Reformatting Web Documents via Header Trees

Minoru Yoshida and Hiroshi Nakagawa
Information Technology Center, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
CREST, JST
mino@r.dl.itc.u-tokyo.ac.jp, nakagawa@dl.itc.u-tokyo.ac.jp

Abstract

We propose a new method for reformatting web documents by extracting semantic structures from web pages. Our approach is to extract trees that describe hierarchical relations in documents. We developed an algorithm for this task by employing the EM algorithm and clustering techniques. Preliminary experiments showed that our approach was more effective than baseline methods.

1 Introduction

This paper proposes a novel method for reformatting (i.e., changing visual representations,) of web documents. Our final goal is to implement the system that appropriately reformats layouts of web documents by separating semantic aspects (like XML) from layout aspects (like CSS) of web documents, and changing the layout aspects while retaining the semantic aspects.

We propose a *header tree*, which is a reasonable choice as a semantic representation of web documents for this goal. Header trees can be seen as variants of XML trees where each internal node is not an XML tag, but *a header* which is a part of document that can be regarded as tags annotated to other parts of the document. Titles, headlines, and attributes are examples of headers. The left part of Figure 1 shows an example web document. In this document, the headers are `About Me`, which is a title, and `NAME` and `AGE`, which are attributes. (For example, `NAME` can be seen as a tag annotated to `John Smith`.) Figure 2 shows a header tree for the example document. It should be noted that each node is labeled with parts of HTML pages, not abstract categories such as XML tags.

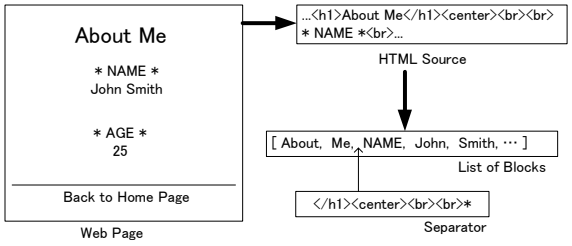

Figure 1: An Example Web Document and Conversion from HTML Documents to Block Lists.

Therefore, the required task is to extract header trees from given web documents. Web documents can be reformatted by converting their header trees into various forms including Powerpoint-like indented lists, HTML tables[1], and Tree-class objects of Java. We implemented the system that produces these representations by extracting header trees from given web documents.

One application of such reformatting is a *web browser on small devices* that shows extracted header trees regardless of original HTML visual rendering. Trees can be used as compact representations of web documents because they show internal structures of web documents concisely, and they can be further augmented with open/close operations on each node for the purpose of closing unnecessary nodes, or sentence summarization on leaf nodes containing long sentences. Another application is a *layout changer*, which change a layout (i.e., HTML tag usage) of one web page to another, by aligning extracted header trees of two web documents. Other applications include HTML to XML transformation and audio-browsable web content (Mukherjee et al., 2003).

[1]For example, the first column represents the root, the second column represents its children, etc.

Proceedings of the ACL Interactive Poster and Demonstration Sessions,
pages 121–124, Ann Arbor, June 2005. ©2005 Association for Computational Linguistics

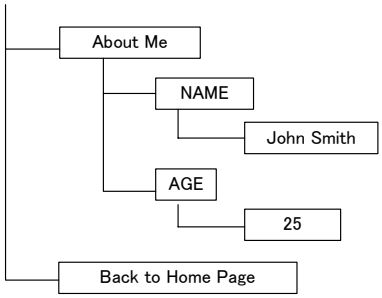

Figure 2: A Header Tree for the Example Web Document

1.1 Related Work

Several studies have addressed the problem of extracting logical structures from general HTML documents without labeled training examples. One of these studies used domain-specific knowledge to extract information used to organize logical structures (Chung et al., 2002). However, their approach cannot be applied to domains for which any knowledge is not provided. Another type of study employed algorithms to detect *repeated patterns* in a list of HTML tags and texts (Yang and Zhang, 2001; Nanno et al., 2003), or more structured forms (Mukherjee et al., 2003; Crescenzi et al., 2001; Chang and Lui, 2001) such as DOM trees. This approach might be useful for certain types of web documents, particularly those with highly regular formats such as `www.yahoo.com` and `www.amazon.com`. However, in many cases, HTML tag usage does not have so much regularity, and, there are even the case where headers do not repeat at all. Therefore, this type of algorithm may be inadequate for the task of header extraction from arbitrary web documents.

The remainder of this paper is organized as follows. Section 2 defines the terms used in this paper. Section 3 provides the details of our algorithm. Section 4 lists the experimental results and Section 5 concludes this paper.

2 Definitions

2.1 Definition of Terms

Our system decomposes an HTML document into a list of *blocks*. A block is defined as the part of a web document that is separated by a *separator*. A separator is a sequence of HTML tags and *symbols*. Symbols are defined as characters in texts that are neither numbers nor letters. Figure 1 shows an example of the conversion of an HTML document to a list of blocks.

[[About Me, [NAME, John Smith], [AGE, 25]], Back to Home Page]]

Figure 3: A List Representation of the Example Web Document

A header is defined as a block that modifies subsequent blocks. In other words, a block that can be a tag annotated to subsequent blocks is defined as a header. Some examples of headers are Titles (e.g., “About Me”), Headlines (e.g., “Here is my profile:”), Attributes (e.g., “Name”, “Age”, etc.), and Dates.

2.2 Definition of the Task

The system produces header trees for given web documents. A header tree can be seen as an indented list of blocks where the level of each node’s indent is equal to the depth of the node, as shown in Figure 2. Therefore, the main part of our task is to give a *depth* to each block in a given web document. After that, some heuristic rules are employed to construct header trees from a list of depths. In the next section, we discuss the task of assigning a depth to each block. Therefore, an input to the system is a list of blocks and the output is a list of depths.

The system also produces *nested-list representation* of header trees for the purpose of evaluation. In nested-list representation, each node that has children is represented by the list whose first element represents the parent and remaining elements represent the children. Figure 3 shows list representation of the tree in Figure 2.

3 Header Extraction Algorithm

In this section, we describe our algorithm that receives a list of blocks and returns a list of depths.

3.1 Basic Concepts

The algorithm proceeds in two steps: separator categorization and block clustering. The first step estimates local block relations (i.e., relations between neighboring blocks) via probabilistic models for characters and tags that appear around separators. The second step supplements the first by extracting the undetermined relations between blocks by focusing on global features, i.e., regularities in HTML tag sequences. We employed a *clustering* framework to implement a flexible regularity detection system that is robust to noise.

3.2 STEP 1: Separator Categorization

The algorithm classifies each block relation into one of three classes: NON-BOUNDARY, RELATING,

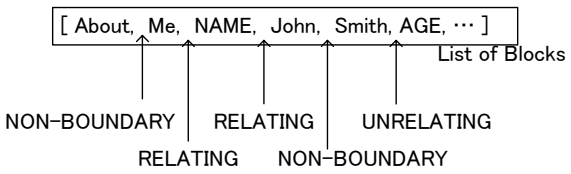

Figure 4: An Example of Separator Categorization.

and UNRELATING. Both RELATING and UNRELATING can be considered to be boundaries; however, blocks that sandwich RELATING separators are regarded to consist of a header and its modified block. Figure 4 shows an example of separator categorization for the list of blocks in Figure 1.

The left block of a RELATING separator must be in the smaller depth than the right block. Figure 2 shows an example. In this tree, `NAME` is in a smaller depth than `John`. On the other hand, both the left and right blocks in a NON-BOUNDARY separator must be in the same depth in a tree representation, for example, `John` and `Smith` in Figure 2.

3.2.1 Local Model

We use a probabilistic model that assumes the locality of relations among separators and blocks. In this model, each separator s and the strings around it, l and r, are modeled by means of the hidden variable c, which indicates the class in which s is categorized. We use the character zerogram, unigram, or bigram (changed according to the number of appearances[2]) for l and r to avoid data sparseness problems.

For example, let us consider the following part of the example document:

```
NAME: John Smith.
```

In this case, `:` is a separator, `ME` is the left string and `Jo` is the right string.

Assuming the locality of separator appearances, the model for all separators in a given document set is defined as $P(\mathbf{l}, \mathbf{s}, \mathbf{r}) = \prod P(l, s, r)$ where $\mathbf{l}$ is a vector of left strings, $\mathbf{s}$ is a vector of separators, and $\mathbf{r}$ is a vector of right strings.

The joint probability of obtaining l, s, and r is

$$P(l, s, r) \quad = \quad P(s)P(c|s)P(l|c)P(r|c)$$

assuming that l and r depend only on c: a class of relation between the blocks around s.[3][4]

[2]This generalization is performed by a heuristic algorithm. The main idea is to use a bigram if its number of appearances is over a threshold, and unigrams or zerograms otherwise.

[3]If the frequency for (l, r) is over a threthold, $P(l, r|c)$ is used instead of $P(l|c)P(r|c)$.

[4]If the frequency for s is under a threthold, s is replaced by its longest prefix whose frequency is over the threthold.

Based on this model, each class of separators is determined as follows:

$$\hat{c} = \arg\max_c P(s)P(c|s)P(l|c)P(r|c).$$

The hidden parameters $P(c|s)$, $P(l|c)$, and $P(r|c)$, are estimated by the EM algorithm (Dempster et al., 1977). Starting with arbitrary initial parameters, the EM algorithm iterates E-STEPs and M-STEPs in order to increase the (log-)likelihood function $\log P(\mathbf{l}, \mathbf{s}, \mathbf{r}) = \sum \log P(l, s, r)$.

To characterize each class of separators, we use a set of typical symbols and HTML tags, called *representatives* from each class. This constraint contributes to give a structure to the parameter space.

3.3 STEP 2: Block Clustering

The purpose of block clustering is to take advantage of the regularity in visual representations. For example, we can observe regularity between `NAME` and `AGE` in Figure 1 because both are sandwiched by the character `*` and preceded by a null line. This visual representation is described in the HTML source as, for example,

```
... <br><br>* NAME *<br> ...
... <br><br>* AGE *<br> ...
```

Our idea is to define the similarities between (context of) blocks based on the similarities between their surrounding separators. Each separator is represented by the vector that consist of symbols and HTML tags included in it, and the similarity between separators are calculated as cosine values. The algorithm proceeds in a bottom-up manner by examining a given block list from tail to head, finding the block that is the most similar to the current block, and collecting them into the same cluster. After that, all blocks in the same cluster is assigned the same depth.

4 Preliminary Experiments

We used a training data that consists of 1,418 web documents[5] of moderate file size[6] that did not have "src" or "script" tags[7]. The former criteria is based on the observation that too small or too large documents are hard to use for measuring performance of algorithms, and the latter criteria is caused by the fact our system currently has no module to handle image files as blocks.

We randomly selected 20 documents as test documents. Each test document was bracketed by hand

[5]They are collected by retrieving all user pages on one server of a Japanese ISP.

[6]from 1,000 to 10,000 bytes

[7]Src tags indicate inclusion of image files, java codes, etc

Algorithm	Recall	Precision	F-measure
OUR ALGORITHM	0.477	0.266	0.329
NO-CL	0.178	0.119	0.139
NO-EM	0.389	0.211	0.265
PREV	0.144	0.615	0.202

Table 1: Macro-Averaged Recall, Precision, and F-measure on Test Documents

to evaluate machine-made bracketings. The performance of web-page structuring algorithms can be evaluated via the nested-list form of tree by *bracketed recall* and *bracketed precision* (Goodman, 1996). Recall is the rate that bracketing given by hand are also given by machine, and precision is the rate that bracketing given by machine are also given by hand. F-measure is a harmonic mean of recall and precision that is used as a combined measure. Recall and precision were evaluated for each test document and they were averaged across all test documents. These averaged values are called *macro-average recall, precision, and f-measure* (Yang, 1999).

We implemented our algorithm and the following three ones as baselines.

NO-CL does not perform block clustering.

NO-EM does not perform the EM-parameter-estimation. Every boundary but representatives is defined to be categorized as "UNRELATING".

PREV performs neither the EM-learning nor the block clustering. Every boundary but representatives is defined to be categorized as "NON-BOUNDARY"[8]. It uses the heuristics that "every block depends on its previous block."

Table 1 shows the result. We observed that use of both the EM-learning and block clustering resulted in the best performance. NO-EM performs the best among the three baselines. It suggests that only relying on HTML tag information is not a so bad strategy when the EM-training is not available because of, for example, the lack of a sufficient number of training examples.

Results on the documents that were rich in HTML tags with highly coherent layouts were better than those on the others like the documents with poor separators such as only one space character or one line feed. Some of the current results on the documents with such poor visual cues seemed difficult for use in practical systems, which indicates our system still leaves room for improvement.

[8]This strategy is based on the fact that it maximized the performance in a preliminary investigation.

5 Conclusions and Future Work

This paper proposed a method for reformatting web documents by extracting header trees that give hierarchical structures of web documents. Preliminary experiments showed that the proposed algorithm was effective compared with some baseline methods. However, the performance of the algorithm on some of the test documents was not sufficient for practical use. We plan to improve the performance by, for example, using larger amount of training examples. Finding other reformatting strategies in addition to the ones proposed in this paper is also important future work.

References

Chia-Hui Chang and Shao-Chen Lui. 2001. IEPAD: Information extraction based on pattern discovery. In *Proceedings of WWW2001*, pages 681–688.

Christina Yip Chung, Michael Gertz, and Neel Sundaresan. 2002. Reverse engineering for web data: From visual to semantic structures. In *ICDE*.

Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. 2001. ROADRUNNER: Towards automatic data extraction from large web sites. In *Proceedings of VLDB '01*, pages 109–118.

A.P. Dempster, N.M. Laird, and D.B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society: Series B*, 39:1–38.

Joshua Goodman. 1996. Parsing algorithms and metrics. In *Proceedings of ACL96*, pages 177–183.

Saikat Mukherjee, Guizhen Yang, Wenfang Tan, and I.V. Ramakrishnan. 2003. Automatic discovery of semantic structures in HTML documents. In *Proceedings of ICDAR 2003*.

Tomoyuki Nanno, Suguru Saito, and Manabu Okumura. 2003. Structuring web pages based on repetition of elements. In *Proceedings of WDA2003*.

Yudong Yang and Hongjiang Zhang. 2001. HTML page analysis based on visual cues. In *Proceedings of ICDAR01*.

Yiming Yang. 1999. An evaluation of statistical approaches to text categorization. *INRT*, 1:69–90.