A General Approach for Partitioning Web Page Content Based on Geometric and Style Information

Hui Guo Jalal Mahmud Yevgen Borodin Amanda Stent
I.V. Ramakrishnan
Computer Science Department, Stony Brook University
Stony Brook, NY 11794-4400
{huguo, jmahmud, borodin, stent, ram}@cs.sunysb.edu

Abstract

In this paper, we describe a general-purpose approach for partitioning Web page content. The novelty of our approach lies in the use of detailed layout information from a Web page renderer to determine spatial locality and identify visual separators, and the use of relaxed matching over presentation style information to determine presentation style similarity. We present several examples to illustrate the generality of our approach.

1. Introduction

Considerable research has been done on automatic partitioning of Web page content into semantically related groups. This research has many applications, such as wrapper induction [12], information extraction [1, 5], support for the Semantic Web [7], task learning [4] and accessibility [10]. A key problem in partitioning is determining how to use layout and presentation information most effectively to group content. In this paper, we present an approach to partitioning of Web page content that is more general than that in prior work. This approach is based on the observation that information about spatial locality is most often used to **cluster**, or draw boundaries *around* groups of items in a Web page, while information about presentation style is used to **segment**, or draw boundaries *between* groups of items. These two types of information are complementary. By clustering items into large groups using spatial locality first, and then by segmenting them within those groups using presentation style, we can reduce partitioning errors.

Consider the Amazon page shown in Figure 1. It contains two navigation menus, one vertical and one horizontal. It also contains several product descriptions in the main part of the page. The page contains visual cues to help the viewer find information quickly and easily. The alignment of the two menus, and the images associated with them, help them stand out from other content. Vertical and horizontal spacing and font color, size and style differentiate the sale items in the main part of the page and the submenus in the vertical menu. Of course, a computer cannot see the Web page as the user does. Instead, visual cues must be mapped to features associated with the Web page markup, and the human visual segmentation process into algorithms that cluster and segment the Web page content.

Most prior work in this area uses the DOM tree for the Web page as an approximation of the visual layout for the page. Some of these techniques are domain [6] or site [13] specific. Others are more general, but because DOM trees contain only information about the location of Web page elements relative to each other, they lose information about the visual layout that is useful for partitioning [1, 3, 5, 15]. In addition to the DOM tree itself, some algorithms identify HTML tags that serve as visual separators [2, 9] or groupers [14]. Finally, some approaches require semantic information such as ontologies [11].

The key novel features of our approach lie in how we determine spatial locality, find presentation similarities, and identify visual separators. In determining *spatial locality* we do not rely on the DOM tree, where the only location information is parent-child relationships between HTML tags. Instead, we use the Mozilla frame tree [8] to get exact coordinates for content elements. To find *presentation similarities*, we permit relaxed matching on presentation styles: presentation styles that are similar but not identical can form parts of patterns for segmentation. In identifying *visual separators*, we do not restrict ourselves to HTML tags, but use the content coordinates from the Mozilla frame tree to identify white space in the page. Our approach does not depend on manually specified rules or any domain knowledge. In comparison to other existing approaches to Web page partitioning, our approach can handle a wider variety of Web pages with greater precision.

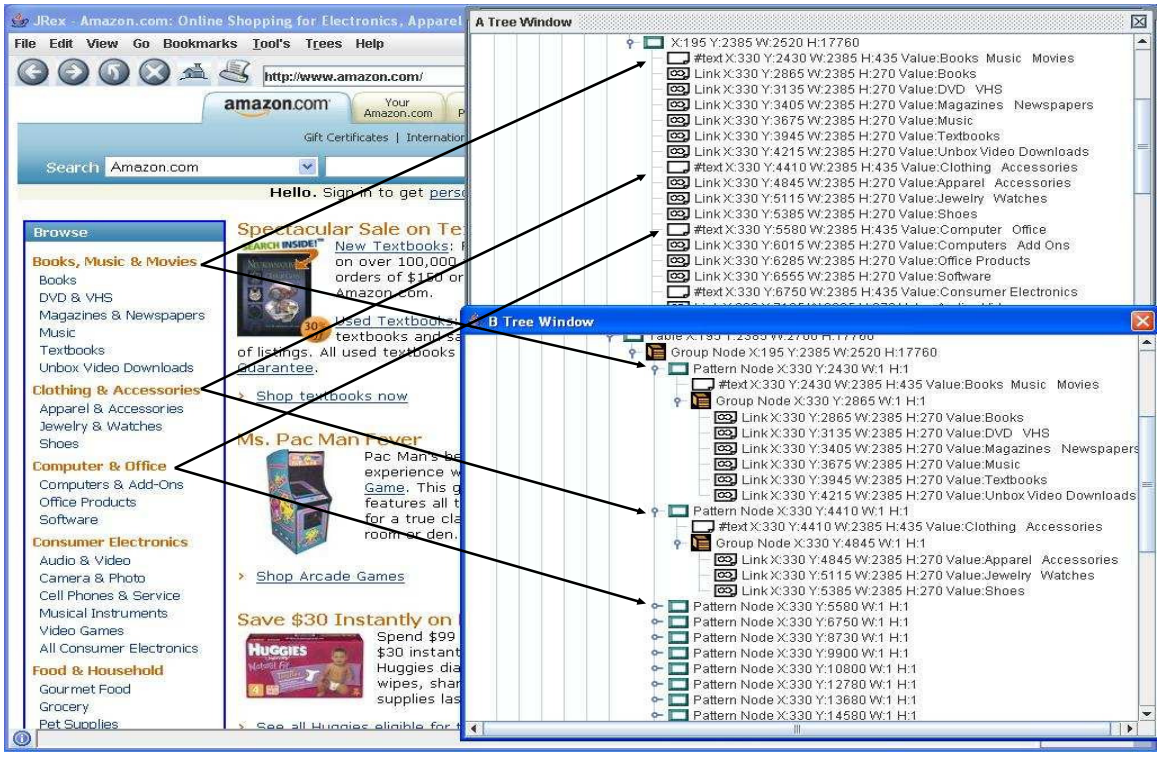

Figure 1. Block and partition finding in the Amazon home page

2. System Architecture

In our system, an input Web page passes through several stages of processing, each of which makes use of different information. The input HTML passes through the Mozilla engine. The output from this stage is a *frame tree* [8], which contains layout and style information sufficient for a visual rendering of the Web page. The frame tree passes through the **Block Finder**. This stage uses detailed *visual layout information* from the frame tree to cluster the frames (Section 3). The output is a *block tree*: a frame tree in which geometrically aligned blocks have been identified. The block tree passes through the **Partition Finder**. This stage uses *presentation style information* to segment each block into one or more partitions (Section 4). The output is a *partition tree*. Each block and partition is a frame subtree.

3. Clustering: The Block Finder

Semantically related pieces of content in a Web page tend to be located near each other, often sharing the same vertical and/or horizontal alignment. Consequently, geometric alignment of frames in the Mozilla frame tree may imply semantic relatedness. If all descendants of a frame are consistently aligned either along X or Y axes, we call such a frame *consistent*. A *maximal semantic block*, or simply *block*, is the largest of the consistent frames on the path from a leaf to the root of a frame tree.

Algorithm *FindBlocks*
Input: $Frame$: node of a frame tree
Output: *Blocks*: set of maximal semantic blocks

1. Identify all children $C_1, C_2, \ldots, C_m$ of $Frame$
2. $Frame.IsConsistent \leftarrow$**true**
3. **for** $j \leftarrow 1$ **to** m
4. **do if** $C_j.IsLeaf$ = **false**
5. **then** *FindBlocks*(C_j)
6. **if** $C_j.Alignment$ = NONE
7. **then** $Frame.IsConsistent \leftarrow$**false**
8. **if** $Frame.IsConsistent$ = **false**
9. **then for** $j \leftarrow 1$ **to** m
10. **do if** $C_j.Alignment \neq$ NONE
11. **then** $Blocks \leftarrow Blocks \cup \{C_j\}$
12. **else** $Frame.Alignment \leftarrow$*GetAlignment*($Frame$)
13. **if** $Frame.Alignment$ = NONE
14. **then for** $j \leftarrow 1$ **to** m
15. **do if** $C_j.Alignment \neq$ NONE
16. **then** $Blocks \leftarrow Blocks \cup \{C_j\}$
17. **return** $Blocks$

The *FindBlocks* algorithm is used to find the blocks in a frame tree. The algorithm runs a depth-first search over the frame tree and recursively determines whether the frames are consistent, ignoring the alignment of leaf frames. A frame is *consistently X-aligned* if all of its non-leaf descendants are X-aligned. Similarly, a frame is *consistently Y-aligned* if all of its non-leaf descendants are Y-aligned. Otherwise, the frame is not considered to be consistent. In this case, all of its children are marked as blocks.

The *FindBlocks* algorithm uses the *GetAlignment* algorithm to check whether the children of a frame have matching alignment. The *GetAlignment* algorithm determines that a frame is *X-aligned* if all of its children are aligned on the left, right, or center of the X-axis. Y-alignment of a frame is computed in a similar fashion.

Algorithm *GetAlignment*
Input: $Frame$: node of a frame tree
Output: $Alignment$: alignment of $Frame$'s descendants
1. Identify all children $C_1, C_2, \ldots, C_m$ of $Frame$
2. $XFirst \leftarrow C_1.X$; $YFirst \leftarrow C_1.Y$
3. $XAlignedDescends \leftarrow$**true**
4. $YAlignedDescends \leftarrow$**true**
5. $Alignment \leftarrow$NONE
6. **for** $j \leftarrow 2$ **to** m
7. **do if** $C_j.IsLeaf$ = **false**
8. **then** $XCord \leftarrow C_j.X$
9. $YCord \leftarrow C_j.Y$
10. **if** $XCord \neq XFirst$
11. **then** $XAlignedDescends \leftarrow$**false**
12. **if** $YCord \neq YFirst$
13. **then** $YAlignedDescends \leftarrow$**false**
14. **if** $C_j.Alignment \neq XAlign$
15. **then** $XAlignedDescends \leftarrow$**false**
16. **if** $C_j.Alignment \neq YAlign$
17. **then** $YAlignedDescends \leftarrow$**false**
18. **if** $XAlignedDescends$ = **true**
19. **then** $Alignment \leftarrow XAlign$
20. **if** $YAlignedDescends =$ **true**
21. **then** $Alignment \leftarrow YAlign$
22. **return** $Alignment$

4. Segmenting: The Partition Finder

In web page content, geometric alignment is generally a strong indicator of semantic relatedness. However, content that is geometrically aligned can frequently be segmented into smaller content-related groups using presentation style information. For example, on the NY Times home page in Figure 2 the headline story items all have the same Y-alignment, but they form several individual stories, each composed of: an article title in a large blue font, a byline in a small grey font, a short abstract in a small black font, and (optionally) related links in a small blue font. If we can find these patterns of presentation styles, then we can segment content into semantically related partitions.

The *Partition* algorithm finds partitions in a block tree. It runs bottom-up over the frame subtree formed by each block. For each node N, it finds the maximal repeating pattern(s) in presentation style in N's children.

Two issues complicate this process. First, the patterns may not be exact; for example, there may be no list of related items for a news story. We use separators to determine how much content a pattern should cover and choose between multiple possible patterns [2]. We define a separator as follows: A *separator* is either one of the HTML tags HR or P, or is white space (horizontal or vertical) between two adjacent nodes that is greater than the mean amount of white space between adjacent nodes in this node list.

Second, the elements of the pattern may not be exact matches; for example, on the NY Times page the first headline has a slightly larger font size than the other headlines but they all have the same color and font. We use relaxed matching to deal with slight variations in presentation style. The presentation style of a node is a tuple formed of the font type, font size and font style (color, bold face, italics etc.) of text in the node. Two nodes have similar presentation style if the font type and font style of the two nodes is identical, and the font sizes are within two points of each other.

The *PartitionList* algorithm finds maximal repeating patterns in a list of nodes. The $GetFirstSequence$ and $GetNextSequence$ methods return a sequence of nodes bounded by separators. The $SimilarSequence$ method computes the longest common subsequence of two node sequences. If this subsequence covers at least 60% of each of the node sequences, then they are considered to be similar.

Algorithm *PartitionList*
Input: $L \leftarrow (C_1, C_2, \ldots, C_m)$: children of block root N
Output: P: A partitioning of these $C_1, C_2, \ldots, C_m$
1. **for** $j \leftarrow 1$ **to** m
2. **do** $Replace(C_j, Partition(C_j), L)$
3. $Current \leftarrow GetFirstSequence(L)$
4. **if** $Equal(Current, L)$
5. **then return** L
6. $Replace(Current, PartitionList(Current), L)$
7. **while** $NotEmpty(Current)$
8. $Next \leftarrow GetNextSequence(L, Current)$
9. $Replace(Next, PartitionList(Next), L)$
10. **if** $SimilarSequence(Current, Next)$
11. **then** $addToGroupNode(Current, Next, L)$
12. $Current \leftarrow Next$
13. **return** L

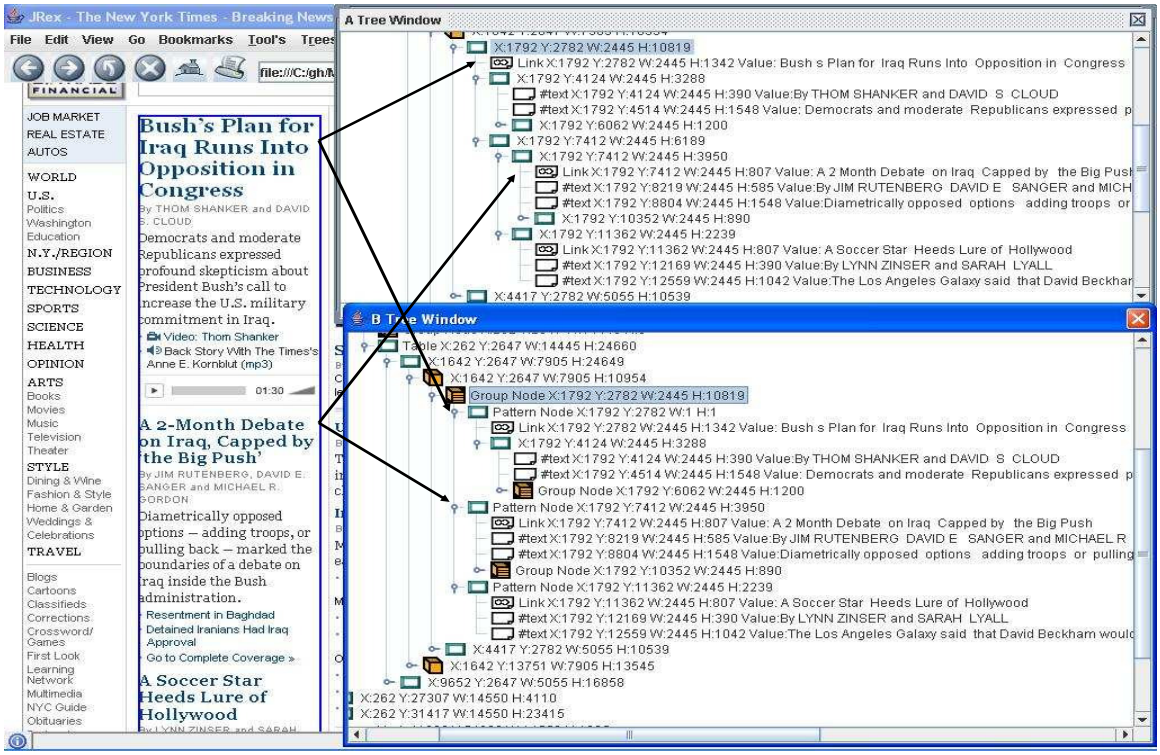

Figure 2. Block and partition finding in the NY Times home page

5. Examples

In this section, we show example output from our system for a range of Web pages. For each example, a part of the page is shown, along with the block tree (top) and partition tree (bottom). Nodes marked "Group Node" and "Pattern Node" are created by the Partition Finder. Nodes marked with box icons are created by the Block Finder. All other nodes are from the frame tree. Arrows connect content in the page to elements in the block and partition trees.

Figure 3 shows part of a Blackboard user page that gives a list of system functionalities. Clustering groups each sublist into a block, including its title and related image. Segmenting merely adds a grouping of the elements of the sublist. This example shows the power of having exact visual layout information.

Figure 1 shows part of the Amazon home page. The structure of this part of this page may seem similar to that of Blackboard. However, because the menu items are all vertically aligned links, clustering groups all of them into a single block. Segmenting gives one partition for each submenu. Partitions are found even though some submenus have more items than others, because the separators and presentation style information identify patterns.

Figure 2 shows part of the NY Times home page. Clus-

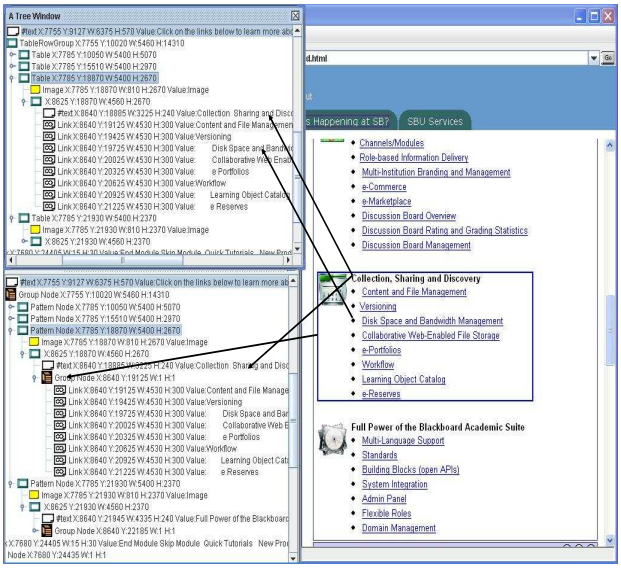

Figure 3. Block and partition finding in the Blackboard system

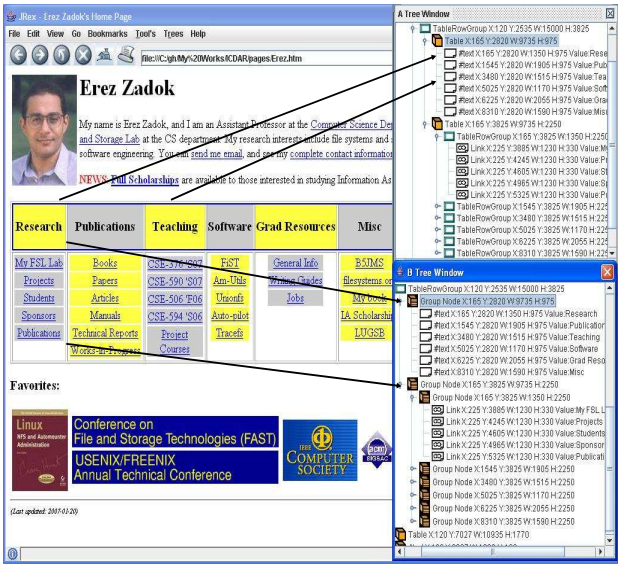

Figure 4. Block and partition finding in a personal home page

tering groups all the headline news stories into a single block. Segmenting gives a partition for each headline news story. Even though the font size for the first headline is a little larger than the font size for the others, the first headline news story is also a partition because segmenting uses relaxed matching for presentation style information.

Figure 4 shows part of a personal home page, including a table. There are two ways for items to be grouped in a table: as a set of rows (horizontal alignment) and as a set of columns (vertical alignment). Clustering currently prioritizes horizontal alignment, so blocks and partitions for this table are incorrect. In the future, the Block Finder may also use separators – because the white space between columns in this table is greater than that between rows, white space would help to disambiguate in this case.

6. Conclusions and Future Work

In this paper, we have presented a novel approach to partitioning Web page content. This approach uses visual layout and presentation style information in a two-stage process that gives good results for a range of Web pages from different domains. Our system is in use in the HearSay accessible Web browser [10].

We are conducting an evaluation of our system. We are also experimenting with parallel/iterative use of the BlockFinder and the PartitionFinder to improve system performance. Finally, we are extending the system to automatically label topics in blocks and partitions.

References

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of SIGMOD 2003*, 2003.

[2] D. Cai et al. VIPS: A vision-based page segmentation algorithm. Technical Report MSR-TR-2003-79, Microsoft Research, 2003.

[3] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *Proceedings of WWW 2001*, 2001.

[4] N. Chambers et al. Using semantics to identify web objects. In *Proceedings of AAAI 2006*, 2006.

[5] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of VLDB 2001*, 2001.

[6] D. Embley and L. Xu. Record location and reconfiguration in unstructured multiple-record web documents. In *Proceedings of WebDB 2000*, 2000.

[7] O. Etzioni et al. Web-scale information extraction in KnowItAll. In *Proceedings of WWW 2004*, 2004.

[8] Gecko:key Gecko structures and invariants. http://wiki.mozilla.org/Gecko. Viewed on June 7, 2007.

[9] H. Guo and A. Stent. Taxonomy based data extraction from multi-item web pages. In *Proceedings of the Workshop on Web Content Mining with Human Language Technologies at ISWC 2006*, 2006.

[10] J. Mahmud, Y. Borodin, and I. Ramakrishnan. CSurf: A context-driven non-visual web-browser. In *Proceedings of WWW 2007*, 2007.

[11] S. Mukherjee, G. Yang, and I. Ramakrishnan. Automatic annotation of content-rich HTML documents: Structural and semantic analysis. In *Proceedings of ISWC 2003*, 2003.

[12] I. Muslea, S. Minton, and C. Knoblock. Active learning with strong and weak views: A case study on wrapper induction. In *Proceedings of ICJAI 2003*, 2003.

[13] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Site-wide annotation: Reconstructing existing pages to be accessible. In *Proceedings of ASSETS 2002*, 2002.

[14] Y. Yang and H. Zhang. HTML page analysis based on visual cues. In *Proceedings of ICDAR 2001*, 2001.

[15] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of WWW 2005*, 2005.