headings, our survey shows that only 1/3 of headings in our definition are marked up with these tags as explained before. The other headings are expressed by a text or IMG element with an explicit specification of the visual style. Because a heading only describes the topic of some block, a single text or image node is sufficient to express a heading in most cases.

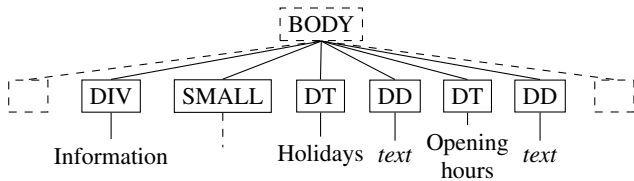


Figure 4: Node sequence representing “Information” block in HTML document in Figure 3. It consists of six sibling nodes inclusive of their descendants but not their parent node (BODY).

3.3.2 Blocks and Node Sequences

Next, we discuss how blocks are represented in a DOM tree. Because a block is a coherent segment, it corresponds to a contiguous segment in the HTML source text. In addition, it is very rare that a block only partially overlaps with a subtree in the DOM tree. This is because a block is a semantic coherent unit, while a block containing only the starting or ending tag of a tag pair is not a self-contained HTML fragment. On the other hand, a block also never partially overlaps with a text node because we cannot change the visual style in the middle of a text node, which means it is difficult to represent a boundary of a block in the middle of a text node. The only sub-structure in a tree structure satisfying the three conditions above is a *node sequence* consisting of adjoining sibling nodes (or consisting of a single node) inclusive of their descendants. Therefore, a block corresponds to a node sequence in the DOM tree.

Because a heading appears at the beginning of its block (Assumption 1) and is represented by a single text or image node, the heading of a block is represented either by the first node of the corresponding node sequence or by its descendant.

Figure 4 shows a node sequence corresponding to the “Information” block in the document in Figure 3. It consists of six sibling nodes, and its heading is represented by the text node “Information,” which is a descendant (child) of the first node of the sequence.

3.3.3 Nested Blocks and Block Lists

Lastly, we consider how nested blocks and block lists are represented in a DOM tree structure. When a block b_1 represented by a node sequence s_1 includes another block b_2 represented by a node sequence s_2 , s_2 is either a substring of s_1 or a substring of the children of a self-or-descendant of a member of s_1 .

A block list is represented by a set of node sequences that have headings of the same visual style (Assumption 3). Blocks in a block list may have different logical depths in the block hierarchy, as explained before. On the other hand, node sequences representing blocks in a block list usually have the same depth in the DOM tree. This is because they have headings with the same visual style, and the visual style of an HTML node is affected by the tag paths of the node, i.e., a sequence of tags along the path from the root to the node. To add the same visual style to all headings in a heading list, the author usually places them beneath the same tag path. If two blocks in a block list are placed at different depths in the DOM tree, the author cannot transpose them while maintaining the tag paths of their headings. Therefore, all node sequences representing blocks in a block list usually have the same depth in the DOM tree.

4. STRUCTURE EXTRACTION METHOD

In this section, we explain our structure extraction method, which we call HEPS (HEading-based Page Segmentation). HEPS simulates the behavior of human readers described in Observation 2

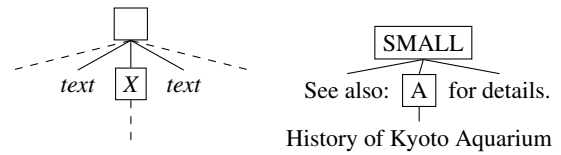


Figure 5: General structure of a sentence-breaking node X (left) and an example from the document in Figure 3 (right).

based on the assumptions explained in Section 3. It divides a document into nested blocks by recursively detecting and scanning headings in the order of their significance level. The method is composed of three steps except for preprocessing. First, it classifies DOM nodes in a given document into node sets according to their visual styles. Second, it tentatively assumes that every set represents a heading list, and sorts the sets in the order of their presumed significance level based on their position in the DOM tree and their visual prominence. Third, it determines a node set representing top-level headings and divides the document into blocks. It recursively repeats the last step for each obtained node set and extracts the hierarchical logical structure of the document.

4.1 Preprocessing

Before the main steps, we remove two kinds of nodes: blank nodes and sentence-braking nodes.

DOM trees of HTML documents may contain *blank nodes*, i.e., text nodes that only include whitespace characters (defined in Unicode 6.0). Because most blank nodes are used for indenting the source text and not for describing information, we remove all blank nodes from DOM trees before the other steps.

DOM trees also often contain subtrees corresponding to smaller structures than a “sentence”. For example, link anchor tags and emphasizing tags are often used for marking up only a part of a sentence. They produce nodes that divide a sentence into multiple text nodes. Such *sentence-breaking nodes* are rarely headings, but our method often misread them for headings, and their removal significantly improves the accuracy of our method (see Section 5.4.5).

We regard nodes satisfying the following two conditions illustrated in Figure 5 (left) as sentence-breaking nodes: (1) a sentence-breaking node X must be an internal node, and (2) it must appear between two sibling text nodes. The two conditions above, however, cannot detect sentence-breaking nodes at the beginning or end of a text. To detect such nodes, we use their visual styles. Because sentence-breaking nodes often have specific visual styles, if some nodes have the same visual style as some of the already-found sentence-breaking nodes, we also regard them as sentence-breaking nodes. The equivalence of visual styles is discussed in Section 4.2.

For example, the document in Figure 3 has two sentence-breaking nodes represented by small font size and underlines. The one in the “History” section satisfies the conditions above, as shown in Figure 5 (right). As a result, the other one of the same style in the “Information” section is also regarded as a sentence-breaking node.

We remove a sentence-breaking node by creating a text node from its contents and concatenating it with adjoining text nodes. We explain how to convert the contents into a string in Section 5.2.

4.2 Classification of Nodes by Visual Styles

In the first main step of HEPS, we classify nodes by their visual styles. As we discussed in Section 3.3.1, a heading is either a text node or an IMG element. Therefore, we consider them as *candidate-heading nodes* and classify them. To determine their visual styles, we use three types of information explained below.

Tag path: The tag path of a node is a sequence of tag names along the path from the root node to the node [23]. For example, /HTML/BODY/TABLE/TR/TD/UL/LI/text() is a tag path. We use tag paths to determine visual style because the visual style of a HTML node is affected by its and all the ancestor nodes' names. For example, a text node with the tag path above is rendered with an enclosing border by TD tags and a bullet attached by LI tags.

Computed style: In HTML, authors can also directly assign visual styles to nodes by using CSS. To include styles specified by CSS, we also use the *computed styles* of nodes calculated by Web browsers based on several factors, such as external style sheets and *style* attributes of the nodes. Computed styles contain many properties, but according to our survey, only a few of them are used for making headings prominent: *font-size*, *font-style* (to display in italics), *font-weight* (in bold), *text-decoration* (sidelines) and *color*. These properties are also effective for classifying image nodes because they specify the visual styles of their alternative texts.

Height of images: According to our experiments, height is also useful for the classification of image nodes because the content of an image used as a heading is usually characters, and the height of such an image reflects the font-size of the characters.

We consider that two nodes have the same visual style if they have the same tag path, the same values for the five CSS properties above, and the same height if they are images. Because of Assumption 2 and Assumption 3, each set produced by this classification is either a heading list defined in Section 3.2 or a set of nodes that are not headings. We call these node sets *candidate-heading lists*.

4.3 Sorting Candidate-Heading Lists

The first step above produces a set of candidate-heading lists. In the second step, we tentatively assume that all of them are true heading lists and sort them in their presumed significance level order. Those that are not heading lists will be removed later. We sort them based on three factors: depth in the DOM trees, visual styles, and the document order, in this priority order.

4.3.1 Sorting by Block Depth

To sort heading lists by their significance level, we first sort them by the depth of their associated blocks in the DOM tree. Note that all blocks in a block list must have the same depth (Section 3.3.3).

It corresponds to topological sorting by inclusion relationship among block lists. Because the depth of a block is smaller than or equal to the depth of its descendant blocks (Section 3.3.3), the sorting by the depth of associated blocks never contradicts the ancestor-descendant relationship between heading lists. On the other hand, if two heading lists have no ancestor-descendant relationship in a page, it makes no sense to sort them by the depth of their associated blocks, but the ordering between them is unimportant anyway.

We determine the depth of the associated blocks of a heading list by finding the least upper bound of two adjoining headings in the heading list. We compute it by the following procedure.

- Input:** a candidate-heading list $[e_1, \dots, e_n]$
- 1: if $n = 1$ return $[p_1]$ where p_1 is document root node
 - 2: $a := [e_1, \dots, e_n]$
 - 3: $p := [p_1, \dots, p_n]$ where p_i is parent node of e_i
 - 4: if $p_i = p_j$ for some $i \neq j$ return a else $a := p$
 - 5: jump to 3
- Output:** first nodes of blocks associated with $[e_1, \dots, e_n]$

We call each node in the output of this procedure the *front node* of the corresponding input heading node because it is the first (in document order, i.e., preordering in the DOM tree) node in the block associated with the input heading node. The depth of the front node is the depth of the associated block.

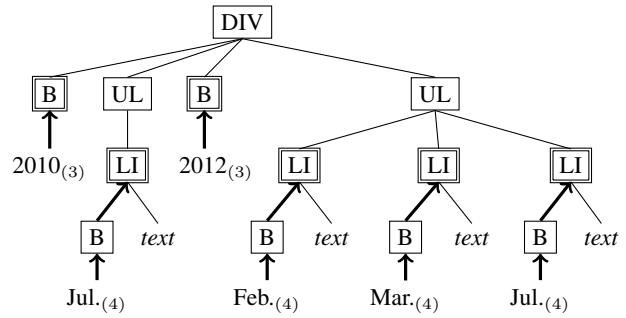


Figure 6: Partial DOM tree of document in Figure 3. This figure shows how we compute depth of blocks associated with heading lists (3) and (4). Arrows represent node replacement and double rectangles represent front nodes.

Figure 6 illustrates how this procedure works when the input is the heading list (3) or (4) in Figure 3. When the input is the heading list (3), two nodes “2010” and “2012” in the input are first replaced with their parent nodes, which are B nodes. In the second round, because their parents are the same DIV node, the procedure returns an array of the two B nodes. They are the first nodes of the blocks associated with the headings “2010” and “2012”, and their depth is the depth of the associated blocks. When the input is heading list (4), the four input nodes are replaced with B nodes in the first round then by LI nodes in the second round. In the third round, because some nodes share the same parent node, which is the UL node at right, the procedure returns an array of the four LI nodes. They are the first nodes of the blocks associated with the four input headings, and their depth is the depth of the associated blocks.

All headings in a heading list have the same depth in the DOM tree (Section 3.3.3). In each candidate-heading list produced in the first step, all elements have the same depth in the DOM tree because we used tag paths to classify nodes. Therefore, all front nodes in the output of this procedure also have the same depth. Therefore, we can sort heading lists by the depth of their front nodes.

The procedure above works only if there is a pair of adjoining blocks. When there is such a pair of blocks, they correspond to two adjoining node sequences in the DOM tree, and all top-most nodes in these sequences have the same parent node. Therefore, the procedure stops just before that parent node, i.e., at the top-most nodes of the blocks. We start from the headings, each of which must be the first node of the corresponding node sequence or its descendant (Section 3.3.2). Therefore, the procedure returns the first node in the node sequence representing the associated block.

However, when there is no pair of adjoining blocks, the procedure above fails to return correct front nodes. It happens when a candidate-heading list consists of only one node, and also when all the associated blocks of the headings are interleaved by other components. In the former case, we treat the document root as its front node for convenience, but such a list is anyway removed in the next step. The latter case is very rare, and we ignore it in this paper.

For each candidate-heading list produced in the first step, we compute the depth of the associated blocks by this procedure, and sort the lists in ascending order of the depth. When we have a tie, we break it based on their visual styles, as explained below.

4.3.2 Sorting by Visual Style

HTML has tags for specifying multi-level headings (H1, H2, and so on). Today’s popular Web browsers render upper headings with a larger font, or a bold font of the same size. We assume that au-

thors who directly specify the visual styles of headings follow this convention so that readers can understand the relationship between the headings. When we have two candidate-heading lists whose front nodes have the same depth, we sort them first in descending order of their *font-size* then in descending order of their *font-weight*. All headings in a candidate-heading list have the same value for these properties because we use them when we classify nodes.

If we still have a tie, we break it based on document order.

4.3.3 Sorting by Document Order

Because headings are placed at the beginning of the corresponding blocks, when we have multi-level headings, the first heading of higher significance level usually appears prior to the first heading of lower significance level. Based on this observation, we finally break ties by sorting the heading lists in the document order of the first element of each list. Because all nodes in a document have different positions in document order, we have no more ties.

4.4 Recursive Document Segmentation

Given a sorted list of candidate-heading lists produced in the second main step, we first segment the document into top-level blocks by using the first candidate-heading list then segment these blocks by using the next candidate-heading list. We segment the document into nested blocks by repeating this process, as human readers do.

In each step, we also determine whether the given candidate-heading list is really a heading list. To determine this, we first try to segment the given blocks by using the given candidate-heading list, and if it produces an inappropriate block structure, we determine that it is not really a heading list and skip it.

Because we use the document segmentation in that determining process, we first explain how we segment documents then how we determine whether a given list is really a heading list.

4.4.1 Segmentation Based on Headings

As explained in Section 4.3.1, a front node is the first node of a node sequence representing a block associated with a given heading. Therefore, we can extract the node sequence representing a block by starting from the front node, scanning its following siblings, then detecting the last node of the node sequence.

We use the following conditions for detecting the last node.

No following sibling: If there is no more following sibling, we stop the scan and regard the current node as the last node.

Another front node: Front nodes produced from one candidate-heading list represent the first nodes of blocks belonging to the same block list. These blocks in the same block list never include nor overlap each other. Therefore, if the following sibling is a front node of another heading in the current candidate-heading list, we stop the scan and regard the current node as the last node.

Node including already-found upper-level headings: Because a heading appears at the beginning of its block, and two blocks never share the same heading, a lower block never contains a heading of any upper block. Therefore, if the next following sibling includes any already-found upper-level headings, we regard the current node as the last node. Because we determine headings and segment a document in top-down manner, in each recursive step, all the upper-level headings have already been determined.

Node not included in current upper block: When we scan sibling nodes representing a block, its parent block has already been determined because of our top-down procedure. As discussed in Section 3.3.3, a node sequence of a block is contained by the node sequence of its parent block. Therefore, if the next sibling is not contained in the node sequence of the parent block of the current block, we regard the current node as the last node.

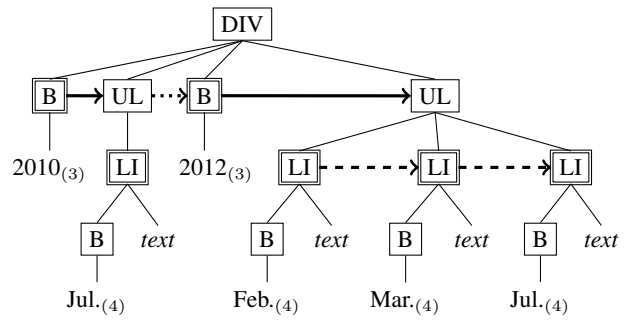


Figure 7: Detection of blocks starting from front nodes found in Figure 6. Scan proceeds along arrows and stop before dashed arrows. Double rectangles represent front nodes.

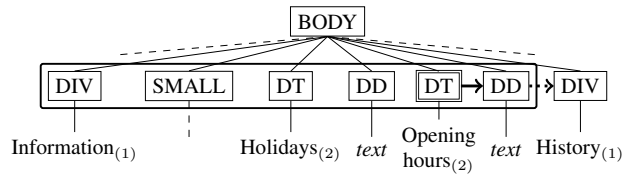


Figure 8: Scan starting from front node of heading “Opening hours” stops at next DD node because the following sibling is outside upper “Information” block, which corresponds to six sibling nodes enclosed by rectangle in this figure.

Figure 7 shows how we detect a node sequence representing a block starting at each front node found in Figure 6, which are represented by double rectangles. When we start from the left-most B node, we stop at the next UL node because the following sibling, i.e., the third B node from the left, is another front node for the same candidate-heading list. When we start from the left-most LI node, we stop immediately because there is no following sibling.

Figure 8 shows an example in which a scan stops at the last sibling node within the upper block. Six children of the BODY node, which are enclosed by a rectangle in this figure, correspond to the “Information” block in the document in Figure 3. When we start from the front node of the heading “Opening hours”, which is the DT node on the right, we stop at the next DD node because the following sibling is outside the upper “Information” block.

4.4.2 Determining Heading Lists

Next, we explain how we determine whether a given candidate-heading list is really a heading list. We use five conditions below for detecting non-heading nodes. For each condition, we compute the ratio of the number of nodes satisfying it to the number of nodes in the candidate-heading list, and if it is larger than a threshold θ for any condition, we determine that the list is not a heading list.

1. Node whose front node includes upper-level headings: If the front node of a candidate heading includes headings of upper blocks, the block produced from the front node would also include them. A block, however, never includes headings of upper blocks, as explained before. Therefore, if a candidate heading has such a front node, we determine that it is not really a heading.

2. Node producing empty block: Because a block must have a heading and a heading must have an associate block by their definitions, we do not consider a block including no text or image node except for its heading. Therefore, if we obtain such an empty block from a candidate heading, we determine that it is not a heading.

Table 1: Thresholds θ and t Optimized for Training Data Set

Condition	Threshold θ
1. Including upper-level headings	0.1
2. Producing empty block	0.2
3. No sibling candidates	0.7
4. Non-unique contents	0.6
5. Too much content as a heading	0.3 ($t = 1.5$)

3. Node without sibling candidates: We call blocks that share the same parent block *sibling blocks*. Similarly, we call nodes in the same already-found parent block and in the same candidate-heading list *sibling candidate headings*. Because headings are usually used to segment a block into multiple child blocks, it is rare that a block has no sibling block. Therefore, we consider that a candidate heading without a sibling candidate heading is not a heading. A candidate-heading list consisting of only one node, which was discussed in Section 4.3.1, is always eliminated by this condition.

4. Node with non-unique contents: Because a heading describes the topic of the associated block (Section 3.2), two sibling blocks rarely have headings with the same content. On the other hand, two blocks that are not siblings may have headings with the same content. For example, in the document in Figure 3, two blocks about July 2010 and July 2012 have the same heading “Jul.”. Therefore, we determine that a candidate heading is not a heading if there is another sibling candidate heading with the same content. The equivalence of content is determined after image-to-text conversion and space normalization, which is explained in Section 5.2.

5. Node with too much content: As a heading describes the topic of its associated block, if the *length of the associated block / length of a candidate heading* $< t$, in other words, if the candidate heading is too long compared with its associated block, we consider that the candidate heading is not a heading. The length of nodes are defined by the number of UTF-8 characters in them (after image-to-text conversion and space normalization, see Section 5.2).

We optimized the thresholds t and θ for our training data set (see Section 5.2) by a greedy algorithm. The result is shown in Table 1.

We first examine if the given list is eliminated by any of these conditions, and if it is not, we also separately remove each candidate heading in the list satisfying conditions 1 or 2. We did not use conditions 3, 4, 5 in this node-level filtering because node-level filtering with these conditions significantly degraded the recall ratio in our experiment with the training data set.

5. EVALUATION

In this section, we show the results of our experiments for evaluating HEPS. Because there is no well-known data set or evaluation measure for hierarchical structure extraction from documents, we created a data set by manually labeling blocks and headings in Web pages based on our definition explained in Section 3.1, and adopted standard precision and recall measures based on exact matching.

In both the manual labeling and the extraction experiment, we assume that we preprocess pages with some main body extraction method, and we only consider their main bodies as the target data. In our preliminary experiment, if we include headers, side bars, and so on in the target data, the accuracy of our structure extraction method is much higher. It is probably because structure in headers or side bars are easier to extract than structures in the main bodies.

5.1 Document Collection

As a standard sample of general Web pages, we used the well-known Web snapshot ClueWeb09 Category B document collection

Table 2: Assignment of Pages to Annotators and Data Sets

	A	B*	C	D	E	F	G	Total
Training set	101	100	93	96	26			416
Test set		200	100	103		200	200	803
Total	101	300	193	199	26	200	200	1,219

* One of authors.

(ClueWeb09B)¹. It was created by random crawling and includes many completely useless pages. In information extraction tasks and information retrieval tasks, the performance over useful pages are more important than the performance over all pages including such useless pages. To select pages meaningful for these applications, we randomly chose pages relevant to at least one query in TREC² 2009–2012 Web track ad-hoc tasks. There were 11,008 relevant documents without duplication of URLs. We then removed all Wikipedia articles, which share the same document structure and amount to 2,728 articles (about 25%) in the collection, in order to prevent their specific document structure from having too much influence on the overall evaluation. We also removed pages known to be redirecting pages³ and documents whose content type was not text/html. We finally obtained 8,013 documents.

Because ClueWeb09B contains no external files, such as external style sheets and images, which affect the visual styles of the pages, we tried to re-download the obtained 8,013 documents and their external files from Internet Archive⁴. To obtain a page data closest to that in ClueWeb09B, we selected snapshots closest to February 29, 2009; when ClueWeb09 crawling finished. We could download 7,187 documents. After that, we randomly chose 1,393 of the 7,187 documents due to limited annotation resource.

5.2 Ground Truth and its Representation

Annotation process: The annotation for creating the ground truth was done by seven participants including one of the authors. We first explained our definitions of headings, heading lists, blocks, and transitions (explained later) to the participants. We did not explain our assumptions in Section 3.2 in order to prevent bias in favor of our method. We also explained the definition of the *content body* of a page in prior research [16, 22, 28]: a content body is a *clearly specified coherent segment that necessarily and sufficiently contains the information the author must have most wanted to disseminate and which should differentiate the page from the others in the same Web site*. Each participant was asked to manually extract the content body from the randomly assigned pages and hand-annotate the hierarchical structure and the headings inside them with our original annotation tool.

To avoid over-fitting, we split the entire data set into a *training* data set and a *test* data set, and optimized the choice of conditions in Section 4.4.2 and their parameters in Table 1 with the training data set. The breakdown of the assignment of pages in each data set to annotators is shown in Table 2.

A block sometimes have more than one component that can be regarded as its heading. In most of such cases found during the annotation of the training data, a block is describing an entity, and both its name and picture can be regarded as its heading. In real applications, the users may regard either of them as the heading. To express such situations, we allowed the annotators to mark multiple

¹ClueWeb09 Wiki <http://boston.lti.cs.cmu.edu/clueWeb09/wiki/>

²Text REtrieval Conference Home Page <http://trec.nist.gov/>

³ClueWeb09 Wiki: Redirects <http://boston.lti.cs.cmu.edu/clueWeb09/wiki/tiki-index.php?page=Redirects>

⁴Internet Archive <https://archive.org/>

headings for a single block. During the annotation, we also found that the borders of headings and blocks are sometimes ambiguous. To express such cases, we allowed the annotators to mark some ranges as *transitions*, which means that the components may or may not be a part of the heading or block. For example, a horizontal line between two blocks is a transition for both blocks.

Exceptions: During the annotation, we removed 174 pages (12.5% of 1,393 pages) from the answer set because their structures were difficult to extract even for human users. The most significant reasons were redirecting page (2.6%) and too much content (2.5%). If a page needed more than 15 minutes for annotation, we removed it as a page with too much content to give priority to the number of annotated pages. We also removed a few dynamic pages (0.4%) because annotating dynamic content is difficult. Note that HEPS itself can be applied to such dynamic pages at any point in time.

Data set statistics: Finally, we obtained annotations for 1,219 pages from 981 domains, which constitute our data set. The maximum number of pages from the same domain was only 9 pages, and 841 pages (69.0% of 1,219 pages) had no other pages from the same domain, which means the data set is sufficiently diversified. Among them, 953 pages (78.2%) contained at least one block other than the entire page, and 423 pages (34.7%) contained hierarchical structures, i.e., at least one parent-child pair of blocks. This result shows that both flat and hierarchical block structures are common in general Web pages. The total number of blocks was 15,560, and 1,194 blocks (7.7%) contained two or more headings. The total number of headings was 16,817. Transitions were assigned to only 171 headings (1.0%) and 124 blocks (0.8%). The medians of the number of blocks were 6 per page and 3 per list.

Data representation: To represent the annotation results, we defined the *raw string* of a page. It is obtained by joining all the text nodes on the page in their document order. To simplify the discussion, we handle an IMG element as a special text node containing the element’s *src* (source) attribute value. We then replace adjoining whitespace characters (defined in Unicode 6.0) with a single space and remove leading or trailing spaces. We ignore the content of SCRIPT and STYLE elements because they are not rendered on browsers. We represent each heading or block as a set of ranges in the raw string. We use a set of ranges instead of a range because annotators sometimes mark some region that is not contiguous in the raw string (e.g., a column of a table) as a heading or a block.

5.3 Evaluation Measures

In our evaluation, an extracted heading or block *matches* an annotated heading or block if and only if they are exactly the same set of ranges except for those marked as transitions in the annotation. However, if an extracted heading matches a heading of a block that has multiple annotated headings, no other extracted heading matches the remaining annotated headings of the same block.

By this definition of match, we can adopt the standard measures: precision and recall. For a page from which at least one heading and block pair was extracted, the precision of heading extraction and block extraction, denoted by P_H and P_B , are defined as follows:

$$P_H = |\text{correctly extracted headings}| / |\text{all extracted headings}|$$

$$P_B = |\text{correctly extracted blocks}| / |\text{all extracted blocks}|.$$

Similarly, for a page containing at least one annotated pair of a heading and a block, the recall of heading extraction and block extraction, R_H and R_B , are defined as follows:

$$R_H = |\text{correctly extracted headings}| / |\text{all annotated headings}|$$

$$R_B = |\text{correctly extracted blocks}| / |\text{all annotated blocks}|.$$

Table 3: Accuracy of HEPS for Training/Test Data Sets

Data set	P_H	R_H	F_H	P_B	R_B	F_B
Training	.691	.564	.621	.617	.539	.575
Test	.638	.569	.602	.586	.563	.574
Total	.656	.567	.608	.596	.555	.575

Although HEPS first extracts headings then extracts blocks based on the extracted headings, P_B and R_B may exceed P_H and R_H , respectively, because HEPS may correctly extract blocks based on incorrectly extracted headings.

In the calculation of precision and recall, we did not count the entire content body as a block because its range is given. When we count the number of all annotated headings, we count multiple headings of a block as one heading.

Precision/recall over a data set are calculated by taking arithmetic means of precision/recall for each page (excluding pages with no extracted or annotated ones for the computation of precision or recall, respectively). We also calculate balanced F-measures $F_H = 2P_H R_H / (P_H + R_H)$ and $F_B = 2P_B R_B / (P_B + R_B)$ to integrate these two measures.

5.4 Experimental Results

In this section, we explain the result of our experiments.

5.4.1 Inter-annotator agreement

First, we discuss inter-annotator agreement because each page is labeled by only one annotator. We calculated Fleiss’ Kappa coefficient [12] for five annotators of our test data set. For this purpose, we collected another *agreement* data set containing 102 pages by the same process as the training and test data sets explained before. From these pages, annotator B extracted content bodies then the five annotators labeled headings and blocks in the bodies.

We calculate Kappa coefficient by considering that annotators make binary judgments whether each candidate is a heading (or a block). Because it is impractical to count all the sets of ranges that can be labeled by our annotation system as the candidates, we only count text and IMG nodes as candidate headings, and only count block-level elements of HTML 4.0 as candidate blocks. They covered 73.1–95.8% (depending on the annotator) of annotated headings and 48.6–61.0% of annotated blocks in the agreement data set.

The arithmetic mean of Fleiss’ Kappa coefficient computed for each page in this setting was 0.693 for heading annotation and 0.583 for block annotation. The former can be regarded as showing a good [12] and substantial [21] agreement, and the latter can be regarded as showing a fair to good agreement [12] or a moderate to substantial [21] agreement. This fact supports validity of our problem definition and reliability of our data sets.

5.4.2 Over-Fitting to Training Data

Because the conditions in Section 4.4.2 and their threshold values in Table 1 are chosen based on the experiment with the training data set, we compared the results of HEPS for the training data set and for the test data set to see if over-fitting occurred. Table 3 lists the results. P_H most significantly degraded (-0.053) for the test data set, while F_B remained the same. These results mean minor over-fitting occurred especially in heading extraction.

5.4.3 Influence of Annotators

We also analyzed the influence of the annotators. Table 4 lists the performance of HEPS for the data subset corresponding to each annotator. Although annotator B is one of the authors, the heading

Table 4: Accuracy of HEPS for Test Set by Each Annotator

Annotator	pp.	P_H	R_H	F_H	P_B	R_B	F_B
B *	200	.692	.610	.648	.651	.603	.626
C	100	.616	.582	.599	.594	.589	.591
D	103	.556	.486	.519	.484	.468	.476
F	200	.573	.527	.549	.565	.548	.556
G	200	.705	.608	.653	.592	.579	.585

* One of the authors.

Table 5: Comparison with Naive Method and Existing Methods

Method	P_H	R_H	F_H	P_B	R_B	F_B
Decision tree [24]	.084	.884	.154			
Hn & DT	.668	.320	.433			
VIPS [5]				.215	.070	.106
HEPS	.638	.569	.602	.586	.563	.574

extraction by HEPS worked best for the pages annotated by annotator G (+0.005 in F_H followed by B). This proved that HEPS does not over-fit to data by a specific annotator, which means that its effectiveness does not depend on readers. For block extraction, HEPS worked best for the pages annotated by B, but the difference was small (+0.035 in F_B , followed by C).

5.4.4 Comparison with Existing Methods

Next, we compare HEPS with some baselines. We first compare the performance of heading extraction. We chose the decision tree method [24] as the first baseline because its F_H (0.852) is better than F_H (0.851) achieved by Sano et al.’s method [26] according to the experiment reported in [24, 26], and detailed sets of rules or features for the other methods [25, 29] are not available. Following the description in [24], we constructed a decision tree by using the J4.8 algorithm of the data mining tool Weka⁵, which is exactly the implementation used in [24]. The J4.8 algorithm has several parameters, e.g., the threshold for pruning, but the parameters used in their experiment are not described in [24], so we used default parameters. We trained it using all 5,379 headings and randomly chosen 5,379 non-heading nodes in our training data set. Its F_H by 10-fold cross validation with the training data set was 0.865, which is close to 0.852 reported in [24]. This suggests that we reproduced their results quite successfully. However, the decision tree trained by the training data set worked poorly for the test data set, as shown in Table 5. It was the best result of 10 attempts. This is probably because our data set is diversified, while their data set was not. As explained in Section 2, the evaluations in [24] were based on cross-validation, in which the training and test data sets may share headings in the same format from the same page.

We also tested a naive heading extraction method, *Hn & DT*, which extracts elements with the proper HTML tags for headings, namely H1 to H6 and DT. The R_H value of the Hn & DT method shown in Table 5 shows that only less than 1/3 of headings were marked up with these proper tags, as explained in Section 1. Our result also showed low P_H : around 1/3 of nodes marked up by them are not headings. To determine the reason, we inspected 100 pages with the worst P_H . In 45 of those 100 pages, some (or all) of H1–H6 and DT nodes mark up some metadata (e.g., author names and timestamps). In 33 pages, most nodes marked up with H1–H6 and DT were not so visually prominent and the annotators overlooked them, which means those nodes do not actually work as headings

⁵Weka 3: Data Mining Software in Java
<http://www.cs.waikato.ac.nz/ml/weka/>

Table 6: Accuracy of Naive Methods and HEPS Variations

Method	P_H	R_H	F_H	P_B	R_B	F_B
HEPS	.638	.569	.602	.586	.563	.574
(a) Extraction of candidate heading nodes						
All text nodes	.118	.871	.208			
All IMG nodes	.110	.086	.097			
All text and IMG nodes	.108	.896	.193			
All HTML nodes	.026	.928	.051			
(b) Removal of sentence-breaking nodes						
Leave breaking nodes	.492	.578	.532	.453	.570	.505
(c) Classification of nodes by visual styles						
Without tag path	.647	.548	.593	.591	.541	.565
Without computed style	.612	.539	.573	.562	.532	.547
Without image height	.630	.571	.599	.579	.564	.571
(d) Sorting candidate heading lists						
Only by document order	.635	.568	.600	.566	.555	.560
(e) Determining actual heading lists						
Without condition 1.	.636	.573	.603	.581	.565	.573
2.	.638	.582	.609	.583	.577	.580
3.	.545	.563	.554	.361	.551	.436
4.	.597	.562	.579	.561	.561	.561
5.	.606	.574	.590	.550	.567	.558

for readers. As explained in Section 1, we suspect that such usage of heading tags is related to search engine optimization (SEO).

The P_H of HEPS was only slightly worse (-0.030) than that of the Hn & DT method, which is quite satisfactory, given that the Hn & DT method is focused only on *explicit* headings. Moreover, HEPS achieved a far better R_H (+0.249) than the Hn & DT method. These results show that HEPS can extract many implicit headings while maintaining satisfactory precision.

We next evaluate the performance of block extraction by HEPS. We compared HEPS with VIPS [5]. VIPS is focused on a top-level page structure while HEPS is focused on a hierarchical structure in the main body as explained in Section 2. Nonetheless, we chose VIPS because there is no available method focused on a hierarchical structure in the main body, and for extraction of top-level page structure, VIPS [5] is most widely used [18, 19, 28]. The implementation of VIPS is available at Deng Cai’s Web site⁶.

VIPS has a parameter called degree of coherence (DoC) that defines the granularity of blocks at which VIPS stops recursive segmentation. The larger DoC is, the deeper level VIPS proceeds to. When measuring P_B , we used the DoC value that achieves the best P_B value for each page. (However, we did not allow too low DoC values that make VIPS produce no block for pages that actually have blocks. Otherwise, we can inflate average P_B of VIPS by eliminating all pages from the computation of average P_B except for the page with the highest P_B .) On the other hand, when measuring R_B , we used the maximum DoC value, which makes VIPS extract as many blocks as possible.

The result summarized in Table 5 shows that HEPS achieved far better P_B (+0.371) and R_B (+0.493) than VIPS. It proves that hierarchical heading structures are far different from top-level layout structures targeted with the many existing methods. HEPS and existing methods are, therefore, complementary to each other.

5.4.5 Effects of Design Decisions and Assumptions

We next measure the effects of various design decisions in HEPS. **Heading candidate extraction:** We assumed that a heading is either a text node or an IMG node. If we extract all of them, R_H is 0.896 (Table 6(a)). It is 0.928 (only +0.032) if we extract all HTML nodes in the page, as in Okada and Arakawa [24] or Tatsumi and

⁶VIPS: a VISION based Page Segmentation Algorithm
<http://www.cad.zju.edu.cn/home/dengcai/VIPS/VIPS.html>

Table 7: Accuracy of Relationship Extraction with HEPS

Relationship	P_H	R_H	F_H	P_B	R_B	F_B
Baseline	.638	.569	.602	.586	.563	.574
Ancestors	.810 (+26.9%)	.582 (+2.28%)	.678 (+12.6%)	.542 (-7.51%)	.431 (-23.4%)	.480 (-19.5%)
Parent	.808 (+26.6%)	.580 (+1.93%)	.675 (+12.2%)	.542 (-7.51%)	.431 (-23.4%)	.480 (-19.7%)
Siblings	.879 (+37.7%)	.813 (+42.8%)	.845 (+40.4%)	.757 (+29.1%)	.719 (+27.7%)	.737 (+28.4%)
Children	.489 (-23.3%)	.582 (+2.28%)	.532 (-13.2%)	.504 (-13.9%)	.563 (+0.00%)	.532 (-7.95%)
Descendants	.476 (-25.3%)	.585 (+2.81%)	.525 (-14.6%)	.504 (-13.9%)	.567 (+0.71%)	.534 (-7.56%)

Asahi [29]. These results show that our strategy of simply choosing text and IMG nodes as candidates is reasonable. Table 6(a) also shows R_H values for all text nodes and all IMG nodes. Note that their sum is not equal to R_H for all text and IMG nodes because there are blocks that have both a text heading and a IMG heading.

Removal of Sentence-breaking nodes: If we do not remove sentence-breaking nodes, P_H and R_H with HEPS become 0.492 (-0.146) and 0.578 (only +0.009), respectively (Table 6(b)). It means that most sentence-breaking nodes are not headings and that HEPS sometimes misread them for headings if we do not remove them.

Information on visual styles: To see if the three types of information used in candidate heading classification are effective, we ran HEPS without each of them. The F_H is 0.593 (-0.009) without tag paths (but, instead, with the length of tag paths), 0.573 (-0.029) without computed styles, and 0.599 (-0.003) without image height (Table 6(c)). These results show that all these types of information are more or less useful for the classification.

Sorting candidate-heading lists: Even when we sort candidate-heading lists only by their document order, F_H dropped by only .0002 (Table 6d). There are two purposes to sorting of candidate lists: (1) processing the correct ones earlier and (2) processing the upper-level headings earlier and the lower-level headings later. If the step filtering out non-heading lists works well, only purpose (2) is important, and the document order may be sufficient for that.

Segmentation based on headings: The precision P_B of blocks segmented by correctly extracted headings is 0.769. This supports the validity of our idea of segmenting a document based on its headings, and implies that we can improve the precision of block extraction by improving the precision of heading extraction.

Determining heading lists: We measured the effects of five conditions for filtering out non-heading lists discussed in Section 4.4.2. Table 6(e) shows the performance of HEPS without set-level filtering with each condition. The values of F_H increased for conditions 1 (including upper-level headings) and 2 (producing empty block). We adopted these two because they were useful for the training data set, but they may not be useful in general. On the other hand, F_H degrades when we remove condition 3, 4, and 5.

We also evaluate the validity of our assumptions in Section 3.1, on which our method relies, by using all the true blocks and their headings in both the training and test data set (1,219 pages).

Positions of Headings: In 94.6% of true blocks, their headings appear at the beginning of the blocks. It validates our assumption.

Visual Styles of Headings: For each page, we computed the ratio of headings that have no non-heading nodes with the same visual style within the page. The ratio averaged over 1,129 pages was 73.6%, which is lower than expected. We examined the error cases, and found that this assumption itself is valid even in most error cases, but the set of visual features we used is not enough, and could not distinguish some headings from non-heading nodes. For example, some headings are distinguished from other texts by centering horizontally, or by using some prominent markers, but our method cannot recognize all of them. We should include more

visual features in our method, but some of them are not easy, e.g., the detection of horizontally centered text in two-column pages.

Visual Styles of Heading Lists: We also computed the ratio of true heading lists whose all members have the same visual style. Its average over 1,129 pages was 76.4%, which is also lower than expected. According to our error analysis, this is because our definition of equivalence between visual styles is too strict. For example, headings in the same list sometimes have slightly different font size (e.g., smaller fonts for longer headings). It suggests that some kind of clustering techniques may improve our method.

5.4.6 Accuracy of Hierarchical Structure Extraction

Finally, we measured the accuracy of hierarchical structure extraction with HEPS. Although tree edit distance is the well-known measure for comparing hierarchical structure, the edit distance is not intuitive. We instead measured the accuracy by precision/recall of the extracted relationships. To calculate them, we collect all instances of the relationship between a correctly extracted heading/block and another true or extracted headings/blocks, then calculate precision/recall of the extracted ones. For example, to measure the accuracy of ancestor block extraction, we collect all pairs of a correctly extracted block and its true or extracted ancestor blocks, then calculated P_H and R_H of the extracted ancestor blocks. Note that it is not symmetric for the descendant side and the ancestor side. Also note that one block is counted many times in the calculation of precision/recall of ancestor extraction when the block has multiple descendant blocks. It is reasonable because such a block is probably important as the ancestor block.

We evaluated HEPS for five types of relationship in a hierarchical structure, namely parent, ancestor (including parents), sibling (excluding self), child and descendant (including children). The results are listed in Table 7. We also listed the accuracy of HEPS for the entire test data set for comparison.

We can see three interesting properties in these results. First, accuracy of sibling heading/block extraction is significantly better than the others. It supports the validity of our observation on heading/block lists in Section 3.3.3. Second, HEPS shows higher precision for ancestors and parents than for children and descendants. It is probably because HEPS often incorrectly extract smaller structure as headings, e.g., sentence-breaking nodes which were not removed by our preprocessing step. Third, the accuracy of ancestor (or parent) block extraction is significantly worse than the overall accuracy of HEPS in spite of high precision of ancestor (or parent) heading extraction. It is probably because the block lists of higher level tend to contain fewer blocks, and blocks of higher level is more likely to have no next block in the same list, which is important for determining the last node of the block.

6. CONCLUSION AND FUTURE WORK

We developed a method that extracts logical hierarchical structure from HTML documents. Our method first extracts hierarchical

headings based on several assumptions on their design, and segment a document into hierarchical blocks by using these headings. This approach is expected to work as long as the document is appropriately designed so that human readers can recognize its hierarchical structure based on its visually prominent headings.

We evaluated our method with a standard Web page corpus, and our experiment shows that our method outperforms the existing page segmentation method. The existing methods, however, focus on top-level page layout structure. The existing methods and our method, therefore, are complementary to each other.

Our analysis on the experimental results suggests that we can improve our method by introducing more visual features and some clustering techniques for correctly classifying nodes.

We have implemented a Web search system on top of our method. Our experiment, which will be reported in another paper, shows that it significantly improves the ranking of search results by exploiting hierarchical blocks and headings extracted by our method. It proves the usefulness of our method.

The reference implementation of our method and the data set we used in our experiment is publicly available on GitHub (<https://github.com/tmanabe>). The data set consists of a list of Internet Archive URLs, a list of XPath expressions representing the content body of each page, and manually labeled heading structures.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 13J06384, 26540163.

7. REFERENCES

- [1] M. D. Adelfio and H. Samet. Schema extraction for tabular data on the web. *Proc. of VLDB*, 6(6):421–432, 2013.
- [2] A. Anjewierden. AIDAS: Incremental logical structure discovery in PDF documents. In *Proc. of ICDAR*, pages 374–378, 2001.
- [3] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proc. of SIGMOD*, pages 337–348, 2003.
- [4] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Accordion summarization for end-game browsing on PDAs and cellular phones. In *Proc. of CHI*, pages 213–220, 2001.
- [5] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. VIPS: A vision-based page segmentation algorithm. Technical Report MSR-TR-2003-79, Microsoft Research, 2003.
- [6] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Block-based web search. In *Proc. of SIGIR*, pages 456–463, 2004.
- [7] D. Chakrabarti, R. Kumar, and K. Punera. A graph-theoretic approach to webpage segmentation. In *Proc. of WWW Conf.*, pages 377–386, 2008.
- [8] H. Chao and J. Fan. Layout and content extraction for PDF documents. In *Document Analysis Systems VI*, volume 3163 of *LNCIS*, pages 213–224. Springer-Verlag, 2004.
- [9] E. Cortez, D. Oliveira, A. S. da Silva, E. S. de Moura, and A. H. Laender. Joint unsupervised structure discovery and information extraction. In *Proc. of SIGMOD*, pages 541–552, 2011.
- [10] S. Debnath, P. Mitra, N. Pal, and C. L. Giles. Automatic identification of informative sections of web pages. *IEEE Transaction on Knowledge and Data Engineering*, 17(9):1233–1246, 2005.
- [11] M. A. El-Shayeb, S. R. El-Beltagy, and A. A. Rafea. Extracting the latent hierarchical structure of web documents. In *Proc. of SITIS 2006*, volume 4879 of *LNCIS*, pages 305–313. Springer-Verlag, 2009.
- [12] J. Fleiss, B. Levin, and M. Paik. *Statistical Methods for Rates and Proportions*. Wiley, John and Sons, Inc., 3rd ed., 2003.
- [13] L. Gao, Z. Tang, X. Lin, Y. Liu, R. Qiu, and Y. Wang. Structure extraction from PDF-based book documents. In *Proc. of JCDL*, pages 11–20, 2011.
- [14] L. Gao, Z. Tang, X. Lin, X. Tao, and Y. Chu. Analysis of book documents’ table of content based on clustering. In *Proc. of ICDAR*, pages 911–915, 2009.
- [15] J. Graupmann, R. Schenkel, and G. Weikum. The SphereSearch engine for unified ranked retrieval of heterogeneous XML and web documents. In *Proc. of VLDB*, pages 529–540, 2005.
- [16] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm. DOM-based content extraction of HTML documents. In *Proc. of WWW Conf.*, pages 207–214, 2003.
- [17] G. Hattori, K. Hoashi, K. Matsumoto, and F. Sugaya. Robust web page segmentation for mobile terminal using content-distances and page layout information. In *Proc. of WWW Conf.*, pages 361–370, 2007.
- [18] M. Keller and H. Hartenstein. GRABEX: A graph-based method for web site block classification and its application on mining breadcrumb trails. In *Proc. of WI*, pages 290–297, 2013.
- [19] M. Keller and M. Nussbaumer. MenuMiner: Revealing the information architecture of large web sites by analyzing maximal cliques. In *Proc. of WWW Conf.*, pages 1025–1034, 2012.
- [20] C. Kohlschütter and W. Nejdl. A densitometric approach to web page segmentation. In *Proc. of CIKM*, pages 1173–1182, 2008.
- [21] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33:159–174, 1977.
- [22] S.-H. Lin and J.-M. Ho. Discovering informative content blocks from web documents. In *Proc. of KDD*, pages 588–593, 2002.
- [23] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. In *Proc. of WWW Conf.*, pages 981–990, 2009.
- [24] H. Okada and H. Arakawa. Automated extraction of non (h)-tagged headers in webpages by decision trees. In *Proc. of SICE Annual Conf.*, pages 2117–2120, 2011.
- [25] F. C. Pembe and T. Güngör. A tree learning approach to web document sectional hierarchy extraction. In *Proc. of ICAART*, pages 447–450, 2010.
- [26] H. Sano, S. Shiramatsu, T. Ozono, and T. Shintani. A web page segmentation method based on page layouts and title blocks. *International Journal of Computer Science and Network Security*, 11(10):84–90, 2011.
- [27] K. Simon and G. Lausen. ViPER: Augmenting automatic information extraction with visual perceptions. In *Proc. of CIKM*, pages 381–388, 2005.
- [28] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning important models for web page blocks based on layout and content analysis. *SIGKDD Explorations Newsletter*, 6(2):14–23, 2004.
- [29] Y. Tatsumi and T. Asahi. Analyzing web page headings considering various presentation. In *Proc. of WWW Conf.*, pages 956–957, 2005.
- [30] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proc. of WWW Conf.*, pages 76–85, 2005.