A Web Page Segmentation Method based on Page Layouts and Title Blocks

Hiroyuki Sano†, Shun Shiramatsu†, Tadachika Ozono†, and Toramatsu Shintani†

†Dept. of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology, Aichi, 466–8555 Japan

Summary
In this work, we describe a new Web page segmentation method to extract the semantic structure from a Web page. A typical Web page consists of multiple elements with different functionalities, such as main content, navigation panels, copyright and privacy notices, and advertisements, and Web page segmentation is the division of the page into visually and semantically cohesive pieces. The proposed method is comprised of three steps. First, it determines the layout template of a Web page by template matching. Second, it divides the page into minimum blocks. Third, it assembles groups of these blocks into Web content blocks. While the minimum blocks can play many roles, in this study we have focused on the those that are the titles of various Web content bits. We used decision tree learning with nine parameters for each minimum block to extract the title blocks from Web pages. Experimental results showed that the decision tree generated by the J48 algorithm is the most suitable for this type of extraction.

Key words:
Web page segmentation, Page layout, Title block, Machine learning

1. Introduction

We have developed an algorithm for dividing a Web page into visually and semantically cohesive pieces called "Web content blocks." This process is known as Web page segmentation. Web page designers appoint a headline to each Web content on a page, which we call "title blocks," to make for easy reading, and it is these blocks that the proposed algorithm focuses on. Title blocks can be used as separators when we segment a Web page. In this paper, we propose the use of decision tree learning to extract the title blocks from Web pages.

A typical Web page consists of multiple elements with different functionalities, such as main content, navigation panels, copyright and privacy notices, and advertisements. Figure 1 shows a screenshot of a page from Reuters.com, a news site that brings viewers the latest news from around the world. This page includes four Web content blocks: the "main content block," which is enclosed by a solid red line, and three "noisy content blocks," which are enclosed by broken blue lines. Visitors to the page are only interested in the main content and have no use for the noisy content. Web page segmentation clearly has a variety of benefits and potential Web applications, but if applications such as information retrieval or extraction treat all content on a Web page equally—e.g., with no differentiation between main and noisy content—there may be a decline in accuracy. It is necessary that such applications deal only with the main content of a Web page.

Fig. 1 A page from Reuters.com that contains both main (solid red line) and noisy (broken blue lines) Web content.

These days, Web page segmentation is an extensively studied topic. The methods proposed in the related works divide a Web page into very small pieces at first and then combine semantically cohesive pieces into Web content blocks. We propose a method for assembling the small pieces that focuses on pieces that function as headlines for specific bits of Web content.

2. Related works

There are several different Web page segmentation algorithms already in place, the most popular of which are DOM-based [3, 6] and layout-based [4, 2, 5].

In DOM-based segmentation, tag information is used to divide a Web page based on the Document Object

Manuscript received October 5, 2011
Manuscript revised October 20, 2011

Model (DOM). The DOM has a tree structure in which each node contains one of the components from an HTML tag.

Buyukkokten et al. [3] split a Web page using some relatively simple DOM nodes such as the <P>, <TABLE>, and <UL> nodes for further conversion or summarization.

Lin and Ho [6] only consider the <TABLE> node and its offspring to be a content block and use an entropy-based approach to determine the informative ones. Nodes such as <TABLE> and <P> are not only used for content organization but also for layout presentation. For example, consider the Web page layout in Fig. 2. In this example, when the part marked "1" in Fig. 2(a) is a single content block, only "<TABLE>" needs to be extracted. On the other hand, when the part marked "2" is a single content block, we must extract and merge the "B" and "D" nodes of the tree. DOM-based segmentation cannot detect the "2" part as a single content block.

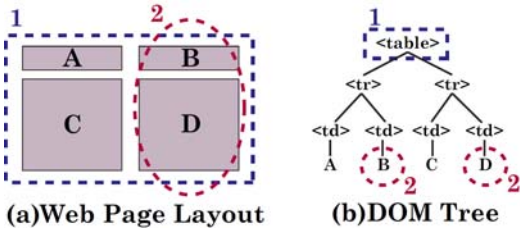

Fig. 2 Problem of DOM-based segmentation.

The layout-based segmentation method uses layout information after rendering, assuming that similar content blocks are located close to each other and have similar shapes.

Cai et al. [4, 2, 5] use layout information such as "font," "color," and "size" to restructure a Web page in a content block tree. Baluja [1] considers the Web page segmentation problem from the perspective of a machine learning framework: he re-examines the task through the lens of entropy reduction and decision tree learning. However, a consideration of the layout differences that exist in the various parts of Web pages seems to be lacking in this layout-based segmentation method. Web pages have various layouts for various parts of themselves (associated pages, "headers," "footers," etc.), so it is difficult to adapt the same rules or parameters for all Web pages and all parts of a Web page.

We have therefore developed a new Web page segmentation algorithm that is a kind of layout-based segmentation algorithm. It first detects the layout template of a Web page and then divides the page into very small pieces and combines the pieces to form various Web content bits—semantically cohesive pieces—while keeping a lookout for pieces that can function as headlines for the content. We introduce the details of the proposed algorithm in the next section.

3. Proposed method

The proposed segmentation algorithm is comprised of three steps: (1) layout template detection, (2) division into minimum blocks and detecting title blocks, and (3) combination into Web content bits.

3.1 Layout template detection

Our segmentation method first detects the layout template of a Web page. This step is useful in terms of speculating where the main content of the page is located.

There are similarities and differences in the layouts of all Web pages. For example, many sites have wide blocks such as a logo or a search form at the top of each page. We assume that all Web pages use some kind of layout template, so our method classifies the templates by considering the similarities and differences. Ideally, Web pages are designed so that users can easily navigate the pages and locate the desired content. For this reason, there tends to not be much variation in terms of layout templates. In this paper, we define the various template blocks of a Web page as the "header (TB_h)," "footer (TB_f)," "left menu (TB_l)," "right menu (TB_r)," and "center (TB_c)." The proposed method classifies a Web page into one of the eight layout templates (T_1–T_8) (shown in Fig. 3), which are combinations of template blocks.

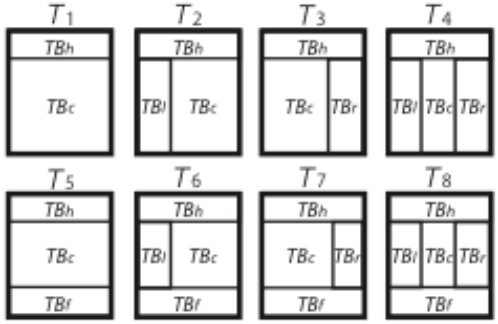

Fig. 3 Eight types of layout templates.

The workflow used to detect the layout template of a given Web page is shown in the ***DetectLayoutTemplate*** procedure in Fig. 4. The procedure uses a DOM structure and size after rendering because the template blocks almost completely branch off at the depth near to the root (BODY) node of the Web page. The ***DevideDOM*** procedure sequentially judges whether a node should be divided or be a block based on D_{size} from the BODY node. The rule is:

(1) If the size of the DOM node's child nodes are all smaller than D_{size}, the DOM node is a block.

(2) If the size of the DOM node is greater than D_{size}, the DOM node is divided into child nodes, and these child nodes are judged again.

(3) If the size of the DOM node is smaller than Dsize, the DOM node is a block.

```
Input : DomTree (DOM tree of a web page)
Output : TB = {tb1, tb2, .., tbn} (Template blocks)

procedure DetectLayoutTemplate(DomTree)
begin
  D_size ← size of BODY node;
  T_candidate ← {}; // candidate for template
  while D_size > θ do
    N ← DivideDOMBySize(DomTree, D_size);
    border ← T_candidate ∪ MakeTemplateBorder(N);
    T_candidate ←
          T_candidate ∪ MakeBlockByBorder(N, border);
    D_size ← D_size * 0.9;
  enddo
  TB ← DetectTemplate(T_candidate);

  return TB;
end.
```

Fig. 4 Algorithm for extracting template block.

```
Input : B = {b1, b2, .., bn} (blocks)
Output : {(x, y1), (x, y2)} (right edge of TBl)

procedure LeftBorder(DomTree)
begin
  // initialize
  y1 ← 0;
  y2 ← height of the web page;
  x ← 0;
  h ← 0;

  for each bi ∈ B do
    if bi touch left edge of the web page then
      if (width of bi)≤ (width of the web page) * 0.5 then
        x ← x + (width of bi) * (height of bi);
        h ← h+ (height of bi);
      else
        if (bottom coordinate of bi) ≤
              (height of the web page) * 0.5 then
          y1 ← max(y1, bottom coordinate of bi);
        else
          y2 ← min(y2, top coordinate of bi);
        endif
      endif
    endif
  enddo
end.
```

Fig. 5 Detecting right edge of template block TB_l.

The ***MatchingTemplates*** procedure judges which template (T_1–T_8) the Web page corresponds to on the basis of the divided blocks.

First, the right edge of TB_l is detected by the ***LeftBorder*** procedure (Fig. 5). This procedure calculates the average width of some of the blocks. The blocks touch the left edge of the Web page and their widths are narrower than half of the page.

However, if the total of the widths is smaller than the widths of all of the blocks that touch the left edge multiplied by the constant rate, the ***LeftBorder*** procedure judges that there is no TB_l on the page. The left edge of TB_r is detected in the same way.

Second, the method determines if the area between TB_l and TB_r or the next area is TB_c. Finally, the area above TB_l, TB_c, and TB is determined to be TB_h, and the area under them is determined to be TB_f. If the Web page has a T_1 or T_5 template then it does not have TB_l or TB_r. In this case, assuming that both TB_h and TB_f have wide blocks, the area above the bottom of the blocks—which is included in the T_y pixels at the top of the Web page and has a width wider than the T_x pixels—is detected as TB_h. TB_f is detected in the same way.

These processes are repeated until D_{size} is smaller than the threshold size T_{size} and the candidates for the layout template $T_{candidate}$ are made. Next, the ***DetectTemplate*** procedure sequentially judges the availability of a layout template in $T_{candidate}$ from the layout template that has many visual classes, like T_8. Assuming that TB_c is the largest on a general Web page, the availability is judged based on the height of TB_h, TB_c, and TB_f and the width of TB_l, TB_c, and TB_r. For example, if the height of TB_c is lower than (or approximately the same as) the height of TB_h, the layout template is invalid. The layout template not judged as invalid is determined to be the layout template for the Web page.

The main content might be in the center block TB_c, but then it might not. Layout template detection is useful to detect approximately where the main content is located.

3.2 Minimum block extraction

The methods proposed in the related works mentioned above divide a Web page into very small pieces at first and then assemble them into semantically cohesive pieces of Web content. Our method also divides pages into very small pieces and combines them. These small pieces, which we call “minimum blocks,” are divided on the basis of block-level elements defined by World Wide Web Consortium (W3C)[1]. All elements in a page are classified as either block-level elements or inline elements. When rendered visually, a block-level element secures a rectangle space and displays child nodes in the same area. The overall Web page layout is determined by these block-level elements, each of which fit inside another. We define a minimum block as any block-level element that does not contain other block-level elements within it. If a

[1] http://www.w3.org/

node is an inline element that is the sibling of a minimum block, we adopt it as a minimum block even though it is technically an inline element. Therefore, all nodes that are rendered in a Web page are eventually assigned to minimum blocks.

The four rules listed below are used to determine whether or not a node n_i is a block-level element.

Rule 1. If n_i is not a valid node, n_i is not a block-level element.
Rule 2. If n_i has a "block" display style, n_i is a block-level element.
Rule 3. If n_i is described in one of the following tags, n_i is a block-level element: p, blockquote, pre, div, noscript, hr, address, fieldset, legend, h1, h2, h3, h4, h5, h6, ul, ol, li, dl, dt, dd, table, caption, thead, tbody, colgroup, col, tr, th, td.
Rule 4. If n_i *does not match Rules 2 or 3,* n_i is not a block-level element.

When a Web browser renders a Web page, some of the nodes are not displayed. We call such nodes "invalid." If a DOM node has the four qualifications listed below, however, we consider it "valid."

(1) The size of the node is greater than one.
(2) The *x* and *y* coordinates on the lower right are over 0.
(3) The display style property of the node is not "none."
(4) The visibility style property of the node is not "hidden."

To go into more detail, with (1), the size of the node means the value after multiplying the pixel width of the node by the pixel height. With (2), the coordinate of the node is expressed by rectangular coordinates, which treat the Web page as a plane. The origin is then at the upper left of the Web page. The *x*-axis has values that increase from left to right, while the *y*-axis has values that increase from top to bottom. With (3), the display style property specifies the type of box an element should generate. Nodes that have been given "none" for the property are not displayed on a Web browser. With (4), the visibility style property defines whether or not the boxes generated by an element are displayed. Nodes that have been given "hidden" for the property are not displayed on a Web browser. However, an invisible box is still secured, and it affects the page layout.

Web designers can specify whether an element is inline or block-level by giving a display style property. An element whose tag is listed in Rule 3 can be an inline element when it has an "inline" value for the display style. We judge the display style in Rule 2 before we judge the tag in Rule 3.

This is how we divide a Web page into minimum blocks. Figure 6 shows a screenshot of a Google search engine result. The search query was "Nagoya Institute of Technology," and the top three hits are shown in the screenshot. Twelve minimum blocks, which are enclosed by solid lines, are in the screenshot.

Fig. 6 Example of twelve minimum blocks extracted from a Google search engine result.

3.3 Creating combined Web content blocks based on title blocks

Our method focuses on title blocks to organize minimum blocks into Web content blocks. As stated earlier, title blocks are minimum blocks that function as headlines for specific Web content. Web page designers assign a title block to each web content on a page to make for easy reading, and these title blocks can be used as separators to segment the different parts of a Web page.

In this study, we focused on the four title block characteristics listed below.

(1) A title block has few child nodes.
(2) A title block has a short text length.
(3) The width of a title block is greater than its height.
(4) The size of a title block is smaller than that of the block underneath it.

These four characteristics, as well as characteristics based on HTML tag names, are used to create the parameters for machine learning. We adopted the nine parameters listed below.

Param 1. Text node length.
Param 2. Size of text nodes/size of whole node.
Param 3. Size of image nodes/size of whole node.
Param 4. Width of the block/height of the block.
Param 5. Whether the size of a title block is smaller than the size of the block underneath it.
Param 6. Whether the block is written by <H1>, <H2>, <H3>, <H4>, <H5>, <H6>, or <DT>.
Param 7. Number of same HTML tags above continuity.
Param 8. Number of same HTML tags below continuity.
Param 9. Number of child nodes.

We used decision tree learning to classify minimum blocks because it has only two possible classifications—"title block" or "not"—and because the learning algorithm can avoid over-fitting due to branch cut. We show the results of experiments on the classifier in Section 4.

Figure 7 shows three patterns that can be used to assemble minimum blocks into Web content by using title blocks. "*tb*" refers to a title block and "*ntb*" to a non-title block. Our algorithm assembles an initial title block followed by consecutive non-title blocks below it. For example, in Fig. 7(a), four blocks (one title block "*tb*" and three non-title blocks "ntb_1," "ntb_2," and "ntb_3") are assembled into one Web content block. The "ntb_1" shown in Fig. 7(c) does not have a title block above it. In this case, the "ntb_1" that is above "tb_1" is not assembled. "tb_1," "ntb_2," and "ntb_3" are assembled in pattern (a).

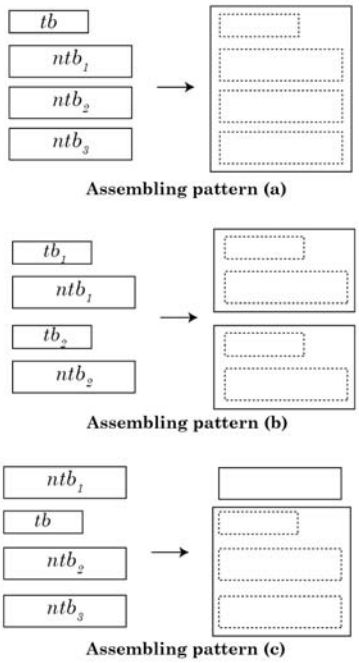

Fig. 7 Patterns to assemble smallest blocks into a Web content block by using title blocks.

4. Experimental results

4.1 Method

As stated above, our Web page segmentation method assembles minimum blocks into Web content based on title blocks. The higher the accuracy of the title block extraction, the higher the accuracy of the segmentation. We specified three classifiers for determining whether minimum blocks are title or non-title blocks. These classifiers were constructed by decision tree learning using a J48 algorithm, decision tree learning using a random tree algorithm, and a support vector machine. We measured the accuracy of the title block extraction by ten cross-validations while making the classifiers.

Fifty Web pages were used in the experiments: we did a Google search with the query word "summer" and used the top 50 pages from the search engine results. The minimum blocks in the Web pages were manually classified into title or non-title blocks and were then used as the training data for the machine learning. We conducted four counts: (*a*) how many times a title block was identified correctly, (*b*) how many times a non-title block was identified correctly, (*c*) how many times a title block was misjudged as a non-title block, and (*d*) how many times a non-title block was misjudged as a title block. The precision and re-call were given by the formula below.

$$P_{tb} = \frac{a}{a+d},\quad P_{ntb} = \frac{b}{b+c},\quad R_{tb} = \frac{a}{a+c},\quad R_{ntb} = \frac{b}{b+d},$$

where P_{tb} is the precision of the title block decision, P_{ntb} is the precision of the non-title block decision, R_{tb} is the re-call of the title block decision, and R_{ntb} is the recall of the non-title block decision.

F-measures F_{tb} and F_{ntb} are given by

$$F_{tb} = \frac{2\cdot P_{tb}\cdot R_{tb}}{P_{tb}+R_{tb}},\quad F_{ntb} = \frac{2\cdot P_{ntb}\cdot R_{ntb}}{P_{ntb}+R_{ntb}}$$

4.2 Results and discussion

Table 1: Precision and recall in extracting title blocks.

	J48		**Random tree**		**SVM**	
Random sampling	No	Yes	No	Yes	No	Yes
(*a*)	622	696	660	710	376	756
(*b*)	7429	1779	7356	1747	7493	1697
(*c*)	243	169	205	155	489	109
(*d*)	141	136	214	168	77	218
P_{tb}	0.815	0.837	0.755	0.809	0.830	0.776
R_{tb}	0.719	0.805	0.763	0.821	0.435	0.874
F_{tb}	0.764	0.821	0.759	0.815	0.571	0.822

P_{ntb}	0.968	0.913	0.973	0.919	0.939	0.940
R_{ntb}	0.981	0.929	0.972	0.912	0.990	0.886
F_{ntb}	0.974	0.921	0.972	0.915	0.964	0.912

Table 1 shows the experimental results. We used 8435 minimum blocks for learning modules as training data. 865 were title blocks and 7570 were non-title blocks. The best F-measure for title block decisions—76.4%—was with the J48 algorithm, which is a poor result, and the worst—57.1%—was with the support vector machine. The re-call of title block decisions with the proposed algorithm was 43.5%. All these values are very low, mostly because there were fewer title blocks than non-title blocks. When these blocks were used as training data, the boundary plane was partial to non-title blocks, which meant that the re-call of title block decisions was likely to be low.

To improve the re-call, we decided to sample non-title blocks randomly and to set the ratio of title blocks and non-title blocks to 1:2. We used 865 title blocks and 1915 non-title blocks as training data. After these changes were made, the F-measures of title block decisions were over 80% with all three algorithms. This demonstrates that it is appropriate to sample non-title blocks randomly.

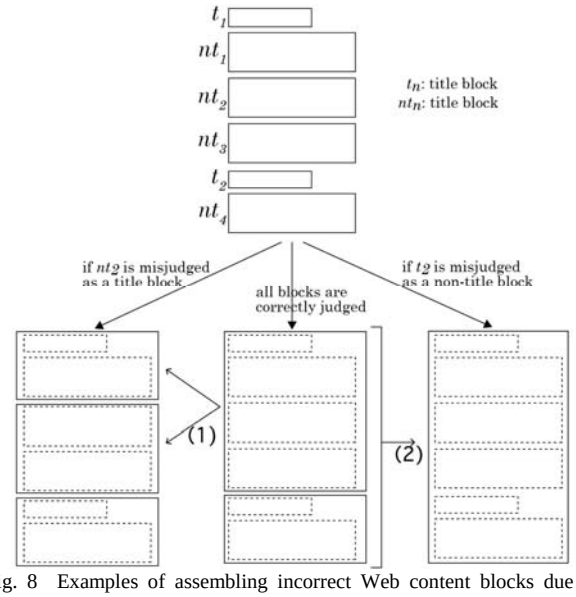

Fig. 8 Examples of assembling incorrect Web content blocks due to errors in identifying title/non-title blocks.

If a non-title block is misjudged as a title block, that block is used to divide a Web content block. For example, in Fig. 8 (1), the non-title block "nt_1" was judged as a title block. One Web content block was divided into two Web content blocks as a result of this error. In the case of a title block being misjudged as a non-title block, two Web content blocks are assembled into one Web content block, as shown in Fig. 8 (2). Visitors to a Web site judge content blocks that have essentially been assembled into one block as an incorrect segmentation. However, they do not judge one Web content block that has essentially been divided into various other blocks as incorrect. Neither of the results shown in Fig. 8 (1) and (2) are perfect, but the result in (2) is better for the reasons mentioned above. This highlights the importance of not judging non-title blocks as title blocks. In other words, we ought to use the algorithm that can get a higher precision for deciding title blocks P_{tb} and a higher re-call for deciding non-title blocks R_{ntb}.

Table 1 shows that the decision tree created by the J48 algorithm attained an 83.7% P_{tb} and a 92.9% R_{ntb}. These were the best results of the three algorithms, which indicates the J4.8 algorithm's suitability for extracting title blocks for Web page segmentation.

5. Conclusion

We proposed a Web page segmentation method that assembles minimum blocks based on title blocks. The proposed method is most effective when it obtains a high precision of deciding title blocks and a high re-call of deciding non-title blocks. We tested three algorithms, and the decision tree created using the J48 algorithm obtained an 83.7% P_{tb} and a 92.9% R_{ntb}, indicating that it is the best algorithm for extracting title blocks for Web page segmentation.

Acknowledgement

Part of this work was supported by KAKENHI (22500128) and Strategic Information and Communications R&D Promotion Programme of the Ministry of Internal Affairs and Communications, Japan.

References

[1] S. Baluja. Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework. In Proceedings of the 15th international conference on World Wide Web, WWW '06, pages 33–42, New York, NY, USA, 2006. ACM.

[2] P. Baudisch, X. Xie, C. Wang, and W.-Y. Ma. Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content. In Proceedings of the 17th annual ACM symposium on User interface software and technology, UIST '04, pages 91–94, New York, NY, USA, 2004. ACM.

[3] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Accordion summarization for end-game browsing on pdas and cellular phones, pages 213–220, 2001.

[4] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma, VIPS: a vision-based page segmentation algorithm, 2003.

[5] Y. Chen, W.-Y. Ma, and H.-J. Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In Proceedings of the 12th international conference on World Wide Web, WWW '03, pages 225–233, New York, NY, USA, 2003. ACM.
[6] S.-H. Lin and J.-M. Ho. Discovering informative content blocks from web documents. In Proceedings of ACM SIGKDD '02, pages 588–593, 2002.