SEGMENTATION AND CLASSIFICATION OF HTML DOCUMENTS FOR DISPLAY ON SMALL SCREEN DEVICES

Peter M. Kelly [1]

[1] School of Computer Science,
The University of Adelaide

ABSTRACT: The increase in availability of hand-held devices capable of browsing the web, such as mobile phones and PDAs, has necessitated new approaches to the presentation of web content. Pages designed for the display capabilities of traditional desktop PCs are awkward to view on low-resolution displays because of the amount of content present. This project investigates a method for transforming the structure of a web page into a format more suitable for display on such devices, by identifying parts of the document that can be removed without altering the main content of the page.

Algorithms for performing segmentation of a page into its constituent parts, and classification of each of those segments are presented. The resulting classifications determine which types of segments should remain in the page and which should be removed. A prototype implementation is described and evaluated using a set of sample web pages, and shown to produce good results on certain types of pages.

INTRODUCTION

The problem of finding an effective way of displaying web pages that takes into account the hardware limitations of hand-held devices is a challenging one. The development of the web through the 1990s saw a major focus on desktop usage, resulting in the vast majority of sites being targeted at powerful computers with high-resolution displays. Advances made by mobile phone and PDA manufacturers, combined with increasing demand by many people for access to the web anytime and anywhere means that there is a significant need for improved usability in these small screen devices.

This paper presents a process for displaying web pages which splits them up into multiple parts, called *segments*. Each segment is assigned a specific *class* based on the type of content it contains. When viewing a page on a hand-held device, these segments can be displayed separately, or in groups based on their class, and the user can switch between them. Initially, the display would include only the segments with the main content of the page, which is likely to be of most interest to the user. While the segments are chosen according to a fixed process of layout analysis, classes are assigned using artificial intelligence mechanisms which have previously been trained by a human to differentiate between segment types.

RELATED WORK

Numerous projects have investigated the problem of providing web access on handheld devices. These efforts can be divided into two broad categories: Those that require pages specially designed for the target platform, and those which work with existing pages.

In terms of specifically designed pages, WML [WAPFL99] is one of the most well-known approaches, and provides access to sites on a screen-by-screen basis. XHTML Mobile Profile [WAPFL01] is a subset of the XHTML standard which is limited to a simplified set of tags. Dynamic documents [KPT94] contain scripts which automatically adjust the way they are displayed based on the properties of the device they are being viewed on.

For working with existing pages, techniques such as text summarisation [BGMP01], hierarchical menus of links and other objects [STHK01], thumbnail-based navigation [BHR+99], image reduction [CGEV01], and others have been explored. The process of modifying a page before displaying it is called *transcoding*, and is implemented either on the client itself, or an intermediate proxy server.

The work presented here works with existing pages, using client-side transcoding to modify the page. This avoids the need for a proxy, and allows the page to be updated with different modifications without requiring a new download. While past work has largely focused on analysis of the HTML code and structure of pages, a key difference in this project is that the page is analysed by looking at the visual aspects of it, based on the way in which the page is rendered.

WEB PAGE MODEL

For the purposes of analysing and modifying a web page, three data structures are considered:

1. *DOM tree*. This consists of a tree of nodes corresponding to HTML elements and text strings in the original document, as defined by the Document Object Model (DOM) specification [WWWC98b]. *Element* nodes correspond to an open tag and optional close tag in the source file. *Text* nodes represent a sequence of characters that is not inside a tag. The children of an Element node are all of the other elements and text strings that reside between the corresponding open and close tags.
2. *Rendering tree*. The structure of this is similar to that of the DOM tree, except that the nodes represent objects that are visually displayed on the page. Most nodes have a one-to-one correspondence in both trees, but there are a few exceptions. Depending on the display characteristics of certain HTML elements, there may be multiple objects for that element, or even no rendering object. For example, the HEAD tag is invisible and therefore does not have any counterpart in the rendering tree.
3. *Box model*. This is a hierarchy of rectangles which are drawn on the page. Each rendering object has one or more boxes associated with it. For *block* rendering objects such as tables and paragraphs, there is one box created, which usually takes up the whole width of the page or parent box minus margins. For *inline* rendering objects, such as text strings, there may be multiple boxes corresponding to the portion of the object that resides on each line. The box model is defined in version 2 of the Cascading Style Sheets (CSS) specification [WWWC98a].

A web page is loaded and displayed by a HTML rendering engine. This is responsible for parsing the HTML file received from the web, creating all the objects in the DOM, rendering and box data structures, and drawing the page based on these objects and their properties.

PAGE MODIFICATION

The process of transforming a page for display on a small screen consists of three stages. *Segmentation* divides the page up into different *segments* based on visual divisions in the layout. *Classification* assigns a class to each segment indicating the type of content. *Page extraction* constructs a new page containing only the segments which have been assigned a particular class.

Segmentation

The purpose of the segmentation process is to find unique sets of nodes in the DOM tree which form part of an area on the page that is logically distinct from other areas. While past work has considered analysing the DOM tree itself [CMZ03], the approach used here is inspired by the method commonly used for Optical Character Recognition (OCR) [Bre02], where the problem is to find blocks of text separated by whitespace on a printed page. However, rather than looking for whitespace, the process performed by the segmentation algorithm searches for changes in background colour.

Before segmentation can begin, the page must first be loaded and processed by the rendering engine. The prototype used the open source KHTML rendering engine[1], which was modified slightly to get access to the internal data structures representing rendering objects and record details about boxes as they were being drawn.

[1] Part of the KDE project, http://www.kde.org

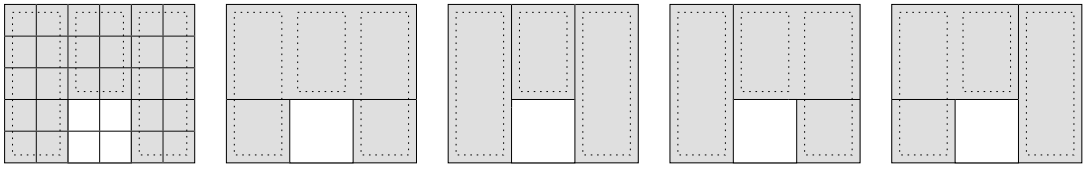

Figure 1: Grid cells with possible region configurations

The first step in the segmentation process is to construct a grid covering the whole page, with horizontal and vertical cell boundaries at each position where the edge of a box sits. The grid cells can then be coloured in according to the most recently-drawn box at each position in the grid. This only takes into account the block boxes corresponding to container elements such as paragraphs and tables; inline boxes belonging to text are ignored at this stage.

The grid is then analysed to identify rectangular regions that have the same background colour. Depending on the page layout, there may be sets of adjacent grid cells with the same colour that do not form a rectangular shape. In this case it is necessary to choose an appropriate division of the area into rectangles. The heuristics used for this ensure that a segment boundary does not cross any text or inline boxes. These heuristics have also been designed so that the chosen rectangles are more likely to be tall and narrow than short and wide - columns are preferred to rows. Figure 1 shows an example of this situation. In this case, (b) is chosen as it is the only choice in which segments do not overlap the boundaries of text boxes, represented as dotted lines.

The next step is to merge some of the adjacent segments together, according to certain rules. If two segments are vertically adjacent, and the top one contains a single line of text and the bottom one multiple lines, they are merged together. This accounts for a common layout technique where the title and body of a page section are given different background colours. The other rule is that vertically adjacent segments with the same background colour that are separated only by whitespace are merged together. This deals with the case where there are multiple coloured paragraphs or title/body pairs in the same column.

Elements residing outside the main flow, such as those which “float” to the left or right of the page, or those which have an absolute position set, are treated separately for segmentation purposes. Each of these is assigned its own segment, which is distinct from the others. This is done to avoid text which wraps around a floating object from being split into multiple segments, and to avoid problems caused by objects that overlap each other because of their assigned positions.

Finally, each segment has a subset of the DOM nodes associated with it that fall within its boundaries. A node is only associated with a segment if the rendering objects of the node itself and all of its children completely reside within that segment. If children of a DOM node are in different segments, then the parent node is not placed in one itself.

Classification

The classification process assigns a class to each segment in the document. It does this using a Bayesian network [RN95], a machine learning technique which can predict the value of an output variable given multiple input variables. For every segment, a set of variables is calculated based on the structure of the DOM tree and the visual rendering. These are fed into a previously-constructed classifier, which determines the most likely class given all of the values. A total of nine classes are defined: *Content*, *Title*, *Navigation*, *Advertising*, *Logo*, *Sidebar*, *ExtraInfo*, *Form*, and *Decoration*.

Each segment has a total of 273 variables. These include 3 different variables for each of the 91 different element types defined in HTML 4.0. The variables for each element type are:

element-text The proportion of characters in the text of the segment that reside in text nodes that are

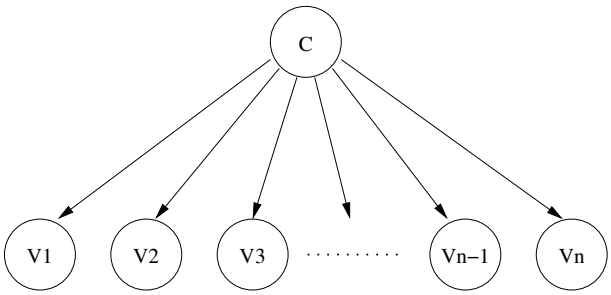

Figure 2: Naive Bayes classifier

descendants of this type of element. This variable ranges from 0–1.

element-rendered The proportion of the total rendered area of the segment corresponding to elements of this type or their descendants. This variable ranges from 0–1.

element-count The number of elements of this type present in the segment

Because the classifier operates on variables with a finite set of possible values, each of these values is scaled to an integer between 0 and 4. This range was chosen to provide a level of granularity that was low enough to have a sufficient number of values for each entry in the conditional probability tables of the Bayesian network. This is necessary to get adequate accuracy when training the classifier.

A particular type of Bayesian network, a *naive Bayes* classifier [Lew98], is used to predict the most likely class. The structure of this is shown in Figure 2, which depicts all of the feature variables $v_1 \ldots v_n$ being dependent on the class variable C. This means that for each variable, a table is maintained which, for every possible value of C, contains the probability of that variable having a particular value, i.e. $P(v = m \mid C)$ for all classes C and values m. While this can be used to determine the probability of a particular variable assignment if the class is known, it can also be used in reverse, i.e. calculating the probability of a particular class given the variable's value. This is done using Bayes' rule, which also relies on the prior probability of the class, $P(C)$, and the prior probability of the variable assignment, $P(v = m)$.

$$P(C \mid v = m) \quad = \quad \frac{P(C)\, P(v = m \mid C)}{P(v = m)}$$

Calculating a class's probability given a set of variable assignments V is simply a matter of multiplying the individual probabilities together. This is because each of the variables are conditionally independent of each other, due to the structure of the naive Bayes classifier. The class assigned to a segment is the one with the highest probability.

$$P(C \mid V) \quad = \quad \prod P(C \mid v_i = m_i) \qquad i = 1 \ldots \|V\|$$

All values of $P(C)$, $P(v = m)$, and $P(v = m \mid C)$ are calculated from a database built during a separate training process, in which all segments in a set of sample pages are manually classified. This process involves cycling through each of the pages and prompting the user to specify which class should be assigned to each segment. Training is done as part of the development process and once the database is constructed, it can be used for automatic classification. No further learning is done by the classifier unless more manual training is performed by a user.

Page Extraction

A page extract is a partial view of a document in which only certain segments are visible. When viewing a page using this system on a small screen device, a series of page extracts is constructed containing different selections of segments from the original document. Each extract contains all segments of a particular class. After loading a page, one extract for each class is generated, and the extract containing *Content* segments is initially displayed. This means that when a user first views a page, they see only the main body of text on the page, and not peripheral information such as navigation and advertising. The user interface provides mechanisms to switch to a different extract in order to view the segments contained within it, so that all of the information on the original page is still accessible.

In implementation terms, a page extract is a separate document which contains copies of a subset of nodes from the original document. The nodes included are those assigned to the relevant segments, as well as certain additional nodes necessary for maintaining the layout and presentational aspects of the relevant parts of the page that are visible in the original. Examples of these additional nodes include those that represent white space between segments, ancestor nodes with certain CSS style rules associated with them, and empty table cells necessary to keep the correct position of remaining content cells in a table.

EVALUATION

Results

The prototype system was tested with three sets of pages collected from the web. The **news_articles** set consists of 100 pages from different news sites. These generally consist of a multi-column layout with a large body of text in the middle, and other peripheral information such as advertising and navigation links on the edges of the page. The **news_frontpage** set contains 100 pages from the same sites as the previous set, but showing the front pages instead of the actual articles. These are also generally well structured, but not as clearly as the articles themselves. Finally, a **general** set was collected based on 100 randomly selected pages. These varied greatly in structure and content.

Two measures of classifier performance were used: *recall* and *precision* [RBJ89]. Recall, denoted ρ, is the probability that if a particular class has been manually assigned to a segment during training, then it will be assigned to that segment during automated classification. Precision, denoted π, is the probability that if a particular class is chosen for a segment by the classifier, then it is correct according to the class manually assigned during training. Using ϕ to represent the classifier, these can be expressed as:

$$\begin{aligned} \rho &= P\left(\text{classified}\left(S\right) = C \mid \phi, \text{trained}\left(S\right) = C\right) \\ \pi &= P\left(\text{trained}\left(S\right) = C \mid \phi, \text{classified}\left(S\right) = C\right) \end{aligned}$$

Initially, a test was run for each data set. Each of these involved training the classifier using manual classifications of all pages in that set, and then testing automatic classification with the same set of pages that were used for training. This resulted in fairly good results for *Content* segments in the **news_articles** set, although most other classifications in this and other sets were not highly accurate. The results of the tests are shown in Table 1. *Accuracy* is the overall measure for each data set, taken from the precision and recall values weighted according to the number of segments in each class.

Another test was run against the same data sets, but this time the classifier was trained on 90% of the pages and tested against the remaining 10%. This was repeated 10 times, each with a different portion of the pages used as the other 10%, and the results averaged. This test ensured that the classifier was never used to classify a page that it had been trained against. Results from this test indicated an average of 52% accuracy for **news_articles**, 43% for **news_frontpage**, and 39% for **general**.

Based on the poor results obtained for the 9 different classes, it was decided to reduce the set of classes to just *Content* and *Other*, the latter representing all other 8 classes grouped together. Given that this

	news_articles		**news_frontpage**		**general**	
	π	ρ	π	ρ	π	ρ
Content	83%	90%	62%	54%	84%	62%
Title	26%	56%	23%	50%	54%	70%
Navigation	88%	65%	74%	63%	55%	54%
Advertising	51%	46%	37%	35%	58%	67%
Logo	46%	56%	39%	41%	38%	45%
Sidebar	40%	65%	46%	49%	54%	69%
ExtraInfo	50%	60%	44%	48%	54%	58%
Form	79%	71%	68%	81%	74%	83%
Decoration	35%	81%	47%	85%	67%	83%
Accuracy	65%		56%		62%	

Table 1: Classification accuracy with test set = training set

	news_articles		**news_frontpage**		**general**	
	π	ρ	π	ρ	π	ρ
Content	76%	92%	52%	63%	71%	67%
Other	99%	97%	89%	84%	84%	86%
Accuracy	97%		79%		80%	

Table 2: Classification accuracy after class merging

work is targeted primarily towards initially displaying only *Content* segments to the user, with the other types normally being of less interest, this was seen as a reasonable compromise to make. The first test was re-run, with much better results, as shown in Table 2.

Even with the 90/10% split method described above, good results were obtained for the 2-class version, with the results being 92%, 74%, and 73% respectively.

Discussion

The prototype has been shown to perform well on pages with clear visual distinctions between the main content and other types of segments. The **news_articles** set in particular has very clear differences, in that all had a large body of text classified as *Content*, which was significantly different from other parts of the page. This indicates that for a restricted set of pages, the approach taken here is, in most cases, sufficient. Data sets with a less consistent page structure did not fare as well, although reasonable results were still obtained in the tests with only two classes. These results could be increased by making improvements to the way in which the segmentation and classification algorithms work, in particular correcting some of the limitations described below.

Limitations

The poor results found when classifying segments based on the original set of 9 classes can be attributed to three broad types of problems: those related to segmentation, to classification ambiguity, and to the set of feature variables.

Segmentation is not always effective because it assumes that the segments of a page are separated by changes in background colour, which is not the case on all pages. Thus, adjacent areas on the page which should logically be placed into separate segments are sometimes put in the one segment. This introduces the problem that a segment can contain content of different types, even though only one can possibly assigned. This can pollute the training database with only partially correct classifications since the trainer must make a choice about which class to assign when there is more than one present in the segment. If whitespace was taken into account as well as background colour changes, then a wider range of pages could be successfully handled by the segmentation algorithm.

The correct class to choose for a given segment is not always clear. The person performing the training process must make a judgment about which to choose when ambiguity arises, which limits the quality of the input data. Reducing the set of classes to just *Content* and *Other* increases the measured level of accuracy since the classifier has less choices to make and therefore fewer misclassifications occur.

The feature variables used for classifying each segment have an impact on the level of accuracy. The more relevant the variables are to the classification, the fewer errors that will result. Unfortunately the different segment classes cannot be distinguished by visual properties alone; often the choice is influenced by the semantic meaning of the content in a segment.

Some HTML features are not supported in the prototype because they complicate the segmentation process. Frame sets are excluded because they actually consist of multiple documents, which would need to be merged in some way for display on the small screen. Javascript is also disabled because scripts which attempt to modify the page are likely to run into problems if they try to work with the modified versions, in which the structure may have changed significantly.

Suggestions for Future Work

The segmentation process should ideally be able to recognise divisions in the page other than those based on changes in background colour. Additional steps could be added to the segmentation process to detect areas of whitespace, lines dividing areas on the page, and other common layout techniques used to separate parts of a page from each other.

More accurate classifications could be obtained by extending the set of feature variables defined for each segment to include additional information such as the position of the segment on the page, the purpose of links, and the meaning of text. This would require semantic analysis of the content. Hard-coded rules could also be used to detect common cases such as images of a certain size being classified as ads. Allowing multiple classes per segment would allow the system to cater for multiple types of information in a segment, although it may not be easy to do this accurately.

CONCLUSION

This project has investigated one approach to displaying web pages which involves the modification of document structure to present a subset of the information normally included in a page. While the approach has been found to be reasonably effective, it is clear that much more work is necessary before it is ready for end-users. Many issues need to be addressed concerning the analysis and classification of web pages, as well as dealing with a wider variety of page layouts. Additionally, support for some of the more advanced functionality provided by modern web browsers needs to be integrated with the way in which pages are analysed, modified and presented.

The ideas introduced here have the potential to be used as a basis for future research in the area. It is hoped that these will lead to improvements in the way that web pages are analysed, and the way in which structure is used to provide new functionality in web browsers. In particular, analysis based on visual aspects of a rendered document has received relatively little attention in the past, and the exploration of this concept here provides insights which may be useful in a range of areas in addition to that of small screen rendering. With sufficient fine-tuning and implementation effort, the ideas presented in this paper can contribute towards a significantly better experience to users of hand-held devices when browsing the web.

ACKNOWLEDGMENTS

Special thanks to Barry Dwyer, who acted as my academic supervisor for this project, and to Paul Coddington and Nickolas Falkner for providing helpful feedback on the paper.

REFERENCES

[BGMP01] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Accordion summarization for end-

game browsing on pdas and cellular phones. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 213–220. ACM Press, 2001.

[BHR+99] S. Bjork, L. E. Holmquist, J. Redstrom, I. Bretan, Rolf Danielsson, Jussi Karlgren, and Kristofer Franzen. WEST: a web browser for small terminals. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 187–196. ACM Press, 1999.

[Bre02] T. M. Breuel. Two algorithms for geometric layout analysis. In *Proceedings of the Workshop on Document Analysis Systems, Princeton, NJ, USA*, 2002. Also selected for inclusion in the post-conference book.

[CGEV01] S. Chandra, A. Gehani, C. S. Ellis, and A. Vahdat. Transcoding characteristics of web images. In M. Kienzle and W. Feng, editors, *Multimedia Computing and Networking (MMCN'01)*, volume 4312, pages 135–149, San Jose, CA, jan 2001. SPIE - The International Society of Optical Engineering.

[CMZ03] Y. Chen, W. Ma, and H. Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *Proceedings of the twelfth international conference on World Wide Web*, pages 225–233. ACM Press, 2003.

[KPT94] F. Kaashoek, T. Pinckney, and J. Tauber. Dynamic documents: Mobile wireless access to the WWW. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.

[Lew98] D. D. Lewis. Naive Bayes at forty: The independence assumption in information retrieval. In C. Nédellec and C. Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[RBJ89] V. Raghavan, P. Bollmann, and G. S. Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7(3):205–229, 1989.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[STHK01] B. N. Schilit, J. Trevor, D. M. Hilbert, and T. K. Koh. m-links: An infrastructure for very small internet devices. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 122–131. ACM Press, 2001.

[WAPFL99] Wireless Application Protocol Forum Ltd. Wireless Application Protocol: Wireless Markup Language Specification, November 1999. Version 1.2.

[WAPFL01] Wireless Application Protocol Forum Ltd. XHTML Mobile Profile, October 2001. WAP-277-XHTMLMP-20011029-a.

[WWWC98a] World Wide Web Consortium. Cascading Style Sheets, level 2: CSS2 Specification. W3C Recommendation, May 1998.

[WWWC98b] World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, October 1998. Version 1.0.