VIPS: a Vision-based Page Segmentation Algorithm

Deng Cai
Shipeng Yu
Ji-Rong Wen
Wei-Ying Ma

Nov. 1, 2003

Technical Report
MSR-TR-2003-79

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

VIPS: a Vision-based Page Segmentation Algorithm

Deng Cai[1*], Shipeng Yu[2*], Ji-Rong Wen[*] and Wei-Ying Ma[*]

[*]Microsoft Research Asia
5F, Beijing Sigma Center
No.49, Zhichun Road Haidian District
Beijing, P.R.China
{jrwen, wyma}@microsoft.com

[1]Tsinghua University
Beijing, P.R.China
cai_deng@yahoo.com

[2]Peking University
Beijing, P.R.China
ysp@is.pku.edu.cn

VIPS: a Vision-based Page Segmentation Algorithm

Abstract

A new web content structure analysis based on visual representation is proposed in this paper. Many web applications such as information retrieval, information extraction and automatic page adaptation can benefit from this structure. This paper presents an automatic top-down, tag-tree independent approach to detect web content structure. It simulates how a user understands web layout structure based on his visual perception. Comparing to other existing techniques, our approach is independent to underlying documentation representation such as HTML and works well even when the HTML structure is far different from layout structure. Experiments show satisfactory results.

1 Introduction

Today the Web has become the largest information source for people. Most information retrieval systems on the Web consider web pages as the smallest and undividable units, but a web page as a whole may not be appropriate to represent a single semantic. A web page usually contains various contents such as navigation, decoration, interaction and contact information, which are not related to the topic of the web-page. Furthermore, a web page often contains multiple topics that are not necessarily relevant to each other. Therefore, detecting the semantic content structure of a web page could potentially improve the performance of web information retrieval.

Many web applications can utilize the semantic content structures of web pages. For example, in web information accessing, to overcome the limitations of browsing and keyword searching, some researchers have been trying to use database techniques and build wrappers to structure the web data [1-3, 18]. In building wrappers, it is necessary to divide the web documents into different information chunks [2, 17]. Previous work uses ad hoc methods to deal with different types of web pages. If we can get a semantic content structure of the web page, wrappers can be more easily built and information can be more easily extracted. Moreover, Link analysis has received much attention in recent years. Traditionally different links in a page are treated identically. The basic assumption of link analysis is that if there is a link

between two pages, there is some relationship between the two *whole* pages. But in most cases, a link from page A to page B just indicates that there might be some relationship between some certain *part* of page A and some certain *part* of page B. Also, the existence of large quantify of noisy links will cause the topic drift problem in HITS algorithm [7, 20]. Recent works on topic distillation [11, 12] and focused crawling [13] strengthen our observation. However, these works are based on DOM (Document Object Model)[1] tree of the web page which has no sufficient power to semantically segment the web page as we show in the experimental section. Furthermore, efficient browsing of large web pages on small handheld devices also necessitates semantically segmentation of web pages [19].

Much recent work [11, 13, 14, 17] try to extract the structure information from HTML DOM tree. However, because of the flexibility of HTML syntax, a lot of web pages do not obey the W3C html specifications, which might cause mistakes in DOM tree structure. Moreover, DOM tree is initially introduced for presentation in the browser rather than description of the semantic structure of the web page. For example, even though two nodes in the DOM tree have the same parent, it might not be the case that the two nodes are more semantically related to each other than to other nodes. Two examples are shown in the experimental section. To provide a better description of the semantic structure of the web page content, XML[2] is introduced. However, as we can observe, the majority of the web pages are written in HTML rather than XML.

In the sense of human perception, it is always the case that people view a web page as different semantic objects rather than a single object. Some research efforts show that users always expect that certain functional part of a web page (e.g. navigational links, advertisement bar) appears at certain position of that page [6]. Actually, when a web page is presented to the user, the *spatial and visual cues* can help the user to unconsciously divide the web page into several semantic parts. Therefore, it might be possible to automatically segment the web pages by using the spatial and visual cues. Throughout this paper, we use *block* to denote the semantic part of the web page.

[1] http://www.w3c.org/DOM/

[2] http://www.w3c.org/XML/

In this paper, we propose VIPS (**VI**sion-based **P**age **S**egmentation) algorithm to extract the semantic structure for a web page. Such semantic structure is a hierarchical structure in which each node will correspond to a block. Each node will be assigned a value (*Degree of Coherence*) to indicate how coherent of the content in the block based on visual perception. The VIPS algorithm makes full use of page layout feature: it first extracts all the suitable blocks from the html DOM tree, then it tries to find the separators between these extracted blocks. Here, separators denote the horizontal or vertical lines in a web page that visually cross with no blocks. Finally, based on these separators, the semantic structure for the web page is constructed. VIPS algorithm employs a top-down approach, which is very effective.

The rest of the paper is organized as follows. Section 2 provides an overview on the related works. In Section 3, we define the web page content structure based on vision. The VIPS algorithm is introduced in Section 4. The experimental results are shown in Section 5. Finally, we give concluding remarks in Section 6.

2 Related Works

The applications mentioned in the introduction indicate the need of techniques for extracting the content structure of a web page. Many researchers have considered using the tag information and dividing the page based on the type of the tags. Useful tags include <P> (paragraph), <TABLE> (table), <UL> (list), <H1>~<H6> (heading), etc. Diao [15] treats segments of web pages in a learning based web query processing system and deals with these major types of tags. Lin [21] only considers <TABLE> tag and its offspring as a content block and uses entropy based approach to discover informative ones. For adaptive content delivery, Kaasinen [19] and Buyukkokten [10] split the web page by some easy tags such as <P>, <TABLE> and <UL> for further conversion or summarization. Wong [26] defines tag types for page segmentation and gives a label to each part of the web page for classification.

Besides the tag tree, some other algorithms also make use of the content or link information. Embley [17] and Buttler [9] use some heuristic rules to discover record boundaries within a page, which assist data extraction from the web page. Chakrabarti [11] [12] addresses the fine-grained topic distillation and dis-aggregates hubs into regions by analyzing link structure as well as intra-page text distribution. Recently, Bar-Yossef [5] proposes the template detection problem and presents an easy

algorithm only based on link information. Also, Rahman gives some comments on page decomposition in [22] but does not show details on the technique.

A Function-based Object Model (FOM) of a web page is proposed by Chen [14] for content understanding and adaptation. Every undividable element in the tag tree is called a basic object and can be grouped into a composite object. A function type can be defined to each object and helps to build a hierarchical structure for the page. However, the grouping rules and the functions are hard to define accurately, and thus make the whole tree-constructing process very inflexible.

All of the above methods fail to take into account visual structure of the web page. In this paper, we propose a new algorithm, call VIPS (VIsion-based Page Segmentation), to semantically segment the web page. In the next section, we first describe our defined vision-based content structure for web pages.

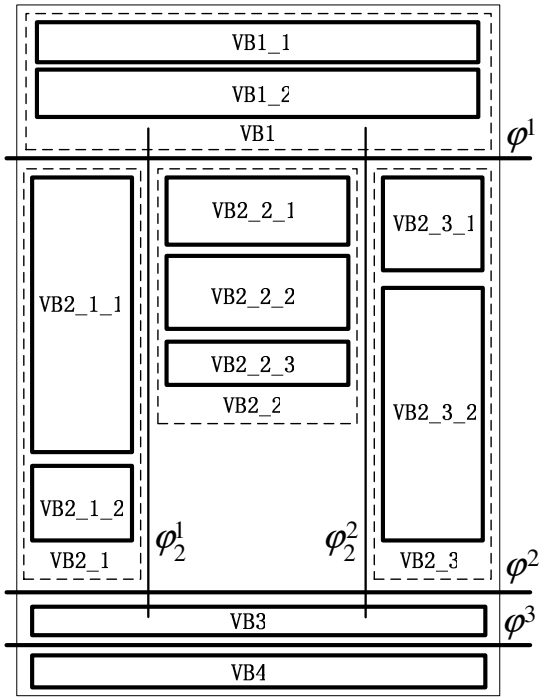
3 Vision-based Content Structure for Web Pages

Similar to [14], we define the *basic object* as the leaf node in the DOM tree that can not be decomposed any more. In this paper, we propose the vision-based content structure, where every node, called a *block*, is a basic object or a set of basic objects. It is important to note that, the nodes in the vision-based content structure do not necessarily correspond to the nodes in the DOM tree.

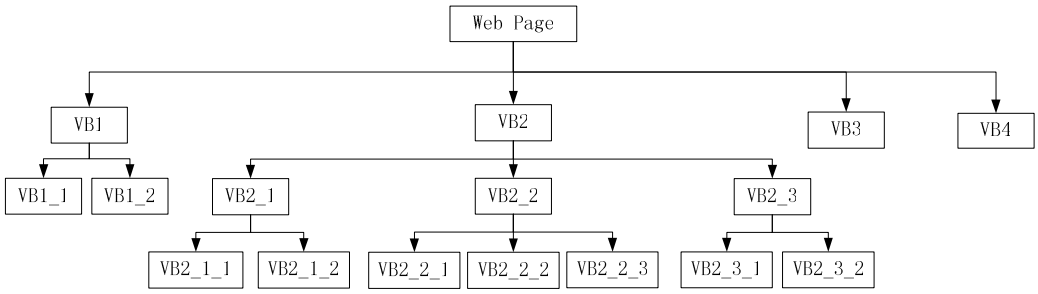
Similar to the description of document representation in [25], the basic model of *vision-based content structure* for web pages is described as follows.

A web page Ω is represented as a triple $\Omega = (O, \Phi, \delta)$. $O = \{\Omega^1, \Omega^2, ..., \Omega^N\}$ is a finite set of blocks. All these blocks are not overlapped. Each block can be recursively viewed as a sub-web-page associated with sub-structure induced from the whole page structure. $\Phi = \{\varphi^1, \varphi^2, ..., \varphi^T\}$ is a finite set of separators, including horizontal separators and vertical separators. Every separator has a weight indicating its visibility, and all the separators in the same Φ have the same weight. δ is the relationship of every two blocks in O and can be expressed as: $\delta = O \times O \to \Phi \cup \{NULL\}$. For example, suppose Ω_i and Ω_j are two objects in O, $\delta(\Omega_i, \Omega_j) \neq NULL$ indicates that Ω_i and Ω_j are exactly separated by the separator $\delta(\Omega_i, \Omega_j)$ or we can say the two objects are adjacent to each other, otherwise there are other objects between the two blocks Ω_i and Ω_j.

(a)



(b)



(c)

$$O = (VB1, VB2, VB3, VB4)$$

$$\Phi = \{\varphi^1, \varphi^2, \varphi^3\}$$

$$\delta \begin{pmatrix} (VB1, VB2) \\ (VB2, VB3) \\ (VB3, VB4) \\ else \end{pmatrix} = \begin{pmatrix} \varphi^1 \\ \varphi^2 \\ \varphi^3 \\ NULL \end{pmatrix}$$

(d)

$$VB2 = (VB2\_1, VB2\_2, VB2\_3)$$

$$\Phi^2 = \{\varphi_2^1, \varphi_2^2\}$$

$$\delta^2 \begin{pmatrix} (VB2\_1, VB2\_2) \\ (VB2\_2, VB2\_3) \\ else \end{pmatrix} = \begin{pmatrix} \varphi_2^1 \\ \varphi_2^2 \\ NULL \end{pmatrix}$$

(e)

Figure 1. The layout structure and vision-based content structure of an example page. (d) and (e) show the corresponding specification of vision-based content structure.

Since each Ω_i is a sub-web-page of the original page, it has similar content structure as Ω. Recursively, we have $\Omega_s^t = (O_s^t, \Phi_s^t, \delta_s^t)$, $O_s^t = \{\Omega_{st}^1, \Omega_{st}^2, ..., \Omega_{st}^{N_{st}}\}$,

$\Phi_s^t = \{\varphi_{st}^1, \varphi_{st}^2, ..., \varphi_{st}^{T_{st}}\}$ and $\delta_s^t = O_s^t \times O_s^t \to \Phi_s^t \cup \{NULL\}$ where Ω_s^t is the t^{th} object in the sub-web-page level s, N_{st} and T_{st} are the number of objects in O_s^t and number of separators in Φ_s^t.

Figure 1 shows an example of vision-based content structure for a web page of Yahoo! Auctions (http://list.auctions.shopping.yahoo.com/21600-category.html?alocale=0us). It illustrates the layout structure and the vision-based content structure of the page. In the first level, the original web page has four objects or visual blocks VB1~VB4 and three separators $\varphi^1 \sim \varphi^3$, as specified in Figure 1(d). Then we can further construct sub content structure for each sub web page. For example, VB2 has three offspring objects and two separators. It can be further analyzed as shown in Figure 1(e).

For each visual block, the *Degree of Coherence* (DoC) is defined to measure how coherent it is. DoC has the following properties:

- The greater the DoC value, the more consistent the content within the block;
- In the hierarchy tree, the DoC of the child is not smaller than that of its parent.

In our algorithm, DoC values are integers ranging from 1 to 10, although alternatively different ranges (e.g., real numbers, etc.) could be used.

We can pre-define the *Permitted Degree of Coherence* (PDoC) to achieve different granularities of content structure for different applications. The smaller the PDoC is, the coarser the content structure would be. For example in Figure 1(a), the visual block VB2_1 may not be further partitioned with an appropriate PDoC. Different application can use VIPS to segment web page to a different granularity with proper PDoC.

The vision-based content structure is more likely to provide a semantic partitioning of the page. Every node of the structure is likely to convey certain semantics. For instance, in Figure 1(a) we can see that VB2_1_1 denotes the category links of Yahoo! Shopping auctions, and that VB2_2_1 and VB2_2_2 show details of the two different comics.

4 The VIPS Algorithm

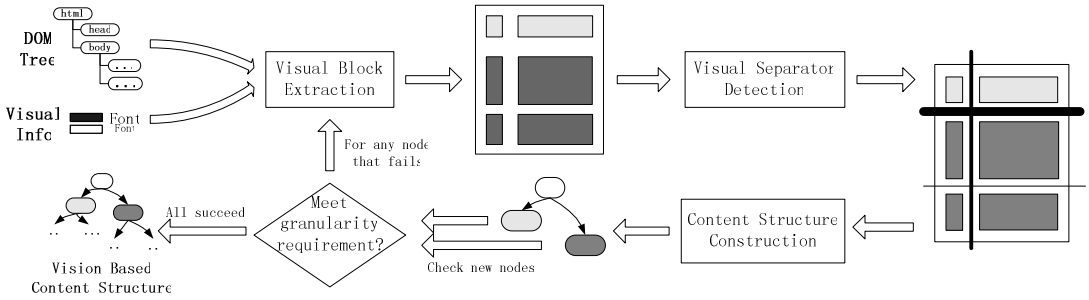
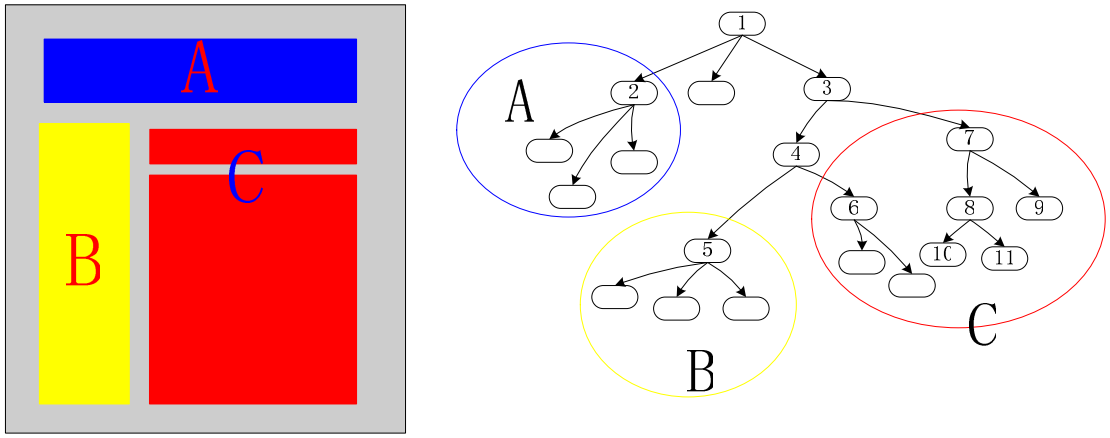


Figure 2. The vision-based page segmentation algorithm

In this section, we introduce our VIPS algorithm. Basically, the vision-based content structure of a page is obtained by combining the DOM structure and the visual cues. The segmentation process is illustrated in Figure 2. It has three steps: block extraction, separator detection and content structure construction. These three steps as a whole are regarded as a round. The algorithm is top-down. The web page is firstly segmented into several big blocks and the hierarchical structure of this level is recorded. For each big block, the same segmentation process is carried out recursively until we get sufficiently small blocks whose DoC values are greater than pre-defined PDoC.

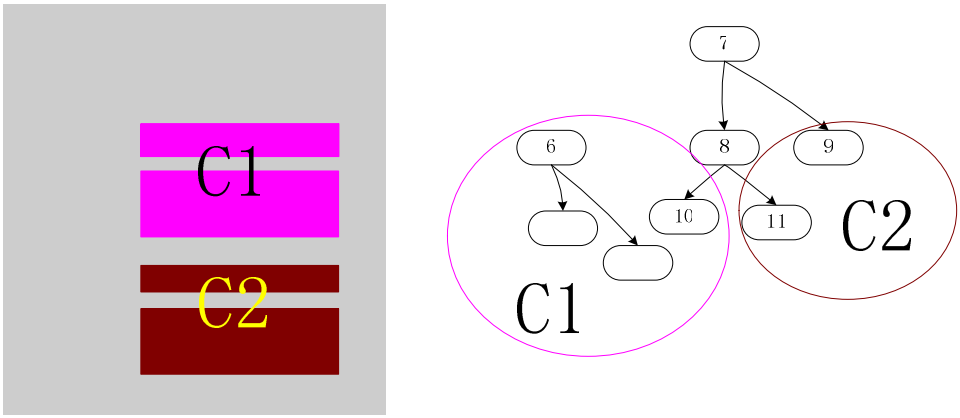


(a) (b)

Figure 3. (a) Page Layout (b) DOM tree of the page

For each round, the DOM tree with its visual information corresponded to the current block (page for the first round) is obtained from a web browser, as we show in Figure 3. Then, from the root node(s) of the DOM tree (e.g. DOM node 1 in Figure 3b), the block extraction process is started to extract blocks from the DOM tree based on visual cues. Every DOM node (node 1, 2, 3, 4, 5, 6, 7 as shown in figure 3b) is checked

to judge whether it forms a single block or not. If not (node 1, 3, 4 in figure 3b), its children will be processed in the same way. We will assign a DoC value to each extracted block (node 2, 5, 6, 7 in figure 3b) based on the block's visual property. When all blocks of the current round in current page or sub-page are extracted, they are put into a pool. Separators among these blocks are identified and the weight of a separator is set based on properties of its neighboring blocks. The layout hierarchy was constructed based on these separators. After constructing the layout hierarchy of the current round, each leaf node of the content structure is checked to see whether or not it meets the granularity requirement. If not, this leaf node will be treated as a sub-page and will be further segmented similarly. For example, if the Block C in figure 3 does not meet the requirement, we treat this block as a sub-page and it will be further segmented into two parts, C1 and C2, as shown in Figure 4a, 4b.



(a) (b)

Figure 4. (a) Layout of the block C (b) DOM tree of block C

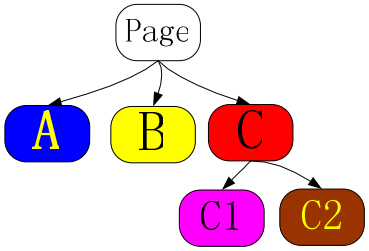


Figure 5. Vision-based content structure

After all blocks are processed, the final vision-based content structure for the web page is outputted. In the above example, we finally obtain a vision-based content structure

tree as shown in Figure 5. In the following three subsections, we will describe the block extraction, separator detection and content structure construction, respectively.

4.1 Visual Block Extraction

In this step, we aim at finding all appropriate visual blocks contained in the current sub-page. In general, every node in the DOM tree can represent a visual block. However, some "huge" nodes such as <TABLE> and <P> are used only for organization purpose and are not appropriate to represent a single visual block. In these cases, the current node should be further divided and replaced by its children. Due to the flexibility of HTML grammar, many web pages do not fully obey the W3C HTML specification, so the DOM tree can not always reflect the true relationship of the different DOM node.

For each extracted node that represents a visual block, its DoC value is set according to its intra visual difference. This process is iterated until all appropriate nodes are found to represent the visual blocks in the current sub-page.

```
Algorithm DivideDomtree(pNode, nLevel)
{
  IF (Dividable(pNode, nLevel) == TRUE){
    FOR EACH child OF pNode {
        DivideDomtree(child, nLevel);
    }
  } ELSE {
    Put the sub-tree (pNode) into the
        pool as a block;
  }
}
```

Figure 6. The visual block extraction algorithm

We judge if a DOM node can be divided based on following considerations:

- The properties of the DOM node itself. For example, the HTML tag of this node, the background color of this node, the size and shape of this block corresponding to this DOM node.
- The properties of the children of the DOM node. For example, the HTML tags of children nodes, background color of children and size of the children. The number of different kinds of children is also a consideration.

Based on WWW html specification 4.01[1], we classify the DOM node into two categories, *inline node* and *line-break node*:

- *Inline node*: the DOM node with inline text HTML tags, which affect the appearance of text and can be applied to a string of characters without introducing line break. Such tags include <B>, <BIG>, <EM>, <FONT>, <I>, <STRONG>, <U>, etc.
- *Line-break Node*: the node with tag other than inline text tags.

Based on the appearance of the node on the browser and the children properties of the node, we give some definitions:

- *Valid node*: a node that can be seen through the browser. The node's width and height are not equal to zero. (For example the second and fourth child of <TR> is not a valid node in Figure 7)
- *Text node*: the DOM node corresponding to free text, which does not have an html tag.
- *Virtual text node (recursive definition)*:
  - *Inline node* with only *text node* children is a *virtual text node*.
  - *Inline node* with only *text node* and *virtual text node* children is a *virtual text node*.

The visual block extraction algorithm DivideDomtree is illustrated in Figure 6. Some important cues which are used to produce heuristic rules in the algorithm are:

- Tag cue:
  1. Tags such as <HR> are often used to separate different topics from visual perspective. Therefore we prefer to divide a DOM node if it contains these tags.
  2. If an *inline node* has a child which is *line-break node*, we divide this inline node.
- Color cue: We prefer to divide a DOM node if its background color is different from one of its children's. At the same time, the child node with different background color will not be divided in this round.

[1] http://www.w3.org/TR/html4/

- Text cue: If most of the children of a DOM node are text nodes or virtual text node, we prefer not to divide it.
- Size cue: We predefine a relative size threshold (the node size compared with the size of the whole page or sub-page) for different tags (the threshold varies with the DOM nodes having different HTML tags). If the relative size of the node is small than the threshold, we prefer not to divide the node.

Based on these cues, we can produce heuristic rules to judge if a node should be divided. If a node should not be divided, a block is extracted and we will set the DoC value for this block. We list the heuristic rules in Table 1 by their priority.

Table 1. Heuristic rules in block extraction phase

Rule 1	If the DOM node is not a text node and it has no valid children, then this node cannot be divided and will be cut.
Rule 2	If the DOM node has only one valid child and the child is not a text node, then divide this node.
Rule 3	If the DOM node is the root node of the sub-DOM tree (corresponding to the block), and there is only one sub DOM tree corresponding to this block, divide this node.
Rule 4	If all of the child nodes of the DOM node are text nodes or virtual text nodes, do not divide the node. ● If the font size and font weight of all these child nodes are same, set the DoC of the extracted block to 10. ● Otherwise, set the DoC of this extracted block to 9.
Rule 5	If one of the child nodes of the DOM node is line-break node, then divide this DOM node.
Rule 6	If one of the child nodes of the DOM node has HTML tag <HR>, then divide this DOM node
Rule 7	If the sum of all the child nodes' size is greater than this DOM node's size, then divide this node.
Rule 8	If the background color of this node is different from one of its children's, divide this node and at the same time, the child node with different background color will not be divided in this round. ● Set the DoC value (6-8) for the child node based on the html tag of the child node and the size of the child node.
Rule 9	If the node has at least one text node child or at least one virtual text node child, and the node's relative size is smaller than a threshold, then the node cannot be divided

	• Set the DoC value (from 5-8) based on the html tag of the node
Rule 10	If the child of the node with maximum size are small than a threshold (relative size), do not divide this node. • Set the DoC based on the html tag and size of this node.
Rule 11	If previous sibling node has not been divided, do not divide this node
Rule 12	Divide this node.
Rule 13	Do not divide this node • Set the DoC value based on the html tag and size of this node.

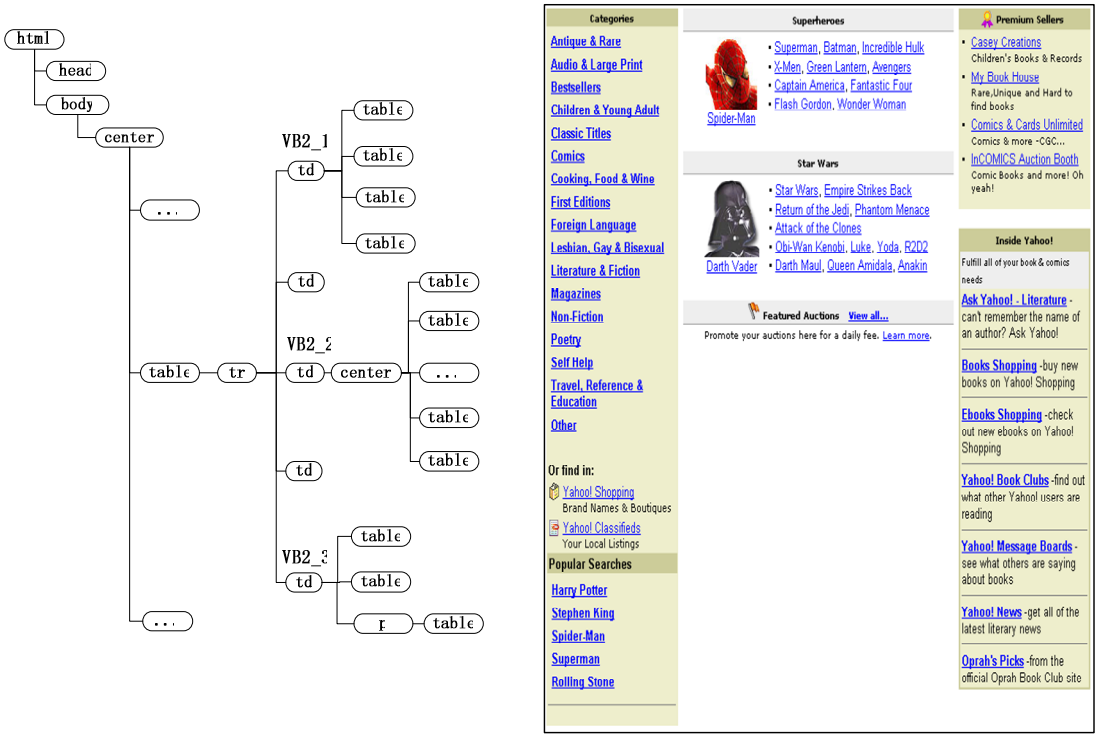
For different DOM nodes with different HTML tags, we will apply different rules. We listed them in Table 2.

Table 2. Different rules for different DOM nodes

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
Inline Text Node	√	√	√	√	√	√	√		√	√		√	
<TABLE>	√	√	√					√		√			√
<TR>	√	√	√				√	√		√			√
<TD>	√	√	√	√					√	√	√		√
<P>	√	√	√	√	√	√	√		√	√		√	
Other Tags	√	√	√	√		√	√		√	√		√	

Let us consider an example shown in Figure 1. At the first round of block extraction, VB1, VB2_1, VB2_2, VB2_3, VB3 and VB4 will be extracted and put into the pool. Below we explain how the VB2_1, VB2_2 and VB2_3 are extracted in details. Figure 7(b) is a table, which is a part of the whole web page. Its DOM tree structure is shown on the left side. In the block extraction process, when the <TABLE> node is met, it has only one valid child <TR>. We trace into the <TR> node according to the rule 2. The <TR> node has five <TD> children and only three of them are valid. The first child's background color is different from its parent's background color. According to the rule 8, the <TR> node is split and the first <TD> node is not divided further in this round and is put into the pool as a block. The second and fourth child of <TR> node is not valid and will be cut. According to the rule 11, the third and fifth children of <TR> will

not be divided in this round. Finally, we obtain three blocks, VB2_1, VB2_2, and VB2_3.



(a) (b)

Figure 7. Visual block extraction of a sample page

4.2 Visual Separator Detection

After all blocks are extracted, they are put into a pool for visual separator detection. Separators are horizontal or vertical lines in a web page that visually cross with no blocks in the pool. From a visual perspective, separators are good indicators for discriminating different semantics within the page. A visual separator is represented by a 2-tuple: (P_s, P_e), where P_s is the start pixel and P_e is the end pixel. The width of the separator is calculated by the difference between these two values.

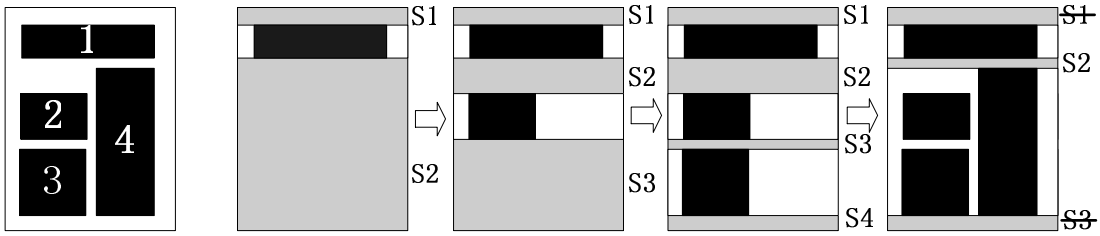
4.2.1 Separator Detection

The visual separator detection algorithm is described as follows:

1. Initialize the separator list. The list starts with only one separator (P_{be}, P_{ee}) whose start pixel and end pixel are corresponding to the borders of the pool.

2. For every block in the pool, the relation of the block with each separator is evaluated
   a) If the block is contained in the separator, split the separator;
   b) If the block crosses with the separator, update the separator's parameters;
   c) If the block covers the separator, remove the separator.
3. Remove the four separators that stand at the border of the pool (e.g. S1 and S3 in figure 8).

Take Figure 8(a) as an example in which the black blocks represent the visual blocks in the page. For simplicity we only show the process to detect the horizontal separators. At first we have only one separator that is the whole pool. As shown in Figure 8(b), when we put the first block into the pool, it splits the separator into S1 and S2. It is the same with the second and third block. When the fourth block is put into the pool, it crosses the separator S2 and covers the separator S3, the parameter of S2 is updated and S3 is removed. At the end of this process, the two separators S1 and S3 that stand at the border of the pool are removed.



(a) (b)

Figure 8. A sample page and the separator detection process

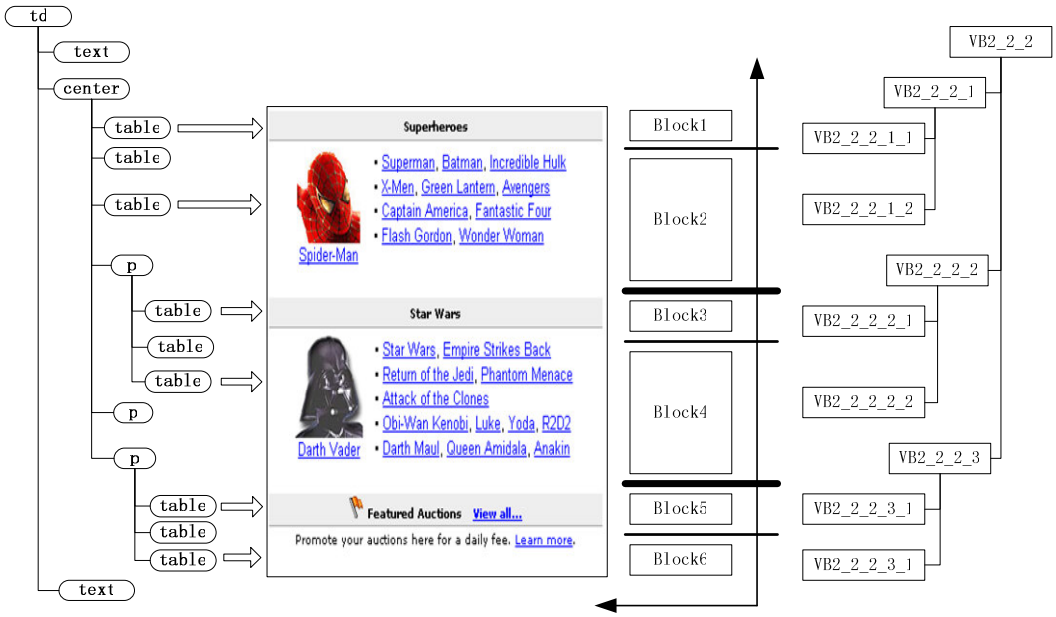
4.2.2 Setting Weights for Separators

The separators are used to distinguish blocks with different semantics, so the weight of a separator can be assigned based on the visual difference between its neighboring blocks. The following rules are used to set a weight to each separator:

- The greater the distance between blocks on different side of the separator, the higher the weight.
- If a visual separator is overlapped with some certain HTML tags (e.g., the <HR> HTML tag), its weight is set to be higher.
- If background colors of the blocks on two sides of the separator are different, the weight will be increased.

- For horizontal separators, if the differences of font properties such as font size and font weight are bigger on two sides of the separator, the weight will be increased. Moreover, the weight will be increased if the font size of the block above the separator is smaller than the font size of the block below the separator.
- For horizontal separators, when the structures of the blocks on the two sides of the separator are very similar (e.g. both are text), the weight of the separator will be decreased.

Take the third <TD> node in Figure 7 as an example. The sub-page corresponding to this node is shown in Figure 9(b) and the DOM tree structure is shown in Figure 9(a). We can see that many nodes in the DOM tree are invalid in our definition and can not be seen in the browser. They are ignored in the block extraction process. After the block extraction step, six blocks are put in a pool and five horizontal separators are detected. Then the weights of these separators are set based on the above five rules. In this example, the separator between Block 2 and Block 3 will get higher weight than the separator between Block 1 and Block 2 because of the different font weights. For the same reason, the separator between Block 4 and Block 5 will also get a high weight. The final separators and weights are shown in Figure 9(c), in which a thicker line means a higher weight.



(a) (b) (c) (d)

Figure 9. Separators and their weights

4.3 Content Structure Construction

When separators are detected and separators' weights are set, the content structure can be constructed accordingly. The construction process starts from the separators with the lowest weight and the blocks beside these separators are merged to form new blocks. This merging process iterates till separators with maximum weights are met. The DoC of each new block is set based on the maximum weight of the separators in the block's region.

After that, each leaf node is checked whether it meets the granularity requirement. For every node that fails, we go to the Visual Block Extraction step again to further construct the sub content structure within that node. If all the nodes meet the requirement, the iterative process is then stopped and the vision-based content structure for the whole page is obtained. The common requirement for DoC is that DoC > PDoC, if PDoC is pre-defined.

Take Figure 9 as an example. In the first iteration, the first, third and fifth separators are chosen and Block 1 and 2 are merged to form the new block VB2_2_2_1. Similar merging is conducted for Block 3 and 4 (resulting in a new block VB2_2_2_2) and Block 5 and 6 (resulting in a new block VB2_2_2_3). The new blocks VB2_2_2_1, VB2_2_2_2 and VB2_2_2_3 are the children of VB2_2_2 and can also be viewed as a partition of VB2_2_2. Every leaf node, such as VB2_2_2_1_1, VB2_2_2_1_2 and VB2_2_2_2_1, will be checked to see whether it meets the granularity requirement. After several iterations, the final vision-based content structure of the page is constructed.

In summary, the proposed VIPS algorithm takes advantage of visual cues to obtain the vision-based content structure of a web page and thus successfully bridges the gap between the DOM structure and the semantic structure. The page is partitioned based on visual separators and structured as a hierarchy. This semantic hierarchy is consistent with human perception to some extent. VIPS is also very efficient. Since we trace down the DOM structure for visual block extraction and do not analyze every basic DOM node, the algorithm is totally top-down.

5 Experiments

In this section, we will first show two example pages, comparing our VIPS result and the DOM tree structure. Then we provide some performance evaluation of our proposed VIPS algorithm based on a large collection of web pages from Yahoo. We also conduct experiments to evaluate how the algorithm can be used to enhance information retrieval on the Web.

5.1 Simple examples of web page segmentation

In this sub-section, several simple examples of web page segmentation are presented to give people an intuition how our VIPS algorithm works. In the meantime, we show the DOM tree of these pages. We can clearly find that we can not get the right structure of the blocks only based on naïve DOM tree. Moreover, it is hard for us to decide which node conveys semantic meanings and where we should stop in different applications.

Figure 10 shows our VIPS result on a sample page[1]. The left part shows the page with different regions (different visual blocks in VIPS algorithms) marked with rectangles. The block marked with red rectangle is the block VB1-2-2-1. The upper right part shows the vision-based content structure of this page, while the lower one shows some statistics of selected visual block.

From the right VIPS tree, we can know the hierarchical structure of different blocks. In different applications, we can control the partition granularity by setting PDoC, as shown in the northeast corner. The DoC value of each block is shown behind the node name (in parenthesis) in the right VIPS tree.

Although from page layout we see that VB1-2-1, VB1-2-2-1 and VB1-2-2-1 are parallel, in our heuristic rules the separator between VB1-2-1 and VB1-2-2-1 will get higher weight than the separator between VB1-2-2-1 and VB1-2-2-1 because of the different background colors. So VB1-2-2-1 and VB1-2-2-2 will be merged to VB1-2-2, paralleled to VB1-2-1.

[1] http://standards.ieee.org

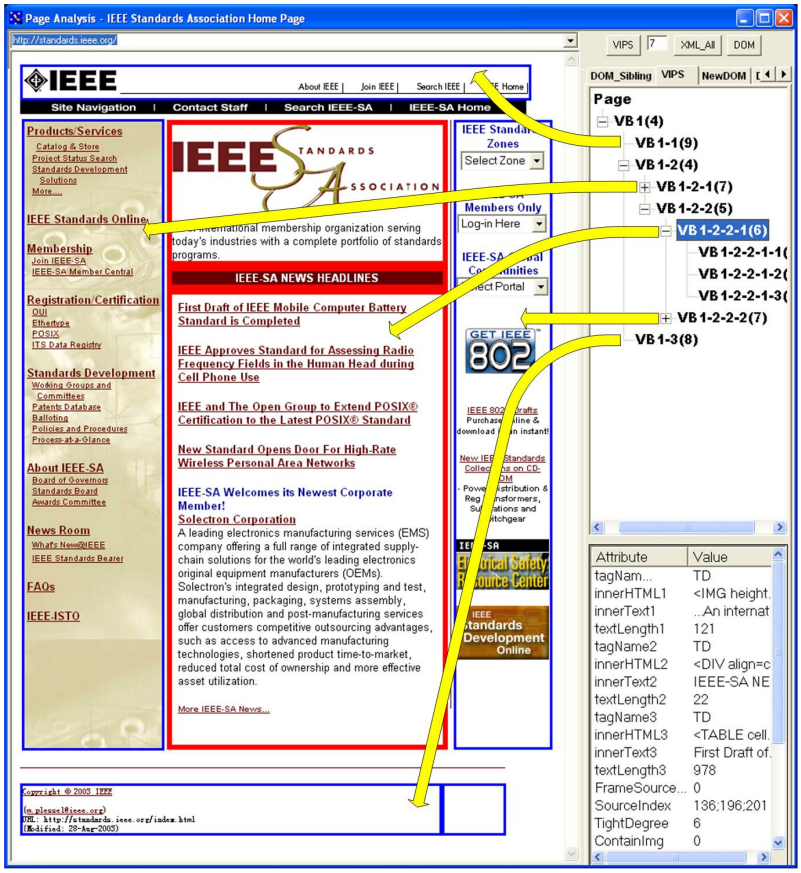


Figure 10. VIPS segmentation result of an IEEE page

Figure 11. DOM tree Partition result of an IEEE page

For comparison, we show the DOM tree and its corresponding blocks in Figure 11. We can see that the area marked with red line is a <TR> node (with three <TD> children). From visual perspective, these <TD> nodes should not be grouped together, but we can not get this information from DOM tree structure, while in VIPS this problem can be solved with the spatial and visual information of these blocks. We got the right content structure using our VIPS algorithm.

Take another example web page[1]. We show the DOM tree structure in Figure 12 and VIPS segmentation result in Figure 13.

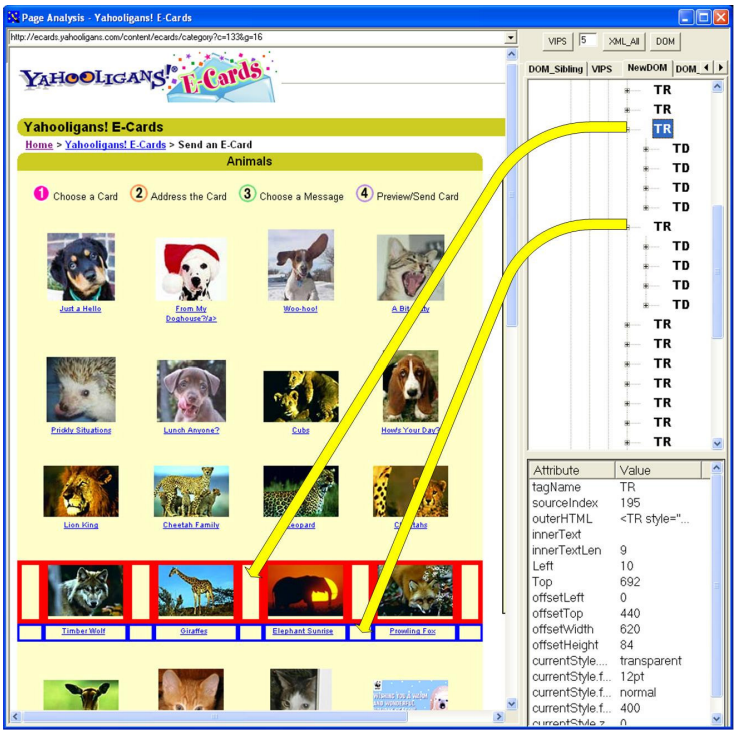


Figure 12. DOM tree structure

[1] http://ecards.yahooligans.com/content/ecards/category?c=133&g=16

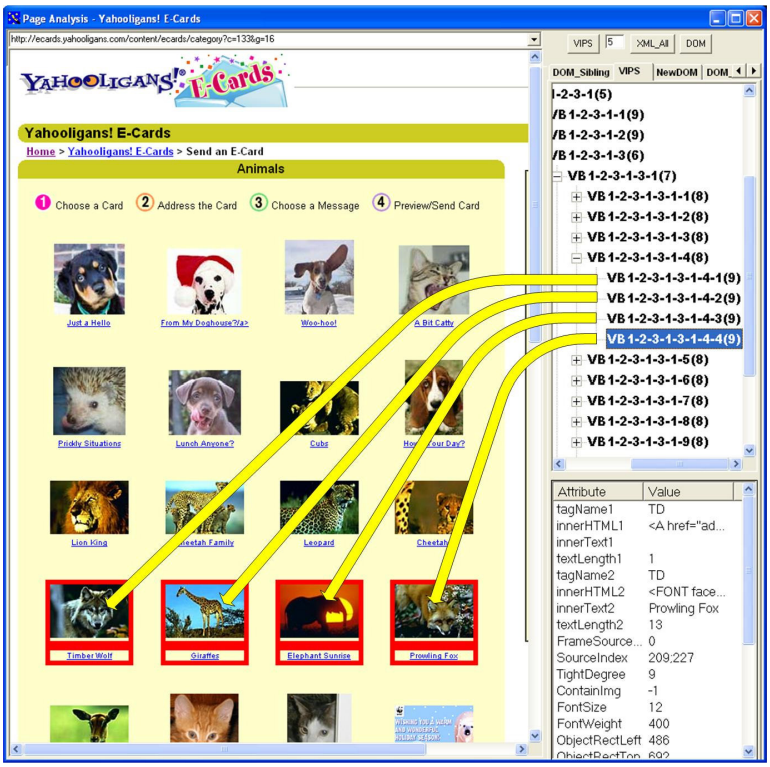


Figure 13. VIPS Partition result

In DOM tree structure, the images and the texts are belonging to different <TR> nodes. It is hard to decide the exact description text of the image. VIPS result clearly reveals the semantic relationship of the images and their surrounding texts. We can use these surrounding texts to represent the images and used this text representation in a web image search system.

From these examples, we can see that our VIPS algorithm can successfully identify the relationships among different blocks in the web page, while DOM structure fails. Moreover, VIPS algorithm assigns a DoC value to each node in vision-based content structure, which is critical in determining where to stop in different applications.

5.2 Performance of VIPS algorithm

To evaluate the performance of VIPS algorithm, we select 600 web pages from popular sites listed in 14 main categories of Yahoo! directory (http://www.yahoo.com). The VIPS algorithm is run on all the pages and the results are assessed by several individuals. Five volunteers are asked to judge the results. Table 3 shows the result.

Table 3. Evaluation of the VIPS algorithm

Human judgment	user1	user2	user3	user4	user5	All
Perfect	299	307	374	310	377	1667
Satisfactory	231	266	193	237	197	1124
Fair	64	23	29	48	20	184
Bad	6	4	4	5	6	25

As can be seen, 1667+1124=2791 (93%) pages have their semantic content structures correctly detected. For those "fair" pages, the hierarchies are incorrectly constructed because they contain insufficient visual cues to separate blocks while people get the structure by understanding the semantic meaning of the blocks. For the "bad" cases, one major reason is that the browser (i.e. Internet Explorer in our experiments) provides wrong position information so that our algorithm can not get the correct content structure. Another reason is that, for several pages the images (e.g., a very thin image representing a line) are used to divide different content blocks. Our algorithm currently can not handle this situation.

5.3 Experiments on web information retrieval

Query expansion is an efficient way to improve the performance of information retrieval.[8, 16] The quality of expansion terms is heavily affected by the top-ranked documents. Noises and multi-topics are the two major negative factors for expansion term selection in the web context. Since our VIPS algorithm can group semantically related content into a block, the term correlations within a segment will be much higher than those in other parts of a web page. With improved term correlations, high-quality expansion terms can be extracted from segments and used to improve information retrieval performance.

We choose Okapi [23] as the retrieval system and WT10g [4] in TREC-9 and TREC 2001 Web Tracks as the data set. WT10g contains 1.69 million pages and amounts to about 10G. The 50 queries from TREC 2001 Web Track are used as the query set and only the TOPIC field for retrieval, and use Okapi's BM2500 [24] as the weight function and set k_1 = 1.2, k_3 = 1000, b = 0.75, and $avdl$ = 61200. The baseline is 16.55% in our experiments.

An initial list of ranked web pages is obtained by using any traditional information retrieval methods. Then we apply different page segmentation algorithms (including our VIPS algorithm with PDoC(6) and a naïve DOM-based approach) to the top 80 pages

and get the set of candidate segments. The most relevant (e.g. top 20) segments from this candidate set are used to select expansion terms. These selected terms are used to construct a new expanded query to retrieve the final results.

We compared our method with the traditional pseudo-relevance feedback algorithm using whole document and a naïve segmentation method based on DOM tree, which are briefly described below:

- Our Vision-based approach (denoted as VIPS): The PDoC is set to 6. To reduce the effect of tiny blocks, blocks less than 10 words are removed. The top 80 pages returned by the initial retrieval phase are segmented to form the candidate segment set.
- Simple DOM-based approach (denoted as DOMPS): We iterate the DOM tree for some structural tags such as TITLE, P, TABLE, UL and H1~H6. If there are no more structural tags within the current structural tag, a block is constructed and identified by this tag. Free text between two tags is also treated as a special block. Similar to VIPS, tiny blocks less than 10 words are also removed, and the candidate segments are chosen from the top 80 pages returned by the initial retrieval phase.
- Traditional full document approach (denoted as FULLDOC): The traditional pseudo-relevance feedback based on the whole web page is implemented for a comparison purpose.

The experimental result is shown in Table 4 and Figure 14.

Table 4. Performance comparison of query expansion using different page segmentation methods

Number of Segments	Baseline (%)	FULLDOC (%)	DOMPS (%)	VIPS (%)
3	16.55	17.56 (+6.10)	17.94 (+8.40)	18.01 (+8.82)
5		17.46 (+5.50)	18.15 (+9.67)	19.39 (+17.16)
10		**19.10 (+15.41)**	18.05 (+9.06)	19.92 (+20.36)
20		17.89 (+8.10)	19.24 (+16.25)	**20.98 (+26.77)**
30		17.40	19.32	19.68

		(+5.14)	(+16.74)	(+18.91)
40		15.50 (-6.34)	19.57 (+18.25)	17.24 (+4.17)
50		13.82 (-16.50)	**19.67** **(+18.85)**	16.63 (+0.48)
60		14.40 (-12.99)	18.58 (+12.27)	16.37 (-1.09)

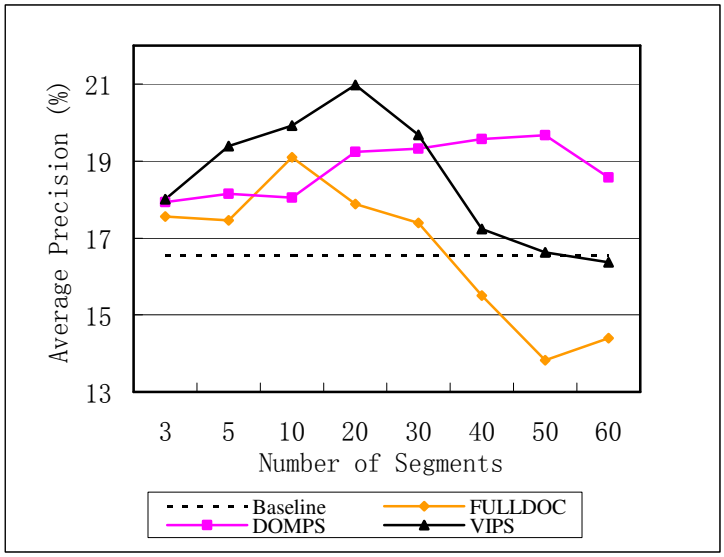


Figure 14. Performance comparison of pseudo-relevance feedback based on three different ways of selecting query expansion terms.

As can be seen, the average retrieval precision can be improved after partitioning pages into blocks, no matter which segmentation algorithm is used. In the case of FULLDOC, the maximal average precision is 19.10% when the top 10 documents are used to expand the query. DOMPS obtains 19.67% when the top 50 blocks are used, a little better than FULLDOC. VIPS gets the best result 20.98% when the top 20 blocks are used and achieves 26.77% improvement.

Document based query expansion FULLDOC uses all the terms within the top documents for expansion. Since the baseline is very low, many of top ranked documents are actually irrelevant and there are many terms coming from irrelevant topics. These cause the retrieval performance relatively low although better than the baseline. For the same reason, the average precision drops quickly

DOM based approach DOMPS does not obtain a significant improvement compared with FULLDOC, partly because the segmentation is too detailed. The segments are usually too short to cover complete information about a single semantic. This is a major limitation of segmentation based on naïve DOM tree which we addressed before. In many cases, good expansion terms are within the previous or proceeding blocks, but are missed because those blocks are not ranked high enough to be selected in pseudo-relevance feedback.

Compared with DOMPS, our VIPS algorithm considers more visual information and is more likely to obtain a semantic partition of a web page. Therefore, better expansion terms can be extracted and better performance can be achieved. About 27% performance improvement on the Web Track dataset was achieved.

The experiments clearly show that vision-based web page content structure is very helpful to detect and filter out noisy and irrelevant information. Thus better expansion terms can be selected to improve retrieval performance.

6 Conclusion

In this paper a new approach for extracting web content structure based on visual representation was proposed. The produced web content structure is very helpful for applications such as web adaptation, information retrieval and information extraction. By identifying the logic relationship of web content based on visual layout information, web content structure can effectively represent the semantic structure of the web page. An automatic top-down, tag-tree independent and scalable algorithm to detect web content structure was presented. It simulates how a user understands the layout structure of a web page based on its visual representation. Compared with traditional DOM based segmentation method, our scheme utilizes useful visual cues to obtain a better partition of a page at the semantic level. It is also independent of physical realization and works well even when the physical structure is far different from visual presentation. The algorithm is evaluated manually on a large data set, and also used for selecting good expansion terms in a pseudo-relevance feedback process in web information retrieval, both of which achieve very satisfactory performance.

References

[1]. Adelberg, B., NoDoSE: A tool for semiautomatically extracting structured and semistructured data from text documents, In Proceedings of ACM SIGMOD Conference on Management of Data, 1998, pp. 283-294.

[2]. Ashish, N. and Knoblock, C. A., Semi-Automatic Wrapper Generation for Internet Information Sources, In Proceedings of the Conference on Cooperative Information Systems, 1997, pp. 160-169.

[3]. Ashish, N. and Knoblock, C. A., Wrapper Generation for Semi-structured Internet Sources, SIGMOD Record, Vol. 26, No. 4, 1997, pp. 8-15.

[4]. Bailey, P., Craswell, N., and Hawking, D., Engineering a multi-purpose test collection for Web retrieval experiments, Information Processing and Management, 2001.

[5]. Bar-Yossef, Z. and Rajagopalan, S., Template Detection via Data Mining and its Applications, In Proceedings of the 11th International World Wide Web Conference (WWW2002), 2002.

[6]. Bernard, M. L.. Criteria for optimal web design (designing for usability). 2002.

[7]. Bharat, K. and Henzinger, M. R., Improved algorithms for topic distillation in a hyperlinked environment, In Proceedings of the 21st ACM International Conference on Research and Development in Information Retrieval (SIGIR98), 1998, pp. 104-111.

[8]. Buckley, C., Salton, G., and Allan, J., Automatic Retrieval with Locality Information Using Smart, In The First Text REtrieval Conference (TREC-1), National Institute of Standards and Technology, Gaithersburg, MD, 1992, pp. 59-72.

[9]. Buttler, D., Liu, L., and Pu, C., A Fully Automated Object Extraction System for the World Wide Web, In International Conference on Distributed Computing Systems, 2001.

[10]. Buyukkokten, O., Garcia-Molina, H., and Paepche, A., Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones, In Proceedings of the Conference on Human Factors in Computing Systems, CHI'01, 2001.

[11]. Chakrabarti, S., Integrating the Document Object Model with hyperlinks for enhanced topic distillation and information extraction, In the 10th International World Wide Web Conference, 2001.

[12]. Chakrabarti, S., Joshi, M., and Tawde, V., Enhanced topic distillation using text, markup tags, and hyperlinks, In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval , ACM Press, 2001, pp. 208-216.

[13]. Chakrabarti, S., Punera, K., and Subramanyam, M., Accelerated focused crawling through online relevance feedback, In Proceedings of the eleventh international conference on World Wide Web (WWW2002), 2002, pp. 148-159.

[14]. Chen, J., Zhou, B., Shi, J., Zhang, H.-J., and Wu, Q., Function-Based Object Model Towards Website Adaptation, In Proceedings of the 10th International World Wide Web Conference, 2001.

[15]. Diao, Y., Lu, H., Chen, S., and Tian, Z., TowardLearningBased Web Query Processing, In Proceedings of International Conference on Very Large Databases, 2000, pp. 317-328.

[16]. Efthimiadis, N. E., Query Expansion, In Annual Review of Information Systems and Technology, Vol. 31, 1996, pp. 121-187.

[17]. Embley, D. W., Jiang, Y., and Ng, Y.-K., Record-boundary discovery in Web documents, In Proceedings of the 1999 ACM SIGMOD international conference on Management of data, Philadelphia PA, 1999, pp. 467-478.

[18]. Hammer, J., Garcia-Molina, H., Cho, J., Aranha, R., and Crespo, A., Extracting Semistructured Information from the Web, In Proceedings of the Workshop on Management fo Semistructured Data, 1997, pp. 18-25.

[19]. Kaasinen, E., Aaltonen, M., Kolari, J., Melakoski, S., and Laakko, T., Two Approaches to Bringing Internet Services to WAP Devices, In Proceedings of 9th International World-Wide Web Conference, 2000, pp. 231-246.

[20]. Kleinberg, J., Authoritative sources in a hyperlinked environment, In Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, 1998, pp. 668-677.

[21]. Lin, S.-H. and Ho, J.-M., Discovering Informative Content Blocks from Web Documents, In Proceedings of ACM SIGKDD'02, 2002.

[22]. Rahman, A., Alam, H., and Hartono, R., Content Extraction from HTML Documents, In Proceedings of the First International Workshop on Web Document Analysis (WDA2001), 2001.

[23]. Robertson, S. E., Overview of the okapi projects, Journal of Documentation, Vol. 53, No. 1, 1997, pp. 3-7.

[24]. Robertson, S. E. and Walker, S., Okapi/Keenbow at TREC-8, In The Eighth Text REtrieval Conference (TREC 8), 1999, pp. 151-162.

[25]. Tang, Y. Y., Cheriet, M., Liu, J., Said, J. N., and Suen, C. Y., Document Analysis and Recognition by Computers, Handbook of Pattern Recognition and Computer Vision, edited by C. H. Chen, L. F. Pau, and P. S. P. Wang World Scientific Publishing Company, 1999.

[26]. Wong, W. and Fu, A. W., Finding Structure and Characteristics of Web Documents for Classification, In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), Dallas, TX., USA, 2000.