Block Extraction and Page Segmentation for Block-Level Web Search Engine

https://doi.org/10.15017/1398392
Block Extraction and Page Segmentation for Block-Level Web Search Engine

Jun Zeng
August 2013
Abstract

Along with the rapid development of web, the quantity of information available on the web today is more than at any point in history. Due to the popularity of web search engines, finding relevant web pages only takes less than one second. This is because before the query terms are submitted, a web search engine has built an index for each word to indicate a list of web pages where the word appears. However, even if web search engines can provide relevant web pages in such a short time, people still need to spend a lot of time reading the pages to find the relevant parts of a web page. An ideal goal of web search engine would be a “block-level” search engine where each web page is segmented into non-overlapped regions of an appropriate size. There have been many researches on extracting relevant parts from web pages for users’ query. However, these approaches are not concerning with covering all the contents of a web page and making an index for the sub-pages of a web page. In other words, it is necessary to formalize the appropriate parts of a web page for a query and to segment a web page into non-overlapped parts in order to realize a “block-level” web search engine. This thesis considers rectangular regions as the index targets and solves these two problems.

The first problem is known as block extraction from a web page with respect to a query. The required blocks may vary according to query. Thus, it is necessary to determine what kind of block is more likely to be a required block with respect to the query. Firstly, we consider the leaf blocks that satisfy the query. Then we analyze the blocks in the path from the leaf blocks to the root of HTML-tree. We analyzed the features of blocks, such as: the text quantity, DOM (Document Object Model) tree depth, number of child blocks, etc. We manually labeled the required blocks from a set of web pages and utilized SVM (Support Vector Machine) to train these features of the labeled blocks in order to determine which feature is most effective in detecting the required blocks. Based on the analysis results, we defined the score of a block as a combination of word weight and block depth. The score is used as a ranking measure of blocks for a query. The experiment results indicated that the proposed method is effective to extract the required block and useful to reduce users’ search time.
The second problem is known as page segmentation. The purpose of page segmentation is to divide a web page into independent segments. We use the visual features of blocks to segment web pages. First, we consider the record blocks where similar kinds of data are displayed adjacent to each other. The automatically generated outputs from databases are typical examples of these record blocks. We give a formulation of the similarity of blocks and introduce the notion of “layout tree”. For two given blocks, they are first transformed into two layout trees. Then the Tree Edit Distance algorithm is used to calculate the distance of the two layout trees. If the distance is less than the threshold, then the two blocks have similar layout. By using the layout tree, we can recognize the data record blocks in a given web page. We used this similarity to cluster blocks of a web page. We introduce two other measurements of seem degree and content similarity of two blocks. The seam degree describes how neatly the blocks are arranged. The content similarity describes the similarity of contents. The method first recognizes and marks the data record blocks in advance. According to the seam degree and content similarity, it can be determined whether a block should be divided or not. The experiment results show that the proposed method can divide a web page into appropriate suitable semantic segments.
Acknowledgement

I would like to express my sincere gratitude to my supervisor Prof. Sachio Hirokawa for his continuous support throughout the course of my Ph.D study and research. It would not have been possible to write this doctoral thesis without his helpful advice, warm encouragement, and insightful comments. His guidance helped me all the time during the research and writing of this thesis. I could not have imagined having a better supervisor and mentor for my Ph.D study.

I am deeply grateful to the committee members of my thesis: Prof. Keijiro Araki and Prof. Tsunenori Mine, for their valuable advice, and challenging questions.

I would also like to express my appreciation to Prof. Eisuke Ito, Prof. Chengjiu Yin, Prof. Tetsuya Nakatoh, and Prof. Takahiko Suzuki for their help and comments. I would like to thank Prof. Kensuke Baba for giving me the chance to be a teaching assistant for his course.

I also thank all the members in Hirokawa laboratory, especially to Mr. Brendan Flanagan. His careful proofreading helped me to publish my papers in international conferences and journals.

Last, but not the least, I would like to thank my family: my parents Huaiyu Ma and Shuguang Zeng, for giving birth to me in the first place and supporting me spiritually throughout my life.
Contents

Chapter 1  Introduction .............................................................................................. 1
  1.1  Background ........................................................................................................ 1
  1.2  Thesis Contribution .......................................................................................... 5
    1.2.1  Block Extraction ....................................................................................... 5
    1.2.2  Page Segmentation .................................................................................... 6
  1.3  Thesis Outline .................................................................................................. 8

Chapter 2  Preliminaries ......................................................................................... 10
  2.1  Blocks of Web Pages ....................................................................................... 10
  2.2  Pre-processing of Web Pages ......................................................................... 12

Chapter 3  Feature Analysis of Blocks .................................................................... 16
  3.1  Introduction .................................................................................................... 16
  3.2  Analyzing Features of a Single Block ............................................................ 17
    3.2.1  Quantity of Text in a Block ..................................................................... 17
    3.2.2  Query Keyword Occurrence Frequency .............................................. 18
    3.2.3  DOM Tree Structure Features of a Block ............................................ 20
  3.3  Analyzing the Feature of a Block in a Parent-Children Group ....................... 20
  3.4  Experiment and Evaluation .......................................................................... 23
    3.4.1  Data Set and Experiment Preparation ................................................... 23
    3.4.2  Evaluation for the Features of a Single Block ....................................... 24
Chapter 4 Extraction of Required Blocks ................................................. 30

4.1 Introduction ......................................................................................... 30
4.2 Related Work ....................................................................................... 31
4.3 Creating an Index for Leaf Blocks .......................................................... 32
  4.3.1 Japanese Morphological Analysis Using ChaSen ............................... 32
  4.3.2 Index Creation Using GETA ............................................................. 33
4.4 Extracting the Required Blocks ............................................................. 34
  4.4.1 Word Weight Based on Co-occurrence Frequency and TF-IDF .............. 34
  4.4.2 Block Score Based on DOM Tree Depth and Word Weight ................... 36
  4.4.3 Extracting the Required Block from Web Pages ..................................... 37
4.5 Experiment and Evaluation ................................................................. 38
  4.5.1 Data Set ....................................................................................... 38
  4.5.2 Experiment for Extracting Required Blocks ........................................... 38
  4.5.3 Usability Study ............................................................................... 41
4.6 Conclusion .......................................................................................... 43

Chapter 5 Recognizing Data Record Blocks Using Layout Tree .......... 44

5.1 Introduction ......................................................................................... 44
5.2 Related Work ....................................................................................... 47
5.3 Visual Features of Data Record Blocks .................................................... 49
5.4 Layout Tree of Blocks ........................................................................... 50
  5.4.1 Layout of Blocks ............................................................................. 50
  5.4.2 The Layout Tree of a Block ............................................................. 55
  5.4.3 The Similarity between Layout Trees ............................................... 57
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>Recognizing Data Record Blocks</td>
<td>59</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Clustering of Similar Layout Blocks</td>
<td>59</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Refining the Clusters of Similar Layout Blocks</td>
<td>62</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Detection of Data Record Region</td>
<td>62</td>
</tr>
<tr>
<td>5.6</td>
<td>Experiment and Evaluation</td>
<td>63</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Data Set</td>
<td>63</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Determining the Optimal Similarity Threshold of Layout Tree</td>
<td>64</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Performance Evaluation in Different Web Sites</td>
<td>66</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>68</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Web Page Segmentation Using Visual Semantics</td>
<td>69</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>69</td>
</tr>
<tr>
<td>6.2</td>
<td>Related Work</td>
<td>71</td>
</tr>
<tr>
<td>6.3</td>
<td>Seam Degree of Blocks</td>
<td>73</td>
</tr>
<tr>
<td>6.3.1</td>
<td>The Seam Degree of Two Adjacent Blocks</td>
<td>73</td>
</tr>
<tr>
<td>6.3.2</td>
<td>The Average Seam Degree of Adjacent Child Blocks in a Block</td>
<td>74</td>
</tr>
<tr>
<td>6.4</td>
<td>Content Similarity of Blocks</td>
<td>75</td>
</tr>
<tr>
<td>6.4.1</td>
<td>The Content Vectors of a Block</td>
<td>75</td>
</tr>
<tr>
<td>6.4.2</td>
<td>The Content Similarity of Two Blocks</td>
<td>77</td>
</tr>
<tr>
<td>6.4.3</td>
<td>The Average Content Similarity of Adjacent Child Blocks in a Block</td>
<td>79</td>
</tr>
<tr>
<td>6.5</td>
<td>Segment Web Pages Using Visual Semantics</td>
<td>80</td>
</tr>
<tr>
<td>6.6</td>
<td>Experiment and Evaluation</td>
<td>81</td>
</tr>
<tr>
<td>6.6.1</td>
<td>Data Set</td>
<td>81</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Evaluation of Seam Degree and Content Similarity</td>
<td>82</td>
</tr>
<tr>
<td>6.6.3</td>
<td>Comparison with VIPS</td>
<td>83</td>
</tr>
<tr>
<td>6.7</td>
<td>Conclusion</td>
<td>85</td>
</tr>
</tbody>
</table>
Chapter 7  Conclusion and Future Work................................................................. 87

  7.1  Conclusion............................................................................................................ 87

  7.2  Future Work......................................................................................................... 89

Bibliography........................................................................................................... 90
List of Figures

1.1: An example of blocks and the corresponding HTML elements ................................... 2
1.2: Nested blocks and the corresponding HTML elements ........................................... 4
1.3: An example of web page segmentation ...................................................................... 7
2.1: The structure of blocks in a web page ...................................................................... 10
2.2: The absolute coordinate and size of a block .......................................................... 11
2.3: Representation of text in web pages, HTML, and DOM trees .................................. 13
2.4: Example of tag with nested content ........................................................................ 14
2.5: The block tree of the example in Figure 2.1 ............................................................ 14
3.1: Two blocks with different average byte size ........................................................... 18
3.2: Two blocks with different keyword density ............................................................... 19
3.3: Structure features of block $b$ in a DOM tree ....................................................... 20
3.4: Two patterns of parent-children group ................................................................... 21
3.5: An instance of pattern (a) in Figure 3.4 ................................................................. 21
3.6: An instance of pattern (b) in Figure 3.4 ................................................................. 22
3.7: Evaluation system for labeling required blocks ...................................................... 23
3.8: Results of SVM learning with Gap ........................................................................ 27
3.9: Results of SVM learning without Gap .................................................................... 28
3.10: ROC curve of two SVM learning experiments ..................................................... 29
4.1: A fragment of frequency file .................................................................................. 33
4.2: Process of extracting required blocks ..................................................................... 37
4.3: Original page and DOM tree of the experiment page ................................................ 39
4.4: The distribution of labeled block rank ..................................................................... 41
4.5: Interface of the evaluation system ........................................................................... 41
5.1: The data record blocks in an online shopping web page ........................................ 44
5.2: Two blocks with different visual features ............................................................... 45
5.3: A simple example of a layout tree .......................................................................... 47
5.4: Two data record blocks from an online shopping page .......................................... 49
5.5: An example of the layout of a block ...................................................................... 51
5.6: Depth-first traversal of Figure 5.5 (b) .................................................................... 51
5.7: The process of determining separators ................................................................... 53
5.8: Algorithm for determining separators ..................................................................... 54
5.9: The process for layout tree generation ..................................................................... 55
5.10: Clustering the similar layout blocks in the same depth of block tree ............... 59
5.11: Algorithm for clustering similar layout blocks ...................................................... 61
5.12: The diagram of average values and different thresholds ....................................... 66
6.1: Segments of a news web page ................................................................................. 70
6.2: Three arrangement types of two blocks ................................................................. 73
6.3: Two blocks with different seam degree ................................................................... 75
6.4: Two blocks with different quantity of text ............................................................. 76
6.5: Content vectors of three blocks .............................................................................. 77
6.6: Average content similarity of adjacent child blocks in a block ......................... 79
6.7: Algorithm for web page segmentation ................................................................... 81
6.8: The coordinate graph of page segment results ...................................................... 83
6.9: The graph of questionnaire results .......................................................................... 85
List of Tables

1.1: Index file for blocks in Figure 1.2 ................................................................. 4
3.1: Average values of SVM learning results ....................................................... 25
3.2: Features of learning examples ....................................................................... 26
3.3: Weight of features ......................................................................................... 27
3.4: The results of two SVM learning experiments ............................................... 28
4.1: Function words in Japanese ........................................................................ 35
4.2: Size and post number of top-10 largest pages .............................................. 38
4.3: Comparison of spent time between E-system and N-system ....................... 43
5.1: The 10 web sites of dataset ......................................................................... 64
5.2: The average values of different thresholds .................................................. 65
5.3: The average values of different sites ............................................................. 67
6.1: Steps for judging a block whether should be divided .................................. 80
6.2: Average number of segments of 121 threshold pairs .................................. 83
6.3: The five criteria of evaluation ....................................................................... 84
6.4: The results of questionnaire evaluation ....................................................... 85
Chapter 1

Introduction

1.1 Background

As the largest database, the web contains a large number of information that may be interesting for researchers and the general public. The quantity of information available on the web today is more than at any point in history, but with this wealth of information comes even greater challenges. The technologies regarding extraction and retrieval of required and valuable information from the web have gained more and more attention. Therefore, commercial web search engines, such as: Google, Yahoo!, Goo, etc., have grown rapidly in recent years. Due to the popularity of web search engines, finding information has become an easy exercise by just typing a few keywords into a search engine text-box. When a user enters a query into a search engine (typically by using keywords), the engine examines its index and provides a list of best-matching web pages. This process usually spends less than one second.

The reason why a web search engine can find the relevant web pages so quickly is that the web search engine has built an index for each word before people submit their queries. The purpose of storing an index is to optimize speed and performance in finding relevant pages for a search query. Without an index, the search engine would scan every page in the database, which would require considerable time and computing power. Different web search engines may utilize different approaches to build their indices, but the essence of these indices is the same. In an index, each word has a list of page where the word appears, and also the frequency of the word appears in each page is also be stored in the index. Therefore, an index can help a web search engine to locate pages containing the query terms quickly.

Even if web search engines can retrieve the relevant web pages in less than one second, it does not mean that a user can find the relevant information he/she requires within such a short
time. When a user opens a web page from the search results of a search engine, he/she still needs to spend significant amount of time to find the information they are looking for in a given web page. According to [1], people spend 75% of their time in post-search phase of looking through individual web pages. Especially, when the given web page contains a great deal of information, such as BBS page, or when a smart phone is used to browse the web pages, people may spend much more time.

In order to help users to find the relevant information quickly, many web search engines (such as: Google, Yahoo!, etc.) provide a short snippet of each page in the search results to indicate the sentences that contain the query terms. However, these snippets also have some limitations. First, these snippets usually contain only one or two sentences and do not have enough information. Second, web page designers may utilize some attractive color, font or layout in order to make the web pages easy to read. Unfortunately, like other snippet search approaches [2][3][4], these snippets do not possess the visual features of the given page, which may make people lose the fun of browsing a web page. Third, these snippets do not indicate the relevant part of the given page. Even, the sentences in a snippet may be far apart from each other in a web page. This is because these snippets ignore the structure of the original web pages. Therefore, these snippets not only cannot help people to find the relevant information in web pages, but also ignore the visual and structure features of web pages.

Each block is defined by an HTML element, and different blocks may have different contents and styles

Figure 1.1: An example of blocks and the corresponding HTML elements
Unlike the text documents, a web page is a semi-structured document which is described in HTML (Hyper Text Markup Language). Each HTML element (except some invisible elements) can define a rectangle region of a web page. In this thesis, these rectangle regions are called blocks (see Section 2.1 for the detailed definition). Figure 1.1 shows an example of web page blocks and the corresponding HTML elements. According to the type of the elements, the blocks can contain different contents. In Figure 1.1, the content of Example 1 is text, the content of Example 2 is a picture, and Example 3 contains a picture, a link and text. These blocks can also be defined in different styles and layout. It is because these various blocks, a web page becomes more attractive and easier to read than a text document. Therefore, a web page is not an indivisible unit and it can be regarded as a set of finite blocks [17][18][19][20][21][22].

However, existing web search engines process web pages as the smallest and undividable units of information, and that is why they can only locate the relevant pages containing user’s query. In other words, existing search engines are page-level web search engines. Many researchers consider the existing page-level web search engines to be coarse-grained [18][20][25]. In a word, even if a web page is relevant to user’s query, this does not mean that every block in the page is relevant to user’s query.

In order to find the meaningful blocks, many approaches have been proposed. Some approaches aim to recognize the informative blocks of a web page [10][11][12][13][14][15][16] and some approaches aim to detect and remove the noisy or meaningless information from a web page [5][6][7][8][9]. However, the extracted blocks might not match the information user demanded and removing noisy or meaningless information cannot guarantee to return required information. This issue gives rise to the question: Is there a solution which can help user to find the relevant blocks quickly.

In this thesis, a solution to this problem is to build a block-level web search engine that aims to locate the relevant blocks rather than the relevant web pages. Compared with a page-level web search engine, a block-level web search engine can provide finer-grained web search results, which can be useful to (i) improve the accuracy of search results; (ii) filter out the irrelevant contents from search results in the block-level; (iii) browse web pages using small screen devices; etc.

In order to realize a block-level search engine, a block-level index needs to be built. Similar to the page-level index, the block-level index is used to locate the blocks containing the query terms. In the page-level, any two web pages are complete independent and search engines can index each page without considering the case that two pages are nested. However, in the block-level, the situation is different. As introduced before, a web page block is defined
by an HTML element, which is an individual component of an HTML document. An HTML element may include other elements. As a result, a block may include other blocks. In this thesis, these blocks are called “nested blocks”. If an index is built for the nested blocks, the search results may also be nested. Let us see an example that is shown in Figure 1.2.

In Figure 1.2 shows three nested blocks and their corresponding HTML elements where `<div>` element includes the two `<p>` elements. Within the three blocks, block 1 is defined by the `<div>` element and block 2 and 3 are defined by two `<p>` elements respectively. There are a total of five words: `I`, `go`, `Fukuoka`, `is`, and `Japan`, excluding the stop words `to` and `in`. Table 1.1 shows the index file for the blocks in Figure 1.2. Suppose that the query term is “Fukuoka”. According to the index, block 1, 2, and 3 will be retrieved as the search results. Since block 1 contains block 2 and 3, these search results are so called nested search results. This is because `Fukuoka` appears in all the three blocks. Obviously, these search results not only cannot help user to find relevant information quickly, but also may make user confused. In this example, there are only three nested blocks. However, in a real web page, there may be tens of nested blocks. For a user, these nested search results are redundant and difficult to understand. Therefore, the cornerstone for building a block-level search engine is to build a proper block-level index which can avoid the nested search results.

![Figure 1.2: Nested blocks and the corresponding HTML elements](image)

### Table 1.1: Index file for blocks in Figure 1.2

<table>
<thead>
<tr>
<th>Word</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1, 2</td>
</tr>
<tr>
<td>go</td>
<td>1, 2</td>
</tr>
<tr>
<td>Fukuoka</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>is</td>
<td>1, 3</td>
</tr>
<tr>
<td>Japan</td>
<td>1, 3</td>
</tr>
</tbody>
</table>
1.2 Thesis Contribution

In order to build a proper block-level index, there are two solutions. First solution is to build an index for the blocks that do not include other blocks. In this thesis, these blocks are called leaf blocks (See Section 2.1 for detailed definition). For example in Figure 1.2, block 2 and 3 are leaf blocks. Since block 1 is not a leaf block, thus block 1 will not be indexed. However, indexing the leaf blocks may ignore the structure of web pages and make the search results too scattered. For example, if the query is Fukuoka, block 2 and 3 will be retrieved as search results. However, block 1 contains both block 2 and 3, thus block 1 may be a better choice than block 2 and 3. Therefore, it is necessary to extract the proper block after the relevant leaf blocks are determined by analyzing the structure of web pages. This process is called block extraction.

Second solution is to divide a web page into segments that are not nested and have independent semantics. This process is called page segmentation. Each segment can be regarded as an independent sub-page and they can be indexed directly. In this case, nested search results will not occur. For example in Figure 1.2, if block 1 is a segment, then only block 1 will be indexed. If the query term is Fukuoka, block 1 will be retrieved as a search result and block 2 and 3 will be ignored. Therefore, this thesis provides two key contributions: block extraction and page segmentation.

1.2.1 Block Extraction

For a given query, people require the blocks that need to satisfy two conditions: (i) it contains as many relevant contents as possible; (ii) it contains as few irrelevant contents as possible. Generally, the coarser the grain of a block is, the more likely the block may contain more irrelevant information. Conversely, the finer the grain of block is, the more likely the block may lose more relevant information. This is a dilemma, and therefore a required block is actually a tradeoff between the rate of relevant and irrelevant information. In order to determine such blocks, we need to analyze the Document Object Model (DOM) structure of web pages. The DOM is a language-independent convention for representing and interacting with objects in HTML documents. Each HTML element has a corresponding DOM object. As the increasing complexity and variety of web pages, heuristics-based and pattern detection methods are not suitable.

This thesis focuses on a formulated method to extract the required blocks. First, instead of analyzing the DOM tree patterns of the nested blocks, we analyze the basic features of blocks,
such as: the depth of a block in a DOM tree, the child number of a block, etc. We manually labeled the required blocks from a set of web pages and utilized SVM (Support Vector Machine) to train these features of the labeled blocks in order to determine which basic feature is most effective to detect the required blocks (Chapter 3). Second, based on the result of the analysis, we proposed a novel method to determine the required blocks from a web page by using the DOM tree depth and word weight (Chapter 4). This method builds index for the leaf blocks of web pages. After a query term is submitted, the scores of all blocks in a web page are calculated based on the DOM tree depth and the total word weight of a block. The word weight is associated with co-occurrence frequency between the word and query term. The higher the word weight is, the more relevant the word will be. After the score of every block is determined, all blocks in a web page are ranked from high to low score. The 1-ranked block will be extracted as the required block of a web page.

1.2.2 Page Segmentation

In order to transform the nested blocks into independent blocks, we need to divide web pages into segments which are not nested and have independent semantics. The process that breaks a web page into segments is known as web page segmentation. A segment is actually a block whose descendant blocks have coherent and complete semantics. Once a block is determined as a segment, its ascendant and descendant blocks will be omitted. Figure 1.3 shows an example of web page segmentation, where (a) is a web page before segmentation, (b) is the page after segmentation. As shown in (b), after segmentation, each segment is completely independent and the nested blocks are omitted. Each segment can be considered to be a sub-page which can be directly indexed like a smaller web page. In this way, the nested search results can be avoided.

The early techniques of web page segmentation are mainly based on machine learning algorithms [48][51][55][56] and rule-based heuristics [49][50][52][53][54][57][58][61][32]. Because of the small scale training data set, machine-learning-based methods can be only applied in some certain fields of web pages. The heuristics-based approaches involve simple rule-based heuristics either by interpreting the meaning of tag structures or visual analysis. While a heuristic approach might work well on small sets of pages, it isn’t suitable for large-scale sets of pages.
In this thesis, we segment web pages using the visual semantics. Web pages are typically designed for visual interaction. In order to support visual interaction, Web pages are designed to consist of multiple segments which have different functions and visual characteristics. For example, in a news site, a long text may be the main content; a link list may be the related news list; a big picture may be an advertisement, etc. Due to the different visual features, people can easily identify each of the segments without any descriptions. With the help of these visual features, humans can also subconsciously filter out the spam information from required information. We call these visual features visual semantics. However, these semantics are intuitive and human friendly. In other words, they are not machine friendly and therefore difficult to be understood by computers. This issue gives rise to the question: How can these visual semantics be formulated. We use three formulated measures to represent these visual
semantics: layout tree (Chapter 5) is used to recognize the data record blocks; Seam Degree is used to describe how neatly the blocks are arranged; Content Similarity is used to describe the content distance between the blocks. Based on these three measures, we can divide a web page into segments visually and semantically (Chapter 6).

1.3 Thesis Outline

The rest of this thesis is organized as follows:

In Chapter 2, the formulated definition of blocks is given, and the pre-processing of web pages is introduced.

Chapter 3 and Chapter 4 aim to extract the required block by indexing the leaf blocks of web pages.

In Chapter 3, we analyzed the features of blocks in order to determine which feature is most effective to detect the required blocks. The features not only include basic features, but also a Gap feature which describes the ratio of a parent block and its largest child block. The basic features include text quantity, query keyword frequency, and DOM tree structure features. We manually labeled the required blocks from a set of web pages and utilized SVM to train these features of the labeled blocks.

In Chapter 4, we proposed a novel method to extract the required block from a web page. This method both focuses on the DOM tree depth and the total word weight of blocks. The DOM tree depth of blocks can have an effect on the grain of the blocks, thus DOM tree depth can have an effect on the relevance of the blocks. The word weight is associated with co-occurrence frequency between the word and query term. The higher the word weight is, the more relevant the word will be. The method first builds an index for leaf blocks of each web page. When a query keyword is given, the method retrieves the relevant web pages and automatically calculates score of all blocks in these web pages using the DOM tree depth and total word weight of each block. In each page, the blocks are ranked from high to low score. The 1-ranked block will be extracted as the required block.

Chapter 5 and Chapter 6 aim to divide a web page into semantically coherent segments.

In Chapter 5, we proposed a novel method to recognize the data record blocks. These data record blocks are often automatically generated from databases, these data records have independent and complete semantics. For example, in an online shopping page, each data record block represents a product and therefore each of these blocks can be regarded as an
individual segment. Based on our observation, these data record blocks in a web page always have the similar layout. Our method focuses on the layout feature of these blocks. We use separators to divide a block into nonoverlapping parts. A separator is actually a horizontal or vertical line that can divide a block into two smaller parts. These separators can be considered as a root of a tree, and the two smaller parts can be considered as the left subtree and the right subtree. Therefore, the given block can be transformed into a tree. We call the tree a “layout tree”. For two given blocks, first they are transformed into two layout trees respectively. Then the Tree Edit Distance (TED) algorithm is used to calculate the similarity of the two layout trees. If the similarity is less than the threshold, then the two blocks are layout similar. By using the layout tree, we can recognize the data record blocks from a given web page.

In Chapter 6, in addition to the method that is proposed in Chapter 5, we proposed two other two visual semantic methods: (i) Seam Degree is used to describe how neatly the blocks are arranged; (ii) Content Similarity is used to describe the content distance between the blocks. Based on these methods, we proposed a formulated method for web page segmentation. Since the data records blocks are special blocks and Seam Degree and Content Similarity are not suitable for them. Therefore, we need to recognize and mark the data record blocks in advance by using the proposed method in Chapter 5. After that, this page segmentation method judge DOM nodes top-down. It begins from the root node of the DOM tree which is set to be the current node. The corresponding block of the current node will be judged according to a set of rules. If the current node should be divided, then its child blocks will be judged as well. If the current node should not be divided, then it will be pushed into an array of segments and its child blocks will not be judged anymore. If there are no blocks that need to be judged, the algorithm terminates.

The contents of Chapter 3, 4, 5, and 6 are published in [66][67], [64][65], [68][69], and [70] ([70] is in press) respectively.
Chapter 2

Preliminaries

2.1 Blocks of Web Pages

A web page is made up of a finite number of blocks. We consider a block as a visible rectangular region in a web page, as shown in Figure 2.1.

The definition of a block is as follows:

Definition 2-1: Block $b = (Obj, Rect)$, where $Obj$ is a visible DOM object, and $Rect$ represents the visible rectangular region where $b$ is displayed in the web page.

According to W3C\(^{(1)}\) standard, a web page can be transformed into a DOM tree, and each DOM object represents an HTML element in the web page. Therefore, it is obvious that each

\(^{(1)}\) http://www.w3.org/standards/techs/dom#w3c_all, 2012
block has a corresponding HTML element. The DOM objects can be divided into five types of objects: \textit{element}, \textit{attribute}, \textit{text}, \textit{comment} and \textit{document}. In the five types of objects, only \textit{element} objects and \textit{text} objects can be seen in a web page. In other words, the visible DOM objects can only be \textit{element} objects and \textit{text} objects. A DOM object not only contains the attributes of a HTML element, such as \textit{"tagName"}, \textit{"id"}, \textit{"value"} etc., but also contains the properties defined by the DOM, such as \textit{"childNodes"}, \textit{"nextSibling"}, etc. However these objects do not contain the absolute coordinate of the corresponding HTML element. They only contain a relative coordinate to the parent element. Moreover the objects often do not specify the value of \textit{width} and \textit{height} properties correctly. For example, an element is displayed correctly in a Web page, but the \textit{width} and \textit{height} properties of its corresponding DOM object may be zero. Therefore, we need to get the absolute coordinate and size by using the APIs of browsers instead of analyzing the property of these objects (the absolute coordinate and size will be used in Chapter 5 and Chapter 6). Thus we introduce \textit{Rect} to represent the visible rectangular region of the corresponding block as shown in Figure 2.2. Here \textit{left} is the horizontal coordinate, \textit{top} is the vertical coordinates of top-left point of block, \textit{width} is the width, and \textit{height} is the height of a block. It should be noted that not all \textit{element} objects have their corresponding blocks. The objects whose tags are \texttt{<head>}, \texttt{<script>}, \texttt{<meta>}, etc. and the objects whose \textit{"display"} property is \textit{"none"} or \textit{"hidden"} property is \textit{"true"} do not have a block.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2_2.png}
\caption{The absolute coordinate and size of a block}
\end{figure}

In Figure 2.1 there are some blocks that are located inside other blocks, we call this relationship \textit{"include"}. For example, \(b_1\) includes \(b_{1,1}\) and \(b_{1,2}\); \(b_2\) includes \(b_{2,1}\) and \(b_{2,2}\); \(b_3\) includes \(b_{3,1}\), \(b_{3,2}\) and \(b_{3,3}\). The detailed definition is as follows:

\textbf{Definition 2-2:} For two given blocks \(b_1 = (\text{Obj}_1, \text{Rect}_1)\) and \(b_2 = (\text{Obj}_2, \text{Rect}_2)\), if \text{Obj}_1 is a child node of \text{Obj}_2, then \(b_1\) is a child block of \(b_2\) and \(b_2\) is the parent block of \(b_1\). If \text{Obj}_1 is a descendant node of \text{Obj}_2, then \(b_1\) is a descendant block of \(b_2\) and \(b_2\) is an ascendant block of \(b_1\),
we define that $b_1$ includes $b_2$, denoted $b_1 \subset b_2$.

Definition 2-3: If a block $b = (\text{Obj}, \text{Rect})$ does not have any child blocks, then $b$ is a leaf block.

Leaf block is a very important type of block, since leaf blocks contains the contents and any two leaf blocks are not nested or overlapping. In the following chapters, we will mention that how to make use of leaf blocks.

Definition 2-4: For two given blocks $b_1$ and $b_2$, if $b_1 \subset b_2$ or $b_2 \subset b_1$, then $b_1$ and $b_2$ are two nested blocks.

Definition 2-5: Given a set of blocks $S = \{b_1, b_2, \ldots, b_n\}$, $\forall b_i \in S$ and $b_j \in S$, if $b_i$ and $b_j$ are two nested blocks, then $S$ is a nested block set.

2.2 Pre-processing of Web Pages

According to Definition 2-1, each block has a corresponding DOM object. Thus, we first obtain the DOM tree of the web page.

In a DOM tree, each node is a DOM object. The DOM objects can be divided into five types of objects: element, attribute, text, comment and document. We further classify the element objects into two categories: visible element objects and invisible element objects. The visible element objects whose width and height properties are not zero and the display property is not none can be seen through the browser. The invisible element objects contains objects whose tags are <head>, <script>, <meta>, etc, which do not have visual attributes. Moreover, we classify the visible element objects into two categories: inline objects and line-break objects. Inline objects affect the appearance of text and can be applied to a string of characters without a line-break, including objects whose tags are <b>, <big>, <font>, etc. The other visible element objects are line-break nodes. Obviously, only the visible element objects and text objects can be displayed in web pages. Thus we need to prune the DOM tree. It should be noted that we just mark the pruned DOM nodes and do not actually delete the pruned nodes from DOM tree. When we analyze the DOM tree, the marked nodes will be omitted. The pruning rules are as follows:

Rule 2-1: The attribute nodes, comment nodes, and document nodes should be cut.

Rule 2-2: The invisible element nodes whose tags are <head>, <script>, <meta>, etc should be cut.

Rule 2-3: The visible element nodes whose width and height properties are zero and
*display* property is *none* should be cut.

Rule 2-4: If a node contains only one node whose node name is `<#text>`, then the `<#text>` node should be cut.

Rule 2-5: If a node only contains `<#text>` nodes and inline nodes, and each inline node has only one `<#text>` node, then all the `<#text>` nodes and inline nodes should be cut.

As mentioned before, Rule 2-1, 2-2, and 2-3 aim to prune the nodes that cannot be displayed in web pages. Here Rule 2-4 and 2-5 will be introduced in detail. In my research, I assume that only the leaf nodes of a DOM tree may have textual content assigned to them. This assumption is important in order to avoid the generation of nested segments during the future phases of our algorithm. Thus, whenever we find internal nodes with textual content, we do not represent their children in our DOM tree representation, and thus it becomes a leaf node. The textual content associated to the removed tags is then associated to this new leaf node. Figure 2.3 shows the representation of text in web pages, HTML, and DOM trees. The text that appears in a web page is within a tag `<p>` in HTML. However, in the DOM tree, the `<p>` node contains an additional child node whose node name is `<#text>`. The `<#text>` node does not have *width* and *height* properties, and its parent node `<p>` also contains the text information of the `<#text>` node. Even if the `<#text>` nodes are cut, its text information will not be lost. Rule 2-4 is used to cut such `<#text>` nodes.

Another instance let us consider the tag `<p>` in DOM tree of Figure 2.4. Notice that such a tag contains a text directly attached to it (*Objects of the*...). Besides, inside this text there is a tag `<b>`, which also has a textual content (anchor text). In this case, we say that the content of
<b> is nested to the content of <p>. As nested contents can generate nested segments in the future phases of our algorithm, the tag <b> is removed from our DOM tree representation, and its textual content is associated to the content of <p>. Rule 2-5 aims to prune the inline nodes with nested <#text> nodes.

(a) Nested text in web pages  (b) Nested text in HTML  (c) Nested text in DOM tree

Figure 2.4: Example of tag with nested content

Figure 2.5: The block tree of the example in Figure 2.1

After the DOM has been pruned, only the visible nodes remain in the DOM tree. It should be noted that both element nodes and text nodes have corresponding DOM objects of the DOM tree. According to Definition 2-1, each block has a DOM object and a rectangular region. Thus, we need to get the corresponding rectangular region of each element object. The element object does not contain the absolute coordinate of the corresponding HTML element, and it only contains a relative coordinate to the parent HTML element. Fortunately, some
browsers provide APIs to get absolute coordinate easily. As for the text nodes, the width and height can be indirectly calculated by analyzing the width and height of parent node and sibling nodes. After the rectangular regions are determined, the corresponding blocks of DOM objects are also determined, therefore the pruned DOM tree can be considered to be a “Block Tree”. Figure 2.5 shows the block tree of the example in Figure 2.1.
Chapter 3

Feature Analysis of Blocks

3.1 Introduction

In Chapter 1, we introduced that a web page is not an atomic unit of information. Existing web search engines process web pages as the smallest and undividable units and cannot provide fine-grained search results. In order to overcome the limitations of existing search engines, a block-level web search engine is necessary. However, the blocks in a web are nested, which may cause nested search results. Therefore, the cornerstone of realizing a block-level web search engine is to build a proper index to eliminate the nested search results. One solution to this problem is to extract a required block by indexing the leaf blocks.

Although there are a lot of approaches have been proposed for content extraction from web pages [10][11][12][13][14][15][16]. These approaches mainly addresses the content extraction problem by analyzing the DOM tree pattern of web page or by interpreting the meaning and importance of tag structures in some way. There are two reasons why these approaches are not suitable for our problem. First, the number of possible DOM layout patterns is virtually infinite, which means that there is no common pattern that can be used for all kinds of web pages. Second, these approaches do not consider user’s query, the required block may change due to different queries.

In order to extract the required blocks, in this chapter, we will analyze the features that can distinguish the required blocks instead of analyzing DOM tree patterns. We roughly classify the features into two types: (i) the features of a single block; (ii) the feature of a block in a parent-children group. The features of a single block are the basic properties of a block, such as: the DOM tree depth, the number of child blocks, the number of leaf blocks, etc. The feature of a block in a parent-children group is to describe the relationship between the block
and the parent-children group where the block is in. We manually labeled the required blocks from web pages and use SVM to train and test these features in order to determine which feature is effective to distinguish the required blocks.

3.2 Analyzing Features of a Single Block

In this section, we aim to formulate the features of a single block. These features are classified into three categories: (i) text quantity; (ii) query keyword occurrence frequency; (iii) DOM tree structure features.

3.2.1 Quantity of Text in a Block

Although, a web page contains various different kinds of contents, such as text, image, video, applet, etc. However the quantity of text is still the most important feature that can reflect the quantity of information in a block. If a block contains just few words, it is more likely to be an irrelevant block. For example, the menus and navigation bars, even if they contain a query term, they cannot be considered as the required block. In order to determine whether the quantity of text is an effective feature or not, we introduce two parameters: byte size and average byte size, which are defined as follows:

Definition 3-1: Given a block $b$, the byte size of a block represents the total byte size of text in block $b$, denoted $\text{Byte}(b)$. If $b$ is a leaf block, $\text{Byte}(b)$ is total byte size of words in $b$. If $b$ is not a leaf block, $\text{Byte}(b)$ will be calculated as in equation (3-1):

$$\text{Byte}(b) = \sum_{\text{leaf}(b)} \text{Byte}(\text{leaf}(b))$$  \hspace{1cm} (3-1)

where $\text{leaf}(b)$ is a leaf block that is a descendant block of $b$.

Definition 3-2: Given a block $b$, average byte size of $b$ represents the average byte size of text in $b$ and its descendant blocks, denoted $\text{AvgByte}(b)$. If $b$ is a leaf block, $\text{AvgByte}(b)$ will be equal to $\text{Byte}(b)$. If $b$ is not a leaf block, $\text{AvgByte}(b)$ will be calculated as in equation (3-2):

$$\text{AvgByte}(b) = \frac{\text{Byte}(b)}{\text{Desc}(b)+1}$$  \hspace{1cm} (3-2)

where $\text{Desc}(b)$ is the total number of descendant blocks of $b$. Figure 3.1 shows two blocks
with different average byte size.

![Figure 3.1: Two blocks with different average byte size](image)

In Figure 3.1, $b_1$ and $b_2$ are two blocks, where $b_1$ includes $b_2$ and two image blocks, and $b_2$ includes three text blocks. According to Definition 3-1, $Byte(b_1)$ and $Byte(b_2)$ are the same. Since $b_1$ has six descendant blocks, and $b_2$ has only three descendant blocks, $AvgByte(b_2)$ is greater than $AvgByte(b_1)$. In this case, the average byte size of text can distinguish $b_1$ and $b_2$.

3.2.2 Query Keyword Occurrence Frequency

As mentioned before, a required block should contain the relevant contents to the user’s query. Empirically, a block containing the user’s query keywords will be more likely to contain relevant contents. It does not mean that the more keywords the block contains, the more likely the block is a required block. For example, the root node of DOM tree contains the most keywords, but it is not the required block. Here, we introduce three parameters: keyword frequency, average keyword frequency and keyword density. The definitions of the three parameters are as following.

**Definition 3-3:** Given a block $b$, keyword frequency represents the times that a query keyword $k$ appears in block $b$, denoted $KF(b, k)$. If $b$ is a leaf block, $KF(b, k)$ is the number of $k$ in block $b$. If $b$ is not a leaf block $KF(b, k)$ will be calculated as in equation (3-3):

$$KF(b, k) = \sum_{i} KF\left(l_{e}(b)k\right)$$  \hspace{1cm} (3-3)

where $leaf_{e}(b)$ is a leaf block that is a descendant block of $b$. 
Definition 3-4: Given a block $b$, average keyword frequency of $b$ represents the average times that a query keyword $k$ appears in $b$ and its descendant blocks, denoted $AvgKF(b,k)$. If $b$ is a leaf block, $AvgKF(b,k)$ will be equal to $KF(b,k)$. If $b$ is not a leaf block, $AvgKF(b,k)$ will be calculated as in equation (3-4):

$$AvgKF(b,k) = \frac{\sumKF(hk)}{Desc(b)+1}$$ (3-4)

where $Desc(b)$ is the total number of descendant blocks of $b$.

Definition 3-5: Given a block $b$, keyword density represents the ratio of total byte size of keyword $k$ to $Byte(b)$, denoted $DKF(b,k)$. It is calculated as in equation (3-5):

$$DKF(b,k) = \frac{Byte(b,k)}{Byte(b)}$$ (3-5)

where $Byte(b,k)$ represents the total byte size of keyword $k$ in block $b$.

Figure 3.2: Two blocks with different keyword density

Figure 3.2 shows two blocks with different keyword density. In Figure 3.2, $b_1$ and $b_2$ are two blocks, where $b_1$ includes $b_2$ and two text blocks. According to Definition 3-3, $KF(b_1,k)$ and $KF(b_2,k)$ are the same. Since $Byte(b_1)$ is greater than $Byte(b_2)$, $KF(b_2,k)$ is smaller than $KF(b_2,k)$. In this case, the keyword density can distinguish $b_1$ and $b_2$. 
3.2.3 DOM Tree Structure Features of a Block

Another important feature is the DOM tree structure feature. Given a block $b$, we introduce four parameters to formulate the structure features of a block in a DOM tree. They are: child block number, leaf block number, root depth, and leaf depth. They are defined as following.

Definition 3-6: Given a block $b$, child block number of $b$ represents the number of child blocks of $b$, denoted $\text{Child}(b)$. If $b$ is a leaf block, $\text{Child}(b)$ will be zero.

Definition 3-7: Given a block $b$, leaf block number of $b$ represents the number of leaf blocks of $b$, denoted $\text{Leaf}(b)$.

Definition 3-8: Given a block $b$, root depth of $b$ represents the depth from the root block to $b$, denoted $\text{RDepth}(b)$.

Definition 3-9: Given a block $b$, leaf depth of $b$ represents the depth from $b$ to one of its deepest leaf block, denoted $\text{LDepth}(b)$.

Figure 3.3 shows the structure features of block $b$ in a DOM tree.

![Figure 3.3: Structure features of block $b$ in a DOM tree](image)

3.3 Analyzing the Feature of a Block in a Parent-Children Group

For a single block, it may be easy to determine its features. However, it may be much more complex to determine the features of a group of nested blocks. In order to simplify the analysis, we will only consider the patterns of parent-children group which is the simplest unit of nested blocks.
Figure 3.4: Two patterns of parent-children group

(a) Google is launching balloons into near space to provide internet access to buildings below on the ground. About 30 of the superpressure balloons are being launched from...

(b) The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages.

Web browsers can also refer to Cascading Style Sheets (CSS) to define the appearance and layout of text and other material.

Figure 3.5: An instance of pattern (a) in Figure 3.4
Figure 3.6: An instance of pattern (b) in Figure 3.4

Figure 3.4 shows two patterns of parent-children group, where $P_1$ and $P_2$ are the parent blocks, and $M_1$ and $M_2$ are the child blocks that have the largest byte size of text (short for largest child block). In pattern (a), $M_i$ is much larger than any other child blocks. Figure 3.5 shows an instance of pattern (a). It is obviously that the article is the largest child block, and has a complete content. The other child blocks such as title, date and viewer are likely to be irrelevant contents. However, in pattern (b), the child blocks have almost the same byte size of text, such as shown in Figure 3.6. Although, Paragraph 3 is the largest child block, it is just one paragraph of the whole article. The other child blocks may also have relevant content as well.

According to the pattern (a) and (b), we found that the ratio between the text byte size of the largest child block and the text byte size of the parent block can distinguish the two patterns. Therefore, we introduce a feature called $Gap$, which is used to describe the relationship between the largest child block and the parent block in a parent-children group. Give block $b$, $Gap(b)$ is defined as in equation (3-6).

$$\begin{align*}
Gap(b) &= \begin{cases} 
\frac{\text{Byte}(b)}{\text{Max}\{\text{Byte}(\text{child}(b))\}} & \text{if } b \text{ is not a leaf block} \\
0 & \text{if } b \text{ is a leaf block}
\end{cases} 
\tag{3-6}
\end{align*}$$

where $\text{child}(b)$ is a child block of $b$, and $\text{Byte()}$ is the text byte size of a block (see Definition 3-1). When $b$ is a leaf block, the $Gap(b)$ will be zero.
3.4 Experiment and Evaluation

In order to evaluate the features of required blocks, we manually labeled the required blocks from web pages and analyzed the features of labeled blocks mentioned in the previous sections. We use SVM to train and test the features to determine whether these features are effective to distinguish the required blocks.

3.4.1 Data Set and Experiment Preparation

We collected 100 commonly encountered diseases as query keywords to conduct searches using Google. For each query keyword, we collected the top 100 pages from the search results of Google. After removing invalid pages, we collected 8197 web pages relating to the 100 commonly encountered diseases. The 8197 web pages are from 882 different web sites, which can guarantee a variety of web pages. This experiment is based on the assumption that the required blocks have a leaf block containing the query keyword. For determining whether a leaf block contains a keyword or not, we build an index for each leaf block of the 8197 web pages.

![Figure 3.7: Evaluation system for labeling required blocks](image)

By using the leaf block index, we submitted the 100 commonly encountered diseases as query keywords one by one to find the leaf blocks that contain these keywords. For a given keyword, we can determine the number of leaf blocks where the keyword appears. We chose the top 20 diseases, which appeared at most leaf blocks. In order to label the required blocks manually, we developed an evaluation system as shown in Figure 3.7.

Three participants joined in the evaluation experiment. Once a participant clicked a query keyword, the system chose 5 nested block sets whose leaf block contains the clicked keyword. For each nested block set, participants manually label the required block. If a block is labeled
by more the two participants, then the block is considered to be required block. As a result, we collected 17 required blocks. These required blocks are regarded as positive examples in SVM learning. Once a required block is determined, the evaluate system automatically chose the parent node and child nodes of the required block as the negative examples in SVM leaning. If the require block is a leaf block, then only the parent block is chosen as negative example. Finally, we collected 54 negative examples. We used the 71 examples (17 positive examples and 54 negative examples) to create the SVM learning data. In the experiments, SVM-light(1) is used as a SVM learning tool.

3.4.2 Evaluation for the Features of a Single Block

In section 3.2, we introduced the features of a single block. These features are classified into three categories: text quantity, query keyword occurrence frequency, DOM tree structure. The goal of this experiment is to determine which category of features is most effective to distinguish the required blocks. To verify the effectiveness of the three categories of features, we conducted the four SVM learning experiments.

1. We conducted the SVM learning using all of the features introduced in section 3.2 (short for All Para).

2. We removed Byte(b) and AvgByte(b), and conducted the SVM learning using the remaining features. (short for No Quan)

3. We removed KF(b, k), AvgKF(b, k) and DKF(b, k), and conducted the SVM learning using the remaining features. (short for No Query)

4. We removed Child(b), Leaf(b), RDepth(b) and LDepth(b), and conducted the SVM learning using the remaining parameters. (short for No DOM)

For each SVM learning, we used 3-fold cross validation. We partitioned the 71 examples into 3 sub-examples. Of the 3 sub-examples, a single sub-example is retained as the validation data for testing the model, and the remaining 2 subsamples were used as training data. Then we analyzed the precision, recall and F-measure of the four SVM learning experiments. Given a threshold \( \alpha \) of SVM learning result, \( \text{Precision}(\alpha) \), \( \text{Recall}(\alpha) \) and \( F(\alpha) \) are calculated as in equation (3-7), (3-8) and (3-9):

\[
\text{Precision}(\alpha) = \frac{\{ e_i \mid \text{svm}(e_i) \geq \alpha \& \ p(e_i) = 1 \} }{\{ e_i \mid \text{svm}(e_i) \geq \alpha \} } \\
\text{Recall}(\alpha) = \frac{\{ e_i \mid \text{svm}(e_i) \geq \alpha \& \ p(e_i) = 1 \} }{\{ e_i \mid p(e_i) = 1 \} }
\]

(1) http://svmlight.joachims.org
where \( \text{svm}(e_i) \) is the SVM learning result of example \( e_i \), \( p(e_i) \) represents whether the example \( e_i \) is a positive example or negative example. When \( e_i \) is a positive example, \( p(e_i) \) will be one, otherwise \( p(e_i) \) will be zero. \( \alpha \) is the threshold of the learning result. If \( \text{svm}(e_i) > \alpha \), example \( e_i \) will be classified as a positive example by SVM, otherwise, \( e_i \) will be classified as a negative example. We randomly chose each sub-example as training data once, and conducted SVM machine learning 100 times. We took the average \( \text{Precision}(\alpha) \), \( \text{Recall}(\alpha) \) and \( F(\alpha) \) of 100 machine leaning results. When \( F(\alpha) \) reached a maximum, we recorded the value of \( \text{Precision}(\alpha) \), \( \text{Recall}(\alpha) \) and \( F(\alpha) \) as shown in Table 3.1.

In Table 3.1, the results of “All Para” are considered to be the baseline. The numbers in the brackets are the percentage of difference between the value and that of baseline. The results show that \( \text{Precision}(\alpha) \), \( \text{Recall}(\alpha) \) and \( F(\alpha) \) had the maximum value when all of the parameters were used (All Para). The decrease of \( \text{Recall}(\alpha) \) was negligible in the other three SVM learning experiments. However, \( \text{Precision}(\alpha) \) and \( F(\alpha) \) decreased noticeable. Especially when the parameters of DOM tree structure were removed (No DOM), \( \text{Precision}(\alpha) \) decreased 34.5% and \( F(\alpha) \) decreased 23.2%. Therefore, the DOM tree structure features are most effective. It is also should to be noted that when \( KF(b, k) \), \( \text{AvgKF}(b, k) \) and \( \text{DKF}(b, k) \) were removed (No Query), the decreases of \( \text{Precision}(\alpha) \), \( \text{Recall}(\alpha) \) and \( F(\alpha) \) are not as great as that in “No Quan” and “No DOM” results. This is because the negative examples were the parent block or child block of positive examples. Therefore the negative examples and positive examples may have the similar \( KF(b, k) \), \( \text{AvgKF}(b, k) \) and \( \text{DKF}(b, k) \).

Therefore, the features of DOM tree structure are the most effective features to distinguish the required blocks. The other two categories of features play supplemental roles.
3.4.3 Evaluation of Gap Feature

In Section 3.3, we introduced a feature called Gap to describe the relationship between the largest child block and the parent block in a parent-children group. In this experiment, we will evaluate the effectiveness of Gap feature. We still use the 71 blocks (see Section 3.4.1) as experiment data. However, we changed the features of these blocks. The features that are used as learning data are shown as in Table 3.2. Since the DOM tree structure features are determined to be the most effective features in Section 3.4.2, we also use these features in this experiment. Since \( \text{Gap}(b) \) is associated with the text byte size of block \( b \) and its largest child block, we use \( \text{Byte}(b) \) and \( \text{Max}\{\text{Byte}(child_i(b))\} \) in this experiment.

To verify the effectiveness of Gap, we conducted two SVM learning experiments. The first experiment, we conducted the SVM learning using all of the features shown in Table 3.2. (short for With Gap) For the second experiment, we removed the Gap feature and conducted the SVM learning using the remaining features. (short for Without Gap) For each SVM learning experiment, we used the 71 examples as learning data, and then used the 71 examples as testing data. After SVM learning, the weight of each feature can be determined. This weight indicates the importance of features for distinguishing the required blocks. Table 3.3 shows the weight of each feature in the two SVM learning experiments. Both in Result 1 and Result 2, \( \text{Ldepth}(b) \) and \( \text{RDepth}(b) \) are two most important features of SVM classifier. However, Gap which is ranked at top 3 is also an important feature.
Table 3.3: Weight of features

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Result 1 (With Gap)</th>
<th>Result 2 (Without Gap)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gap(b)</td>
<td>3.0527</td>
<td></td>
</tr>
<tr>
<td>RDepth(b)</td>
<td>7.3102</td>
<td>10.4221</td>
</tr>
<tr>
<td>Ldepth(b)</td>
<td>47.5014</td>
<td>44.8766</td>
</tr>
<tr>
<td>Child(b)</td>
<td>2.2620</td>
<td>2.3932</td>
</tr>
<tr>
<td>Leaf(b)</td>
<td>1.5554</td>
<td>1.9609</td>
</tr>
<tr>
<td>Byte(b)</td>
<td>1.7082</td>
<td>1.7057</td>
</tr>
<tr>
<td>Max{Byte(child(b))}</td>
<td>2.1459</td>
<td>2.2000</td>
</tr>
</tbody>
</table>

Figure 3.8: Results of SVM learning with Gap
Table 3.4: The results of two SVM learning experiments

<table>
<thead>
<tr>
<th></th>
<th>Result 1 (With Gap)</th>
<th>Result 2 (Without Gap)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F(\alpha)$</td>
<td>0.652</td>
<td>0.611</td>
</tr>
<tr>
<td>Precision(\alpha)</td>
<td>0.517</td>
<td>0.579</td>
</tr>
<tr>
<td>Recall(\alpha)</td>
<td>0.882</td>
<td>0.647</td>
</tr>
</tbody>
</table>

We also analyzed the Precision(\alpha), Recall(\alpha) and $F(\alpha)$ of two SVM learning experiment. Figure 3.8 shows the curve of the three evaluation criteria with Gap, and Figure 3.9 shows the curve of the three evaluation criteria without Gap. Table 3.4 shows the comparison of two results when the $F(\alpha)$ is the maximum value. The $F(\alpha)$ of Result 1 is better than that of Result 2. Although the Precision(\alpha) of Result 1 is worse than that of Result 2, the Recall(\alpha) of Result 1 is much better than that of Result 2. Therefore, considering the precision-recall tradeoff, we regarded that the classifier with Gap is better than that without Gap.

Finally, we analyzed the Receiver Operating Characteristic (ROC) curve of two SVM learning experiment. ROC curve is a graphical plot of the true positive rate and false positive rate for a binary classifier system as its discrimination threshold is varied. Figure 3.10 shows the ROC curve of the SVM learning result. In the ROC space, the best possible prediction
method would yield a point in the coordinate (0, 1). Therefore, if a ROC curve is closer to (0, 1), the classifier will be better. It is obviously that the classifier with Gap is better than the classifier without Gap.

![ROC curve of two SVM learning experiments](image)

Figure 3.10: ROC curve of two SVM learning experiments

### 3.5 Conclusion

In this chapter, we analyzed the features of blocks in order to determine which feature is most effective to detect the required blocks. For a single block, we introduced three categories of features: text quantity, query keyword occurrence frequency, and DOM tree structure features. For a block in a parent-children group, we introduced the Gap feature to describe the relationship between the largest child block and the parent block. We manually labeled the required blocks from a set of web pages and utilized SVM to train these features. The results of the experiments shows that the features of DOM tree structure, especially the depth of a block (including the RDepth and LDepth) are most effective to distinguish the required blocks. The other features can also play supplement roles. Although, Gap feature is not the most effective feature, it can still improve the precision of a classifier for detecting the required blocks.
Chapter 4

Extraction of Required Blocks

4.1 Introduction

In Chapter 3, we analyzed the features of a required block. In this chapter, we propose a novel method to automatically extract the required blocks according to query terms. This method can accept user’s query and score each block of the nested blocks based on the DOM tree depth and the total word weight of blocks. According to Definition 2-1, every block has a corresponding DOM object. If a DOM node is in a higher level of the DOM tree it will have coarser grain. For example, the <html> object has the coarsest grain in a web page, since the corresponding block of <html> object is a whole web page. Conversely, a DOM tree in a lower level of the DOM tree will have a finer grain. For example, the corresponding DOM nodes of leaf blocks have the finest grain. As mentioned before, the grain can have an effect on the relevance of a block, thus we consider the DOM tree depth can also have an effect on the relevance of a block indirectly.

This method also considers the word weight as an important factor. The word weight is associated with co-occurrence frequency between the word and query term. The higher the word weight is, the more relevant the word will be. By calculating the total word weight of each block of the nested blocks, we can determine which block contains the most relevant contents. Combining the DOM tree depth, each block can be scored. The block that has the highest score is the required block.
4.2 Related Work

Web pages are described in HTML and they are typical semi-structured documents. With the increasing of HTML/XML documents, there is a large body of related work in content extraction and information retrieval from semi-structured documents.

The approaches that extract contents [10][11][12][13][14][15][16] from web pages can be regarded as query-independent. These approaches have addressed the content extraction problem by analyzing the DOM structure of HTML page, either by rendering and visual analysis or by interpreting and learning the meaning and importance of tag structures in some way, using heuristic as well as formalized, principled approaches. However, these approaches did not consider user’s query, the extracted content might not match the information user demanded.

Also, a lot of researchers focus on the approaches [18] [20][21][22][23][24][25] that can retrieve information from web pages according to user’s query. D. Cai et al. [18] consider that a web page consists of finite blocks. They divided web pages into segments and investigated how to take advantage of block-level evidence to improve retrieval performance in the web context. They compared four types of methods, including fixed-length page segmentation, DOM-based page segmentation, vision-based page segmentation, and a combined method which integrated both semantic and fixed-length properties. Experiments on block-level query expansion and retrieval were performed. They firmly believed that such a semantic partitioning of web pages effectively deals with the problem of multiple drifting topics and mixed lengths. E. Bruno et al. [20][21][22] paid more attentions on indexing and querying web page blocks. They split up pages into a set of visual blocks. They considered blocks of a page most similar to a query may be returned instead of the page as a whole. The indexing of a block took into account its content, its visual importance and, by permeability, the indexing of neighbors blocks. They modeled a page as a directed acyclic graph. Each node was associated with a block and labeled by the coefficient of importance of this block. Each edge was labeled by the coefficient of permeability of the target node content to the source node content. By using this model, they believed that the most relevant blocks can be retrieved. Z. Liu et al. [23][24] focus on the approach to search the snippets from XML document. Since both HTML and XML documents are semi-structured documents, the approaches for retrieving web page blocks and the approaches for retrieving XML snippets have many common points. The purpose of their research is to generate snippets for XML search results. They identified that a good XML result snippet should be a meaningful information unit of a
small size that effectively summarizes this query result and differentiates it from others, according to which users can quickly assess the relevance of the query result. These properties are also suitable for web page blocks. Web search engines can use these snippets for helping user to find relevant contents for a query. Hirokawa et al. [25] regarded a block as a web page component and proposed a component-based search engine in which the content components gain a high score will appear in a high-ranking the search results.

Our method can be classified as an information retrieval approach. Similar to [25], we score each block in a web page and find the most relevant block as search result.

4.3 Creating an Index for Leaf Blocks

In Chapter 2, we introduce the pre-process to transform a web into a pruned DOM tree. According to Definition 2-3, a leaf node of a DOM tree is corresponding to a leaf block in web page. In order to accept query keywords, we need to create index for each leaf block. There are two reasons why we create index for leaf blocks. First, the leaf blocks contain the real contents and the intermediate nodes are only used to describe the structure of web page. Second, the leaf blocks can be regarded as the atomic unit of a web page and any two leaf blocks are completely independent. In this section, we will introduce how to create index for leaf blocks.

4.3.1 Japanese Morphological Analysis Using ChaSen

For creating index, we first analyze the morpheme of each leaf block. Since the experiment dataset are mainly Japanese web pages, therefore we focus on the Japanese morphological analysis. Here, we use “ChaSen”\(^{(1)}\) to analyze morpheme of every leaf block. ChaSen is a morphological analysis system which basically has the following facilities and features.

(1) It segments Japanese text (sentences) string into morphemes and tags those morphemes with their parts of speech and pronunciations. It also tokenizes conjugative morphemes, i.e., it tags the conjugative morphemes with their base forms and conjugation forms.

(2) In its grammar and dictionaries, morphemes as well as connectivity of two morphemes / parts of speech are defined, where some costs are assigned to their definition.

(3) In its morphological analysis process, ChaSen sums up those costs of morphemes and their connectivities, then outputs results with the minimum cost.

\(^{(1)}\) http://chasen-legacy.sourceforge.jp/
By using ChaSen, the Japanese text in each leaf block can be segmented into morphemes, e.g., the sentence “私は学校に行く。” (I go to school) will be segmented into “私 (I), は, 学校 (school), に (to), 行く (go).”

4.3.2 Index Creation Using GETA

After the Japanese morphological analysis is done, we can calculate the occurrence frequency of the words in each leaf block. Given a set of web pages, we create a frequency file according to the occurrence frequency of each word. (2) http://geta.ex.nii.ac.jp/

Figure 4.1 shows a fragment of the frequency file. The lines beginning with “@” represent the identifier of each leaf block, and each identifier is unique. For example, “@ 1_12” means the 12th leaf block of the page whose id is “1”. Under the lines of identifiers, there are two special lines: (i) the lines that contain “h:” represents the id of the web page; (ii) the lines that contain “p:” represent the X-path of leaf block. Although these lines will not appear at search results, they will be used when we calculate the score of each block. The other lines represent the words and their occurrence frequency.

Given a set of web pages, according to the frequency file, we use GETA (Generic Engine for Transposable Association) to create an index for each leaf block in the given web page set.

```
... 
@ 1_12
1 h:1
1 p://html/body/div[2]/div/div/p[1]/
2 何
3 場所
1 ...
@ 4_2
1 h:4
1 p://html/body/div[2]/div/div/p[2]/
1 ホテル
2 温泉
...
```

*何: what; 場所: place; ホテル: hotel; 温泉: hot spring

Figure 4.1: A fragment of frequency file

(2) http://geta.ex.nii.ac.jp/
4.4 Extracting the Required Blocks

4.4.1 Word Weight Based on Co-occurrence Frequency and TF-IDF

Given a query term, suppose that a set of blocks is relevant to the query term. In order to determine a required block from the set of blocks, we need to calculate the relevance between query term and each block in the set of blocks. Generally, the more relevant terms a block contains, the more relevant the block will be. Therefore, we first need to calculate the relevance between query term and each term in the blocks. In this chapter, we call this relevance “word weight”. Give a query term \( q \) and a term \( w \), we consider there are two factors have effect on word weight of \( w \): (i) the co-occurrence frequency between \( q \) and \( w \) in the same leaf blocks; (ii) TF-IDF (Term Frequency–Inverse Document Frequency) of \( w \).

If two terms often occur in the same leaf blocks, we consider the two terms are relevant. Give a set of leaf blocks \( S \), if a query term \( q \) occurs in a set of leaf blocks \( M \), where \( M \subseteq S \), and a word \( w \) occurs in a set of leaf blocks \( N \), where \( N \subseteq S \). The co-occurrence frequency \( F(w,q) \) of \( q \) and \( w \) can be calculated as in equation (4-1):

\[
F(w,q) = \frac{|M \cap N|}{|M| + |N| - |M \cap N|}
\]  

(4-1)

However, some terms are so common that they may occur in many leaf blocks, e.g. “する (do)”, “行く (go)”, “できる (can)”, etc. It is obvious that these common terms have little relevance to the query term. Therefore, we use TF-IDF to guarantee that a common term cannot get a high word weight. TF-IDF is a numerical statistic which reflects how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. TF-IDF has two parts: TF (Term Frequency) and IDF (Inverse Document Frequency). TF is the measure of how often a term appears in a document. DF is the measure of how many documents of the collection does a term appear and IDF is the inverse of DF. There are various ways for calculating TF-IDF of a term. In this section, we use the simplest way to calculate TF-IDF. In this section, a document is actually a leaf block.

Given a term \( w \) and a leaf block \( b \), where \( b \) contains \( n \) terms, suppose \( w \) appears in \( b \) \( m \) times, then \( TF(w, b) \) can be calculated as in equation (4-2):

\[
TF(w, b) = \frac{m}{n}
\]  

(4-2)
Given a term \( w \) and a set of leaf blocks \( S \), where \( S \) contains \( P \) leaf blocks. Suppose \( w \) appears in \( Q \) leaf blocks, where \( Q < P \), then \( IDF(w, S) \) can be calculated as in equation (4-3):

\[
IDF(w, S) = \ln\left(\frac{P}{Q}\right)
\]  

(4-3)

Combined equation (4-2) and (4-3), \( TF-IDF(w, b, S) \) can be calculated as in equation (4-4):

\[
TF-IDF(w, b, S) = TF(w:b) \times IDF(w, S)
\]  

(4-4)

According to equation (4-1) and (4-4), the word weight \( WordWeight(w, q, b, S) \) of term \( w \) can be calculated as in equation (4-5):

\[
WordWeight(w,q,b,S) = F(w,q) \times IF - IDF(w,b,S)
\]  

(4-5)

Mostly, the leaf block set \( S \) is constant, therefore the word weight is often denoted \( Weight(w, q, b) \) in this chapter. After the word weight of each term is determined, we can calculate the total word weight of each block. However, in Japanese, there are many function words as shown in Table 4.1.

Table 4.1: Function words in Japanese

<table>
<thead>
<tr>
<th>Category</th>
<th>Description and Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pronoun</td>
<td>A pronoun is a pro-form that substitutes for a noun (or noun phrase) e.g. 私(I), 彼(he), これ(This), ここ(here)</td>
</tr>
<tr>
<td>Conjunction</td>
<td>A conjunction is a word that connects words, phrases, clauses, and sentences. e.g. しかし(but), また(or), そして(and)</td>
</tr>
<tr>
<td>Joshi</td>
<td>In Japanese grammar, particles are called joshi. e.g. に(in), で(at), は, を</td>
</tr>
<tr>
<td>Auxiliary verb</td>
<td>An auxiliary verb is a verb that gives further semantic or syntactic information about a main or full verb. e.g. できる(can), ない(not)</td>
</tr>
</tbody>
</table>

Function words express grammatical relationships with other words within a sentence. They have little lexical meaning or have ambiguous meaning. In order to reduce the effect of function words, we should get rid of them before calculating the total word weight of each
block. Fortunately, there are not many function words in Japanese, so we can manually create a function word file in advance. By using this file, we get rid of all the function words when calculating the total word weight of each block.

After all pre-processes are done, we calculate the total word weight of each block as follows:

Given a block $b$, if $b$ is a leaf block, then the total word weight $BlockWeight(b, q)$ of $b$ can be calculated as in equation (4-6):

$$BlockWeight(b, q) = \sum WordWeight(w_i, q, b)$$  \hspace{1cm} (4-6)

where $w_i$ is a term that appears in block $b$.

If $b$ is not a leaf block, then the total word weight $BlockWeight(b, q)$ of $b$ can be calculated as in equation (4-7):

$$BlockWeight(b, q) = \sum BlockWeight(leaf_i(b), q)$$  \hspace{1cm} (4-7)

where $leaf_i(b)$ represents a leaf block that is a descendant block of $b$.

Moreover, give a web page $P$, we can calculate the total word weight $PageWeight(P, q)$ of $P$ as in equation (4-8):

$$PageWeight(P, q) = \sum BlockWeight(leaf_i(P), q)$$  \hspace{1cm} (4-8)

Where $leaf_i(P)$ represents a leaf block in page $P$.

4.4.2 Block Score Based on DOM Tree Depth and Word Weight

Since both DOM tree depth and word weight can have effect on the relevance between a block and a query term, we introduce a score for each block based on DOM tree depth and word weight.

Given a block $b$ and a query $q$, the score of $b$ is calculated as in equation (4-9):

$$Score(b, q) = Depth(b) \times \ln(BlockWeight(b, q))$$  \hspace{1cm} (4-9)

Here $Depth(b)$ represents the DOM tree depth where block $b$ is in, and $BlockWeight(b, q)$
represents the total word weight of block $b$. Generally, the max depth of a DOM tree in a web page is between 2 and 25, but the total word weight of a block may be more than one thousand. The two variables are in different order of magnitude. Therefore, $\ln(\text{BlockWeight}(b, q))$ can make the two variables are in a same order of magnitude.

In equation (4-9), the score of a block consist of two parts: the DOM tree depth of block $b$, and the total word weight of the block. If $\ln(\text{BlockWeight}(b, q))$ is not considered, then blocks that are in the deeper level of DOM tree will have a higher score. In other words, the leaf blocks that are in the deepest level of DOM tree will always have the greatest score. Conversely, if $\text{Depth}(b)$ is not considered, then the root node of DOM tree, in other words the whole page will have the greatest score. Therefore, considering both $\text{Depth}(b)$ and $\ln(\text{BlockWeight}(b, q))$ can guarantee that (i) the leaf blocks do not always have greatest score; (ii) the whole page do not always have greatest score. Since both $\text{Depth}(b)$ and $\ln(\text{BlockWeight}(b, q))$ can have effect on the relevance between block $b$ and query $q$, therefore equation (4-9) can satisfy the condition that a required block is a tradeoff between the rate of relevant and irrelevant information. After the scores of blocks in a web page are determined, the blocks will be ranked from high to low score. The 1-ranked block will be extracted as the required block of the web page.

4.4.3 Extracting the Required Block from Web Pages

![Diagram of process of extracting required blocks](image)

Figure 4.2: Process of extracting required blocks

Figure 4.2 shows the process of extracting required block. First, a query term $q$ is submitted. Based on the leaf block index, the ID list of pages and Xpath list of leaf blocks that contain the query term are generated. According to the page IDs, the web pages can be obtained from page database. For each web page $P_i$, the total word weight $\text{PageWeight}(P_i, q)$ can be calculated as in equation (4-8). The web pages will be sorted from large to small
PageWeigh($P_i, q$). According the leaf block Xpaths, the leaf blocks that contain the query term can be obtained. For a given web page $P_i$, the score of each block will be calculated as in equation (4-9) and the block that has the highest score will be chosen as the required block of page $P_i$. As a result, a list of required blocks will be returned.

4.5 Experiment and Evaluation

4.5.1 Data Set

To evaluate the usability of the proposed method, two experiments are conducted. We collected 1,303 web page of tourism blogs from the site “Kyushu seifuku Blog(3)”, as experimental data. We pre-process the pages according Section 2.2 and create a leaf block index of these pages.

4.5.2 Experiment for Extracting Required Blocks

The goal of this experiment is to determine whether the proposed method can extract the required block from nested blocks in a web page. First, we analyzed the size and post number of the 1,303 blogs. The top-10 largest pages are shown in Table 4.2.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Size (Byte)</th>
<th>Posts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>776943</td>
<td>360</td>
</tr>
<tr>
<td>2</td>
<td>153215</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>149528</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>148991</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>148760</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>148539</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>148463</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>147963</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>146422</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>145404</td>
<td>5</td>
</tr>
</tbody>
</table>

(3) http://www.welcomekyushu.jp/
From Table 4.2, we noticed that the top-1 largest blog contains 360 posts and each post is a gourmet article about a Japanese noodle restaurant. In Figure 4.3, (a) shows the original
page and (b) shows its DOM tree. Each block of the gourmet post has a corresponding DOM node. We take this page as the experiment page, however, the co-occurrence between two terms and TF-IDF of a term are still based on the set of 1,303 pages.

Since each post is an article about a Japanese noodle restaurant, we chose each restaurant name as a query term and manually labeled the block of each post. Each restaurant name has a corresponding block of post. Obviously, for a restaurant name, the block of the corresponding post is the required block. In other words, if we submit a restaurant name as a query term, the block of the corresponding post should get the highest score according to equation (4-9).

We submitted the 360 restaurant names as query terms one by one and ranked all the blocks according to their score. We record the ranking of the corresponding labeled block. For a restaurant name, if the corresponding labeled block is a top ranked block, then the proposed method succeeds in extracting the required block. Otherwise, the proposed method fails to extract the required block.

Figure 4.4 shows the distribution of labeled block rank. The horizontal axis represents the rank of the labeled blocks and the vertical axis represents the number of labeled blocks in each rank. For the 360 query terms, 298 (82.7%) labeled blocks are at the top rank. We also analyzed the cases that the labeled blocks did not at the top rank.

(1) For a restaurant name, another labeled block instead of the corresponding labeled block is at the top rank. That is because the restaurant names, such as: “麺屋(noodle restaurant), 美味(delicious)” are common words. When these restaurant names are chosen as query terms, they may have many co-occurring words in other labeled blocks. In this case, the other labeled blocks may also receive a high score.

(2) For a restaurant name, the block that only contains the restaurant name is at the top rank. That is because these restaurant names are so special that there are few co-occurring words. When these restaurant names are chosen as query terms, the total word weight of the corresponding labeled block is close to 0. In this case, the block that only contains the restaurant name can receive the highest score.
4.5.3 Usability Study

In this experiment, we aim to evaluate the usability of the proposed method. Based on the index of 1,303 blogs, we developed an evaluation system. The system can accept user’s query term and return a list of web pages that contain this query term. For each page, the required block will be extracted automatically by using the proposed method. When a page is clicked, the system will display the selected page as in Figure 4.5. This interface consists of three parts: (i) original selected page; (ii) the DOM node of the required block; (iii) the text of the required block.

To evaluate the usability of the proposed method, we introduce a task-based usability
Additional to the evaluation system, we developed another system which only displays the original page and does not display the required block. We call this system “N-system” and call the evaluation system “E-system”. Ten participants joined in the experiment. We design the task of usability experiment as follows:

(1) First, each participant is assigned a query term. They submit the query term to E-system and N-system respectively. From the search results, each participant must browse 5 pages by using E-system and 5 pages by using N-system.

(2) For each page, participants are asked to find and save the sentences that contains the query term as quickly as possible. For a page $P_i$, $\text{Sentence}(P_i, q)$ represents the set of sentences that is found by participants, and $\text{Time}(P_i)$ represents the time from a participant begins to browse a detail page to he found out $\text{Sentence}(P_i)$. We recorded $\text{Sentence}(P_i, q)$ and $\text{Time}(P_i)$ of each page.

We got 100 experiment records: E-system 50 records and N-system 50 records. This experiment is based on the assumption that the sentences containing user’s query term are the required contents. Therefore, we asked the participants to find and save the sentences containing the query terms. By analyzing the time spent, we can evaluate whether E-system can improve user’s searching experience and efficiency.

Before analyzing the time spent, it is necessary to judge whether the records are valid or not. That’s because participants might be cursory when they were doing the experiment. They might not find out all the right sentences or just use Ctrl+A & Ctrl+C to copy all the sentences. Give a query term $q$ and a page $P_i$, $\text{TotalSentence}(P_i, q)$ represents the set of sentences that contain $q$. We introduce two criteria $\text{Precision}(P_i, q)$ and $\text{Recall}(P_i, q)$ to determine a the validity of records. They can be calculated as follows:

$$\text{Precision}(P_i, q) = \frac{|\text{Sentence}(P_i, q) \cap \text{TotalSentence}(P_i, q)|}{\text{Sentence}(P_i, q)}$$

$$\text{Recall}(P_i, q) = \frac{|\text{Sentence}(P_i, q) \cap \text{TotalSentence}(P_i, q)|}{\text{TotalSentence}(P_i, q)}$$

For a record, if $\text{Precision}(P_i, q) > 0.8$ and $\text{Recall}(P_i, q) > 0.8$, then the record is a valid records. As a result, we got 88 valid records: 46 records of E-system and 42 records of N-system. We compare the average $\text{Time}(P_i)$, min $\text{Time}(P_i)$ and max $\text{Time}(P_i)$ between E-system and N-system as shown in Table 4.3.
Table 4.3: Comparison of spent time between E-system and N-system

<table>
<thead>
<tr>
<th></th>
<th>E-system</th>
<th>N-system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average $Time(P_i)$</td>
<td>47.5 sec</td>
<td>98.2 sec</td>
</tr>
<tr>
<td>Min $Time(P_i)$</td>
<td>15 sec</td>
<td>35 sec</td>
</tr>
<tr>
<td>Max $Time(P_i)$</td>
<td>85 sec</td>
<td>103 sec</td>
</tr>
</tbody>
</table>

In the table, the average $Time(P_i)$, min $Time(P_i)$, and max $Time(P_i)$ of E-system is much less than that of N-system. This result indicates that E-system is more effectively to help participants to find the required sentences than N-system. Also, the result can indirectly determine that extracting the required blocks by using the proposed method is useful to improve user’s effectiveness when a user is seeking the information from web pages.

4.6 Conclusion

In this chapter, we proposed a method to automatically extract the required block from nested blocks in a web page. This method both focuses on the DOM tree depth and the total word weight of blocks. Based on the DOM tree depth and total word weight, each block will be scored. The block that has the highest score is extracted as a required block. The result of an evaluation experiment indicated that the proposed method is effective to extract the required block in most cases, and the result of a usability experiment indicated that extracting the required blocks by using the proposed method is useful to improve user’s effectiveness when a user is seeking the information from web pages.
Chapter 5

Recognizing Data Record Blocks Using Layout Tree

5.1 Introduction

As a solution to the problem of nested search results, page segmentation aims to divide a web page into segments that are not nested. In this way, each segment can be indexed directly without considering the case of nested blocks. As a result, the problem of nested search results can be avoided. Essentially, page segmentation must guarantee each segment has an independent and complete semantic. This chapter will introduce an important step for web page segmentation.

Figure 5.1: The data record blocks in an online shopping web page
In some web pages, data records are generated from databases. These data records have independent and complete semantics and these web pages are known as deep web pages. Figure 5.1 shows an online shopping web page where the dotted rectangles indicate the blocks of data records. Obviously, each block is a notebook computer product record, and thus we consider each of these blocks can be regarded as an individual segment. In this section, we aim to recognize these blocks of data records (short for data record blocks), which is an important step for web page segmentation.

Although many approaches have been proposed for recognizing and extracting data records from deep web pages in recent years, most of the approaches are mainly based on analyzing the HTML source code [29][33][40][46]. Due to increasing complexity of the HTML structure, it is more and more difficult to recognize the data records by analyzing the HTML source code only. Moreover HTML-based methods are language-dependent. Once the version of languages changes, these approaches are not able to adapt to the new version of the language. In order to overcome these limitations, some researchers proposed vision-based methods that rely on visual cues from browser renderings [35][38][45]. Most of the vision-based methods focus on the location, size or font features of elements in data records. These approaches can only be applied to certain Web page templates.

![Diagram of Block A and its HTML structure](image1)

(a) Block A and its HTML structure

![Diagram of Block B and its HTML structure](image2)

(b) Block B and its HTML structure

Figure 5.2: Two blocks with different visual features
Based on our observation, since the data records are generated from databases, the blocks of data records always appear in similar visual feature. For example, in Figure 5.1 each block contains a product picture, a product name and a price. Moreover, these contents are arranged in the same way. Therefore, we can recognize the data record blocks by calculating the similarity of these blocks need to calculate the similarity of these blocks. The HTML-based approaches calculate the similarity of HTML structures or tag paths to identify repetitive or similar patterns, while the vision-based approaches tend to calculate the similarity of image size and font size of the elements to recognize the similar patterns. Because of the limitations mentioned above, the HTML-dependent approaches may have the problem that is shown in Figure 5.2: block A and B look different, and they should be distinguished as different blocks. However A and B have the same HTML structure, same picture size and same font size. Thus, both the HTML-based and vision-based approaches may cluster them into the same group by mistake.

Besides HTML structure and visual cues, there is another important feature that is often ignored. That is the relative position of contents in a blocks. Some may consider relative position as a visual property. Strictly speaking, relative position is different from other visual properties. Visual properties, such as: position, size and font size only refer to just one single block, but relative position refers to at least two blocks. In other words, visual properties like position, size and font size are absolute and a single tuple, but relative position is relative and therefore a double tuple. The relative position of two blocks \( x_1 \) and \( x_2 \) can be represented as \( RP(x_1, x_2) = (x_1, r, x_2) \), where \( r \) is the relative position of \( x_1 \) and \( x_2 \). For example in Figure 5.2 (a), the picture is above the text. We can represent it as \( RP_1(picture, text) = (picture, is above, text) \). Here, \( x_1 \) is “picture”, \( r \) is “is above” and \( x_2 \) is “text”. Similarly, in Figure 5.2 (b), the picture is on the left side of the text. We can also represent it as \( RP_2(picture, text) = (picture, is on the left side, text) \). Here, \( x_1 \) is still “picture” and \( x_2 \) is still “text”, but \( r \) becomes “is on the left side”. Obviously, \( RP_1(picture, text) \) is different from \( RP_2(picture, text) \) and therefore we can conclude that: For two relative position \( RP_1(x_1, x_2) = (x_1, r_1, x_2) \) and \( RP_2(y_1, y_2) = (y_1, r_2, y_2) \), if and only if \( x_1 \) is similar to \( y_1 \), and \( x_2 \) is similar to \( y_2 \), and \( r_1 \) is the same as \( r_2 \), then \( RP_1 \) is similar to \( RP_2 \). Compared with other visual properties, relative position is more strict.

If there are three blocks \( x_1, x_2 \) and \( x_3 \), we can first take \( x_2 \) and \( x_3 \) as a whole \( x_{2,3} \) and get the relative position \( RP_1(x_1, x_{2,3}) \) of \( x_1 \) and \( x_{2,3} \). And then we can get the relative position \( RP_2(x_2, x_3) \) of \( x_2 \) and \( x_3 \). Thus, a tree is formed as shown in Figure 5.3. We call the permutation and combination of the relative positions “Layout”, and we call this tree a “Layout Tree”. A detailed definition is provided in the following section.
In this chapter, we propose a method to recognize the data record blocks from a deep web page using layout trees. We calculate the similarity of the layout trees of blocks and cluster the blocks with similar layout trees together. Finally, the data record blocks will be extracted from the clusters using other visual features.

### 5.2 Related Work

Although the goal of this work is to recognize data record blocks for web page segmentation, it can also be regarded as a method for web page data record extraction from a deep web page. Good surveys of works on web page data record extraction can be found in [27] and [30]. Early techniques used were manual approaches, such as in Minerva [44], TSIMMIS [37], and Web-OQL [34]. These works are difficult to maintain and have low efficiency. In order to tackle these problems, semi-automatic approaches [28][31][36][41][42] were proposed. These works require manual tasks to be carried out, for example, labeling sample pages. Thus, they are labor-intensive and time-consuming. To overcome the above limitations, fully automatic methods have been proposed. Methods such as MDR [29], TPC [33], STEM [46], RST [40], ViNTs [35], ViPER [38], and ViDE [45] were designed to tackle record-level extraction task from a single input page. (STEM [46] can also detect the data records from multiple pages.)

Our work falls into this category, precisely, as it is an automatic, record-level, and single-page based approach. We roughly divide these approaches into two groups: HTML-based approaches [29][33][40][46] and vision-based approaches [35][38][45].

HTML-based approaches often transformed HTML source code into a DOM tree, tag path or other structures. Most HTML-based approaches employ a similarity measurement to identify a region where a similar DOM subtree or tag path appears repeatedly. MDR [29]
identifies data regions by searching for multiple generalize-nodes using edit-distance similarity where generalize-nodes are a fix combination of multiple child nodes and their corresponding subtrees. A limitation of MDR is that it does not handle nested data objects. TPC [33] considers a web page as a string of HTML tags. It focused on comparing a pair of tag path occurrence patterns, and introduced a measure to calculate the similarity of these patterns. By clustering these similar tag path patterns, it can extract sets of tag paths that form the data records. But it is very difficult to differentiate between data records and non-data records when the tag paths are the same. STEM [46] labels each unique tag path as an integer, and a web page can be considered as a sequence of integers. It built a suffix tree based on this sequence and finds repeated tag path patterns. One suffix tree can be constructed for all of the sequences of multiple pages. Thus all the template patterns from multiple pages can be detected. RST [40] constructed a search structure named as Record Segmentation (RST). It can dynamically generate subtree groups from the RST structure during the search process. It used a token-based edit distance which takes each DOM node as a basic unit in the cost calculation. All of them mainly depend on analyzing the web page source code. As a result, they cannot avoid the following inherent limitations: first, they do not capture obvious visual cues and may make distinctions between regions that appear similar; second, many page designers do not obey W3C specification, and there can be a lot of mistakes in the DOM tree.

Vision-based approaches rely on visual cues from browser renderings. Most of the vision-based methods focus on the location, size or font features of elements. These approaches can make good use of the visual information that is defined by Javascript or CSS. ViNTs [35] use the visual content features on query result pages to capture content regularities denoted as Content Lines, and then utilize the HTML tag structures to combine them. ViNTs cannot separate horizontally arranged records, e.g., nested records in a table. ViPER [38] also incorporates visual information on a web page for data record extraction. It also depends on tag structure to detect record regions. In both of the approaches, tag structures are still the primary information utilized, while visual information plays a supplementary role. ViDE [45] can be considered as a pure visual feature based approach. It is effective in extracting records from pages with well organized visual features. It also extracts detailed data items from each data record.

Our method can also be considered to be a pure vision-based method. This method not only makes use of the visual features of blocks, but also considers the layout of these blocks.
5.3 Visual Features of Data Record Blocks

Based on our observation of the deep web pages, we discovered the following features of data record blocks.

(1) In a same web page, the leaf blocks of data record blocks are arranged in a similar layout. We call this feature Similar Layout (SL) feature.

This feature is the essence of our research. Although a data record include many smaller blocks, here only the leaf blocks are considered. This is because if the intermediate blocks are considered, the blocks may overlap each other.

![Figure 5.4: Two data record blocks from an online shopping page](image)

Figure 5.4 shows two data record blocks from an online shopping page. Although the contents of two records are not all the same, the main layout is similar. In both blocks, a picture is on the top of blocks; the product names are under the pictures; the prices are under the product names; evaluations are on the bottom. Block (a) contains some additional contents, but the layout of “picture”, “name”, “price” and “evaluation” is the same in both (a) and (b). Therefore, we consider a novel approach to recognize the data record blocks from a deep web page using the SL feature.

(2) In a deep Web page, the data record blocks have similar shapes and coordinates. We call this feature Similar Shape and Coordinate (SSC) feature for short.

This feature contains two aspects: one is similar shape, the other one is similar coordinate.
As shown in Figure 5.1, the data record blocks are arranged in two rows. In each row, the data records have the same height and vertical coordinates. Similarly, we can also consider that the data record blocks are arranged in four columns. In each column, the data records have the same width and same horizontal coordinates. Therefore, if two blocks have similar height or width, we consider that they have a similar shape. If two blocks have similar horizontal or vertical coordinates, we consider that they have the similar coordinates.

(3) In a deep Web page, the data record region always has the largest area and contains the most similar layout blocks. We call this feature Largest Area and Most Similar Blocks (LAMSB) feature for short.

This feature also contains two aspects: one is the largest area, the other one is the most similar layout blocks. First, the data records are the most significant parts of a deep web page. Web designers always attempt to make them as attractive as possible. Thus the data record region always occupies the largest area in deep web pages. Second, data records are the main contents of a deep web page. Thus the number of data records always is greater than that of other contents in a deep web page. As mentioned before, data records have similar layout. Thus the data record region contains the most similar layout blocks.

5.4 Layout Tree of Blocks

5.4.1 Layout of Blocks

In the previous section, the similar layout feature is mentioned as a key feature to recognize data record blocks from a deep web page. In this section, the description of layout and the creation of layout tree are introduced.

As mentioned in Section 5.1, the layout is used to describe the relative positions of contents in a block. We describe the layout as follows: for a block $B$, where $B$ is not a leaf block, the layout of $B$ is represented as $\text{Layout}(B) = (LB, S)$. $LB = (b_1, b_2, \ldots, b_n)$ is a finite sequence of leaf blocks that are included by $B$. All these leaf blocks are not overlapping. $S = (s_1, s_2, \ldots, s_{n-1})$ is a finite sequence of separators, including horizontal separators and vertical separators. A separator is actually a straight line that can separate a block into two smaller parts. The direction of a separator is a simple and effective way to describe the relative position. If a separator is horizontal, it means the relative position is up-down. If a separator is vertical, it means the relative position is left-right.
Figure 5.5 (a) shows the leaf blocks and separators of a block, and Figure 5.5 (b) shows the block tree structure of the block (a). In Figure 5.5 (a), the solid line rectangles represent the leaf blocks and dotted lines represent the separators. In Figure 5.5 (b), the dotted line boxes represent the intermediate nodes of the block tree. All the intermediate nodes \( \{n_1, \ldots, n_5\} \) are ignored, because if they are considered the blocks may overlap each other, which will make it difficult to determine the separators. Therefore only the leaf blocks are considered to describe the layout of a block. As mentioned before, the leaf blocks are ordered, thus the order of leaf blocks need to determine. Because the block tree is ordered, we determine the order of leaf blocks by depth-first traversal of the block tree. The order of leaf blocks is the same as the result of depth-first traversal. Figure 5.6 shows the result of depth-first traversal of the block tree in Figure 5.5 (b). After removing the root and intermediate blocks, we can get the sequence of leaf blocks.

Figure 5.6: Depth-first traversal of Figure 5.5 (b)

The contribution of different leaf blocks to the layout is different. For example in Figure 5.5 (a), \( b_1 \) is more important than any other leaf blocks. If \( b_1 \) disappeared the layout would change a lot. Conversely, if \( b_2 \) or \( b_3 \) disappeared the change of the layout is much less. Here,
we call this contribution “weight”. For a leaf block \( b_i \) the \( \text{Weight}(b_i) \) is calculated as in equation (5-1):

\[
\text{Weight}(b_i) = \frac{\text{Area}(b_i)}{\text{Area}(B)}
\]  

(5-1)

where \( B \) is the block that contains \( b_i \), \( \text{Area}(b_i) \) and \( \text{Area}(B) \) represent the area of \( b_i \) and \( B \) respectively. In other words, in a same block, the leaf block with greater area has greater weight.

In additional to leaf blocks, separators are also important to the layout. When determining separators, two rules must be followed:

Rule 1: The separator never crosses any blocks;

Rule 2: There must be blocks on both sides of each separator;

Based on the rules, here is an example to introduce the algorithm for determining separators. Figure 5.7 shows how to determine the separators of the block in Figure 5.5 (a). As shown in Figure 5.7 (a), we extend the four edges of the first leaf block \( b_1 \). The four extended edges are candidate separators. According to rule 2, \( S_{1,1}, S_{1,3} \) and \( S_{1,4} \) have blocks on only one side, but \( S_{1,2} \) has blocks on both sides (\( b_1 \) is on the upper of \( S_{1,2} \), \( b_2, b_3 \) and \( b_4 \) are on the lower of \( S_{1,2} \)). Thus \( S_{1,2} \) is the first separator, and will be saved into the array of separators. In Figure 5.7 (b), the separator \( S_{1,2} \) divides the block into two parts \( P_1 \) and \( P_2 \). Because \( P_1 \) only contains one leaf block, there cannot be another separator in \( P_1 \). Hence, we cut \( P_1 \) off and consider the leaf blocks in \( P_2 \). In \( P_2 \), \( b_2 \) becomes the first leaf block. Similarly, we extend the four edges of \( b_2 \). According to rule 2, \( S_{2,1} \) and \( S_{2,3} \) cannot be a separator. According to rule 1, \( S_{2,4} \) cannot be a separator, because it is crossing \( b_4 \). So \( S_{2,2} \) is the second separator and will be saved into the array of separators. In Figure 5.7 (c), \( S_{2,2} \) further divides the \( P_2 \) into two parts \( P_{2,1} \) and \( P_{2,2} \). We cut \( P_{2,2} \) off, because it contains only one leaf block. In \( P_{2,1}, b_2 \) is still the first leaf block. We extend the four edges of \( b_2 \). Obviously, only \( S_{2,4} \) can be the separator. Finally, \( S_{3,4} \) divides \( P_{2,1} \) into two parts \( P_{2,1,1} \) and \( P_{2,1,2} \) as shown in Figure 5.7 (d). As \( P_{2,1,1} \) and \( P_{2,1,2} \) both contain only one leaf block the algorithm terminates. As a result, the sequence of separators \( (S_{1,2}, S_{2,2}, S_{3,4}) \) are obtained. These separators are the same as the separators that are shown in Figure 5.5 (a).
According to the rules, we give the pseudocode of the algorithm for determining separators as shown in Figure 5.8. Lines (1-5) show the main body of the algorithm, the input is a block and the output is an sequence of separators. Lines (6-20) show the “separate” function which is the core of this algorithm. The “separate” function is a recursive function whose parameters are a rectangular region and a sequence of leaf blocks. When a separator is found, the function splits the current rectangular region into two smaller parts along the separator and divides the current set of leaf blocks into two subsets. After that, the smaller rectangular regions and subsets of leaf blocks will be separated further. If none of four extended edges are separators, the loop will go to the next leaf block. The function will end when all the sub-sets of leaf blocks contains only one leaf block.
Algorithm for determining separators

Input: a block $B$
Output: a sequence of separators

Begin
1. leafSet = getLeaves($B$);  // get all the leaf nodes of $B$
2. rectangle = getRect($B$);  // get the rectangular region of $B$
3. separatorSet = null;  // initialize the set of separators
4. Separate(rectangle, leafSet);
5. return separatorSet;

End

// find the separators
6. Separate(rectangle, leafSet) {
7.   if leafSet.length == 1
8.     return;
9.   foreach Li in leafSet
10.      extendedEdgeSet = getExtendedEdges(Li);
11.      separator = getSeparator(extendedEdgeSet);
12.      if separator != null then
13.         push(separator, separatorSet);
14.         {subRect_1, subRect_2} = split(rectangle, separator);
15.         {subLeafSet_1, subLeafSet_2} = divide(leafSet, separator);
16.         Separate(subRect_1, subLeafSet_1);
17.         Separate(subRect_2, subLeafSet_2);
18.         break;
19.  }

// return the four extended edges of the current block
20. getExtendedEdges(block);

// return a separator from extendedEdgeSet
21. getSeparator(extendedEdgeSet);

// split a rectangle into two parts along the separator
22. split(rectangle, separator);

// divide the leafSet into two smaller sub-sets along the separator
23. divide(leafSet, separator);

Figure 5.8: Algorithm for determining separators
Similar to the leaf blocks, each separator has weight. For a separator $S_i$ of block $B$, the $\text{Weight}(S_i)$ is calculated as in equation (5-2):

$$\text{Weight}(S_i) = \frac{\min\{\text{Area}(P_1), \text{Area}(P_2)\}}{\text{Area}(B)}$$  \hspace{1cm} (5-2)

where $P_1$ and $P_2$ are the two smaller parts that are separated by $S_i$. $\text{Area()}$ represents the area.

5.4.2 The Layout Tree of a Block

![Diagram](image)

Figure 5.9: The process for layout tree generation

According to the algorithm mentioned in previous section, it is easy to see that a separator can separate the current rectangular region and leaf blocks into two smaller parts. A separators can
be considered as a root of a tree, and the two smaller parts can be considered as the left subtree and the right subtree. Generally, if a separator is horizontal, the upper part is the left subtree and lower part is the right subtree. If a separator is vertical, the left part is the left subtree and right part is the right subtree. Therefore, the layout of a block can be regarded as a tree. We call the tree a “layout tree”. In this section, the layout tree will be introduced in detail.

We take the block in Figure 5.5 (a) as an example to introduce the process of generating a layout tree as shown in Figure 5.9. Let us suppose that the sequence of the separators \((S_1, S_2, S_3)\) have been determined. In Figure 5.9 (a) the first separator \(S_1\) splits the block into two parts \(P_1\) and \(P_2\). The separator \(S_1\) then is considered as the root, the upper part \(P_1\) is the left subtree and the lower part \(P_2\) is the right subtree. After that, the two subtrees are checked to see if they contain a separator. In Figure 5.9 (b), \(P_1\) only contains leaf block \(b_1\) and does not contain any separators. There is no need to separate \(P_1\) anymore, so \(P_1\) is replaced by \(b_1\). The right subtree \(P_2\) contains the second separator \(S_2\), so it needs to be separated further. Similarly, \(S_2\) is the root, the upper part \(P_{2_1}\) is the left subtree and the lower part \(P_{2_2}\) is the right subtree. In Figure 5.9 (c), \(b_4\) replaces the \(P_{2_2}\) that is because \(b_4\) is the only leaf block in \(P_{2_2}\). Then \(P_{2_1}\) is separated by \(S_3\) into \(P_{2_1_1}\) and \(P_{2_1_2}\). Finally, \(P_{2_1_1}\) is replaced by \(b_2\) and \(P_{2_1_2}\) is replaced by \(b_3\) because both \(P_{2_1_1}\) and \(P_{2_1_2}\) contain only one leaf block. Figure 5.9 (d) shows the final layout tree of the block in Figure 5.5 (a).

Based on the observation of the layout tree, we discover that a layout tree has the following features:

**Feature 1:** A layout tree is a weighted binary tree.

Because each separator divides the current rectangular region and leaf blocks into two smaller parts, the root node and intermediate nodes of a layout tree have two and only two child nodes. Each node has a weight. If a node is a leaf block node the weight can be calculated as in equation (5-1). If a node is a separator node the weight can be calculated as in equation (5-2). Therefore, a layout tree is a weighted binary tree.

**Feature 2:** In a layout tree, the root node and intermediate nodes are always the separator nodes, and the leaf nodes are always the leaf block nodes.

According to feature 1, every separator node has two and only two child nodes, so a separator node cannot be a leaf block. Contrarily, the leaf blocks cannot be separated any more. They do not have any child nodes, thus they can only be leaf node.

**Feature 3:** The layout tree can completely describe the relative positions of leaf blocks in a block. For any separator node, if the separator is horizontal the left subtree is always above the right subtree. If the separator is vertical, the left subtree is always on the left side of the
right subtree.

Feature 3 is the most important feature. It gives the layout tree the ability to accurately describe the relative positions of leaf blocks in a block. Therefore we can calculate the distance of two layout trees in order to compare the similarity of the layouts of two blocks.

5.4.3 The Similarity between Layout Trees

There are many algorithms to calculate the structural similarity between trees, of which the Tree Edit Distance (TED) is a simple and efficient algorithm [39]. We apply the TED algorithm to measure the similarity between layout trees.

The edit distance, \( \delta(F, G) \), between two forests \( F \) and \( G \) (\( F \) and \( G \) can be two trees) is defined as the minimum cost to transform \( F \) to \( G \) by using insertion, deletion, and replacement operations on nodes. Each edit operation is represented by \( (n_1 \rightarrow n_2) \), where \( n_i \) is an actual node or an empty node denoted by \( \varepsilon \). The operation is a node replacement if \( n_1 \neq \varepsilon \) and \( n_2 \neq \varepsilon \), a node deletion if \( n_2 = \varepsilon \), and a node insertion if \( n_1 = \varepsilon \). Given a metric cost function \( \gamma \) defined on pairs of labels, we define the cost of an edit operation by setting \( \gamma(n_1 \rightarrow n_2) = \gamma(n_1, n_2) \). The tree edit distance can be calculated as in formula (5-3):

\[
\delta(\varnothing, \varnothing) = 0 \\
\delta(F, \varnothing) = \delta(F, \varnothing) + \gamma(\varepsilon \rightarrow \varepsilon) \\
\delta(\varnothing, G) = \delta(\varnothing, G) + \gamma(\varepsilon \rightarrow \varepsilon) \\
\text{if } F \text{ is not a tree or } G \text{ is not a tree:} \\
\delta(F, G) = \min \left\{ \begin{array}{l}
\delta(F - v, G) + \gamma(\varepsilon \rightarrow \varepsilon), \\
\delta(F, G - w) + \gamma(\varepsilon \rightarrow \varepsilon) \\
\delta(F_v, G_w) + \delta(F - v, G - w)
\end{array} \right\} \\
\text{if } F \text{ is a tree and } G \text{ is a tree:} \\
\delta(F, G) = \min \left\{ \begin{array}{l}
\delta(F - v, G) + \gamma(\varepsilon \rightarrow \varepsilon), \\
\delta(F, G - w) + \gamma(\varepsilon \rightarrow w), \\
\delta(F - v, G - w) + \gamma(\varepsilon \rightarrow \varepsilon)
\end{array} \right\}
\]

where \( v \) and \( w \) are either both the left most or right most root nodes of the respective forest. \( F_v \) is the subforest rooted in node \( v \) of \( F \), and \( G_w \) is the subforest rooted in node \( w \) of \( G \). \( F - v \) denotes the forest obtained by deleting \( v \) from \( F \), and \( G - w \) denotes the forest obtained by deleting \( w \) from \( G \).

Basing on the TED algorithm and the three features of a layout tree, we introduce the cost
functions to calculate the cost of operations. Equation (5-4) and equation (5-5) show the cost functions of insertion and deletion operations:

\[ \text{Insert}(n) = \text{Weight}(n) \]  
(5-4)

\[ \text{Delete}(n) = \text{Weight}(n) \]  
(5-5)

where \( n \) is a node of a layout tree, and \( \text{Weight}(n) \) is the weight of \( n \). That is to say if \( n \) is inserted into a tree or \( n \) is deleted from a tree the cost will be the weight of \( n \). The greater the weight is, the greater the cost will be. Similarly, the cost function of the replacement operation is calculated as in equation (5-6):

\[ \text{Replace}(n_1, n_2) = \begin{cases} 0 & (n_1 \text{ sim } n_2) \\ \text{Weight}(n_1) + \text{Weight}(n_2) (n_1 \text{ dif } n_2) & \end{cases} \]  
(5-6)

where \( n_1 \text{ sim } n_2 \) represents \( n_1 \) and \( n_2 \) are similar, and \( n_1 \text{ dif } n_2 \) represents \( n_1 \) and \( n_2 \) are not similar. As introduced before, there are two types of nodes in layout tree: separator nodes and leaf block nodes. Moreover, there are two directions of separator nodes: horizontal and vertical. As for leaf block nodes, we roughly divide them into two types: image nodes and text nodes. The following rules are used to determine whether \( n_1 \) and \( n_2 \) are similar or not:

Rule 1: If node \( n_1 \) and node \( n_2 \) are different types (one is a separator node and the other one is a leaf block node), then \( n_1 \text{ dif } n_2 \).

Rule 2: If both node \( n_1 \) and \( n_2 \) are separator nodes, and the directions of \( n_1 \) and \( n_2 \) are different (one is horizontal and the other one is vertical) then \( n_1 \text{ dif } n_2 \). Otherwise \( n_1 \text{ sim } n_2 \).

Rule 3: If both node \( n_1 \) and \( n_2 \) are leaf block nodes, and the types of \( n_1 \) and \( n_2 \) are different (one is image node and the other one is text node) then \( n_1 \text{ dif } n_2 \).

Rule 4: If both node \( n_1 \) and \( n_2 \) are image nodes, then \( n_1 \text{ sim } n_2 \).

Rule 5: If both node \( n_1 \) and \( n_2 \) are text nodes, and \( n_1 \) and \( n_2 \) have the same font and font size, then \( n_1 \text{ sim } n_2 \). Otherwise \( n_1 \text{ dif } n_2 \).

After the edit distance of two layout trees are figured out, the similarity of them can be calculated. Let \( T_1 \) and \( T_2 \) be two layout trees. \( \delta(T_1, T_2) \) is the edit distance of \( T_1 \) and \( T_2 \). The similarity of \( T_1 \) and \( T_2 \) can be calculated as in equation (5-7):

\[ \text{Sim}(T_1, T_2) = \frac{\delta(T_1, T_2)}{\text{Max}\{\sum \text{Weight}(n_i), \sum \text{Weight}(m_j)\}} \]  
(5-7)
where \( n_i \) is a node in \( T_1 \) and \( m_j \) is a node in \( T_2 \). The denominator of equation (5-7) represents the greater one of total weight of the layout trees \( T_1 \) and \( T_2 \). The similarity of \( T_1 \) and \( T_2 \) has the following features:

1. \( \text{Sim}(T_1, T_2) \in [0, 1] \)
2. If \( \text{Sim}(T_1, T_2) \) is closer to 0, then \( T_1 \) and \( T_2 \) are more similar; if \( \text{Sim}(T_1, T_2) \) is closer to 1, then \( T_1 \) and \( T_2 \) are more different.

We introduce a threshold \( \alpha \). If \( \text{Sim}(T_1, T_2) \leq \alpha \), then \( T_1 \) and \( T_2 \) are similar, otherwise they are different. Given two blocks \( B_1 \) and \( B_2 \), their layout trees are \( T_1 \) and \( T_2 \). The similarity of the layout of \( B_1 \) and \( B_2 \) can be calculated as in equation (5-8):

\[
\text{LayoutSim}(B_1, B_2) = \text{Sim}(T_1, T_2)
\] (5-8)

5.5 Recognizing Data Record Blocks

In this section, we introduce the method to recognize data record blocks from a deep web page using the layout tree and other visual features mentioned in the previous section. Our method can be roughly divided into four steps:

Step 1: Cluster the similar layout blocks in the same depth of block tree;

Step 2: Refine the clusters of Step 1;

Step 3: Identify the data records from the block clusters.

5.5.1 Clustering of Similar Layout Blocks

After the block tree of a web page is generated (see Chapter 2), we cluster the similar layout blocks. The blocks of data records are often at the same depth in the block tree. Thus we first...
calculate the similarity of the blocks that are at the same depth of block tree as shown in Figure 5.10. The computation of layout tree similarity is a time taken process. In order to reduce the time consuming, we only calculate the similarity of blocks that have the SSC feature which is mentioned in Section 5.3. This SSC feature has two aspects: similar shape and similar coordinate. Given two block \( B_1 \) and \( B_2 \), the \( SSC(B_1, B_2) \) can be calculated as in equation (5-9):

\[
SSC(B_1, B_2) = ShapeSim(B_1, B_2) \times CoorSim(B_1, B_2)
\]  

(5-9)

where \( ShapeSim(B_1, B_2) \) represents the similarity of the shape, and \( CoorSim(B_1, B_2) \) represents the similarity of the coordinate. As introduced in Section 5.3, if two blocks have similar height or width, we consider them to have a similar shape. Similarly, if two blocks have similar horizontal or vertical coordinate, we consider that they have similar coordinate. The \( ShapeSim(B_1, B_2) \) and \( CoorSim(B_1, B_2) \) can be calculated as in equation (5-10) and equation (5-11):

\[
ShapeSim(B_1, B_2) = \min \left\{ \frac{|height(B_1) - height(B_2)|}{\max\{height(B_1), height(B_2)\}}, \frac{|width(B_1) - width(B_2)|}{\max\{width(B_1), width(B_2)\}} \right\}
\]  

(5-10)

\[
CoorSim(B_1, B_2) = \min \left\{ \frac{|left(B_1) - left(B_2)|}{\max\{width(B_1), width(B_2)\}}, \frac{|top(B_1) - top(B_2)|}{\max\{height(B_1), height(B_2)\}} \right\}
\]  

(5-11)

where \( height(B_i) \) is the height of \( B_i \), \( width(B_i) \) is the width of \( B_i \), \( left(B_i) \) is the horizontal coordinate of \( B_i \), and \( top(B_i) \) is the vertical coordinate of \( B_i \). (See Section 2.1 for the detailed definition of \( height \), \( weight \), \( top \), and \( left \)). Empirically, if \( SSC(B_1, B_2) < 0.1 \), we consider \( B_1 \) and \( B_2 \) have similar shape and coordinate.
Algorithm for clustering similar layout blocks

Input: a block tree $T$

Output: a set of similar layout block clusters $clusterSet$

Begin
1. $clusterSet = null$
2. $d = \text{Depth}(T)$ \hspace{1em} //get the total depth of $T$
3. loop $i = 1 : d$
   4. $\text{NodeList} = \text{getNodes}(T, i)$ \hspace{1em} //get the nodes in $i$th depth
   5. loop $m = 1 : \text{Length(NodeList)} - 1$
      6. loop $n = m + 1 : \text{Length(NodeList)}$
         7. $B_1 = \text{NodeList}[m]$
         8. $B_2 = \text{NodeList}[n]$
         9. if $\text{SSC}(B_1, B_2) \geq 0.1$
            10. then continue
         11. if both $B_1$ and $B_2$ are already in cluster
            12. then continue
         13. if $\text{LayoutSim}(B_1, B_2) > \alpha$
            14. then continue
         15. if $B_1$ is already in a cluster $C$
            16. then push($B_2$, $C$)
         17. if $B_2$ is already in a cluster $C$
            18. then push($B_1$, $C$)
         19. if both the $B_1$ and $B_2$ are not in cluster
            20. then create a new cluster $C'$
            21. push($B_1$, $C'$)
            22. push($B_2$, $C'$)
            23. push ($C'$, $clusterSet$)
         loop end
      loop end
   loop end
4. loop end
5. return $clusterSet$
End

Figure 5.11: Algorithm for clustering similar layout blocks

Figure 5.11 shows the detailed algorithm for clustering similar layout blocks. Line (3-26) is a loop to cluster the similar layout blocks in each depth. Line (5-24) is to compare every two blocks in the same depth. Line (9-10) is to calculate the similarity of SSC feature of the
two blocks. If their SSC ≥ 0.1, the algorithm will go to next loop. If their SSC < 0.1, the algorithm will judge whether the two blocks has been clustered already in line (11-12). If not, the similarity of their layout tree will be calculated. If the similarity LayoutSim(B₁, B₂) > α, then the two blocks are regarded as similar layout block and should be clustered in the same group. Line (15-23) is to cluster the two blocks into the same group. If one of the two blocks has been clustered into a group, then the other block will be clustered into the same group. If both two blocks have not been clustered into any group yet, then create a new group and cluster the two blocks into this new group. As result, the clusters of similar layout blocks can be obtained.

5.5.2 Refining the Clusters of Similar Layout Blocks

After the clusters are obtained, we need to refine them as the following steps:

Step 1: Given a cluster $C = \{B₁, ..., Bₙ\}$, if $\forall Bᵢ \in C$ and $Bᵢ$ is a leaf block, then delete $C$. Since the leaf blocks do not contain other blocks, their layout trees have only one node. We consider this kind of layout tree is meaningless. If all elements in a cluster are leaf blocks, the cluster should be deleted.

Step 2: Given two clusters $C₁ = \{B₁₁, ..., B₁ₙ\}$ and $C₂ = \{B₂₁, ..., B₂ₘ\}$, if $\exists B₁ᵢ \in C₁$, $B₂ⱼ \in C₂$, and $B₁ᵢ \subset B₂ⱼ$, then we remove $B₂ⱼ$ from $C₂$.

Step 2 aims to remove the blocks that contains other similar layout blocks.

Step 3: Given two clusters $C₁ = \{B₁₁, ..., B₁ₙ\}$ and $C₂ = \{B₂₁, ..., B₂ₘ\}$, if $\exists B₁ᵢ \in C₁$, $B₂ⱼ \in C₂$, and $SSC(B₁ᵢ, B₂ⱼ) < 0.1$, $LayoutSim(B₁ᵢ, B₂ⱼ) < α$, then we combine $C₁$ and $C₂$.

Since the algorithm clusters the similar layout blocks in the same depth of a block tree. There may exist other similar layout blocks in different depth of the block tree. Therefore, we combine these clusters in which the blocks have similar SCC feature and similar layout.

5.5.3 Detection of Data Record Region

After all clusters have been refined, the data record region needs to be detected from the block clusters. The LAMSB feature is used to determine which cluster contains the data records. This feature is mentioned in section 5.3. It has two aspects: the largest area and most similar layout blocks. Given a set of clusters $S = \{C₁, ..., Cₙ\}$, we use $AreaWeight(Cᵢ)$ to measure the area weight of $Cᵢ$, and use $NumWeight(Cᵢ)$ to measure the number weight of $Cᵢ$. We calculate the two weight measures as in equation (5-12) and equation (5-13):
\[\text{AreaWeight}(C_i) = \frac{\text{Area}(C_i)}{\text{TotalArea}(S)} \quad (5-12)\]

\[\text{NumWeight}(C_i) = \frac{\text{Num}(C_i)}{\text{TotalNum}(S)} \quad (5-13)\]

where \(\text{Area}(C_i)\) is the total area of blocks in cluster \(C_i\), \(\text{TotalArea}(S)\) is the total area of all the clusters in \(S\). \(\text{TotalNum}(S)\) is the total number of blocks of all the clusters in \(S\). \(\text{Num}(C_i)\) is the total number of blocks in cluster \(C_i\). Here, we introduce a harmonic function \(\text{Weight}(C_i)\) of \(\text{AreaWeight}(C_i)\) and \(\text{NumWeight}(C_i)\). We call this harmonic function LAMSB weight. It can be calculated as in equation (5-14):

\[\text{Weight}(C_i) = \frac{2 \times \text{AreaWeight}(C_i) \times \text{NumWeight}(C_i)}{\text{AreaWeight}(C_i) + \text{NumWeight}(C_i)} \quad (5-14)\]

We calculate the LAMSB weight of all the clusters in \(S\), and determine the cluster that has the highest score as the data record cluster. The blocks in the chosen cluster are the data record blocks.

### 5.6 Experiment and Evaluation

In this section, experiments are conducted to evaluate the effectiveness of the proposed method. Our experiments are done on a CORE i5 2.6 GH, 4GB PC. The contents of the experiments are as follows:

1. Determining the optimal similarity threshold \(\alpha\) that is mentioned in Section 5.4.3;
2. Evaluating the effectiveness of the proposed method in different web sites using the determined optimal similarity threshold \(\alpha\).

#### 5.6.1 Data Set

We collected data from 10 different web sites, in order to guarantee the diversity of the data set. The 10 web sites can be roughly divided into four types: online shopping sites, news sites, video sites, SNS and blog sites. Table 5.1 shows the sites and URLs. These sites can accept user’s query, and search results are represented in the form of data record. For each site, we submitted 10 queries and collected one search result page for each query. Finally, we collected \(10 \times 10 = 100\) pages as the data set.
Table 5.1: The 10 web sites of dataset

<table>
<thead>
<tr>
<th>Type</th>
<th>Site</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Shopping Sites</td>
<td>Amazon</td>
<td><a href="http://www.amazon.co.jp/">http://www.amazon.co.jp/</a></td>
</tr>
<tr>
<td></td>
<td>Rakuten</td>
<td><a href="http://www.rakuten.co.jp/">http://www.rakuten.co.jp/</a></td>
</tr>
<tr>
<td></td>
<td>Kakaku</td>
<td><a href="http://kakaku.com/">http://kakaku.com/</a></td>
</tr>
<tr>
<td>News Sites</td>
<td>Google news</td>
<td><a href="https://news.google.com/">https://news.google.com/</a></td>
</tr>
<tr>
<td></td>
<td>Yahoo news</td>
<td><a href="http://headlines.yahoo.co.jp/">http://headlines.yahoo.co.jp/</a></td>
</tr>
<tr>
<td></td>
<td>Goo news</td>
<td><a href="http://news.goo.ne.jp/">http://news.goo.ne.jp/</a></td>
</tr>
<tr>
<td>Video Sites</td>
<td>YouTube</td>
<td><a href="http://www.youtube.com/">http://www.youtube.com/</a></td>
</tr>
<tr>
<td></td>
<td>MSN Video</td>
<td><a href="http://video.jp.msn.com/">http://video.jp.msn.com/</a></td>
</tr>
<tr>
<td>SNS and Blog Sites</td>
<td>Twitter</td>
<td><a href="https://twitter.com/">https://twitter.com/</a></td>
</tr>
<tr>
<td></td>
<td>Laplog</td>
<td><a href="http://www.yaplog.jp/">http://www.yaplog.jp/</a></td>
</tr>
</tbody>
</table>

5.6.2 Determining the Optimal Similarity Threshold of Layout Tree

In this experiment, we aim to determine the optimal similarity threshold of layout trees. We let the threshold $\alpha$ be 0.1 to 1, and recognize data record blocks from the collected pages using proposed method. Since we consider the area of blocks is an important factor, thus we utilized the area of blocks to calculate the precision, recall and F-measure of the proposed method using different threshold. Given a Web page $P$ and threshold $\alpha_i$, the total area of data record blocks in page $P$ is denoted $TotalRecordArea(P)$ the total area of extracted blocks is denoted $ExtractArea(P, \alpha_i)$, in which the total area of extracted data record blocks is denoted $RecordArea(P, \alpha_i)$. The precision, recall and F-measure are defined as follows:

$$Precision(P, \alpha_i) = \frac{RecordArea(P, \alpha_i)}{ExtractArea(P, \alpha_i)}$$

$$Recall(P, \alpha_i) = \frac{RecordArea(P, \alpha_i)}{TotalRecordArea(P, \alpha_i)}$$

$$F(P, \alpha_i) = \frac{2 \times Precision(P, \alpha_i) \times Recall(P, \alpha_i)}{Precision(P, \alpha_i) + Recall(P, \alpha_i)}$$
After the precision, recall, and F-Measure of each page is determined, we calculate the average value of 100 pages denoted \( Precision(\alpha_i) \), \( Recall(\alpha_i) \), and \( F(\alpha_i) \).

Table 5.2 shows the experiment results and Figure 5.12 is a diagram illustrating the results. The \( Precision(\alpha_i) \) and \( F(\alpha_i) \) reach a maximum when \( \alpha = 0.4 \). Here, \( Precision(0.4) = 1.0000 \) does not mean that there are no false blocks in the extracted results. Because the area of false blocks are too small and \( Precision(0.4) \) is only rounded to four decimal places, the \( Precision(0.4) \) became 1.0000.

It should be noted that the change of the curves is different, when \( \alpha \) is in different intervals. When \( \alpha \) is in the interval \([0.1, 0.4]\), the precision increases gradually. Compared with \( Precision(0.1) \), \( Precision(0.4) \) only changed 2.8%. While \( \alpha \) is in the interval \([0.4, 1]\), the precision decreases steeply. Compared with \( Precision(0.4) \), \( Precision(1.0) \) changed 7.1%. In other words, the ability of layout tree to eliminate different layout blocks decreases steeply when \( \alpha \) is greater than 0.4. Conversely, when \( \alpha \) is in the interval \([0.1, 0.4]\), the recall increases steeply. Compared with \( Recall(0.1) \), \( Recall(0.4) \) changed 17.9%. While \( \alpha \) is in the interval \([0.4, 1]\), the recall changes gradually. Compared with \( Recall(0.4) \), \( Recall(1.0) \) only changed 1.8%. It means that when \( \alpha \) is less than 0.4, the ability of layout tree to recognize the similar layout blocks significantly improves while \( \alpha \) increases. If \( \alpha \) is greater than 0.4, the ability of layout tree to recognize similar layout blocks improves gradually. Therefore, when \( \alpha = 0.4 \), the layout tree is optimal for recognizing similar layout blocks.

Table 5.2: The average values of different thresholds

<table>
<thead>
<tr>
<th>( \alpha_i )</th>
<th>( Precision(\alpha_i) )</th>
<th>( Recall(\alpha_i) )</th>
<th>( F(\alpha_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.9726</td>
<td>0.8299</td>
<td>0.8957</td>
</tr>
<tr>
<td>0.2</td>
<td>0.9874</td>
<td>0.8725</td>
<td>0.9264</td>
</tr>
<tr>
<td>0.3</td>
<td>0.9900</td>
<td>0.8942</td>
<td>0.9396</td>
</tr>
<tr>
<td>0.4</td>
<td>1.0000</td>
<td>0.9783</td>
<td>0.9890</td>
</tr>
<tr>
<td>0.5</td>
<td>0.9747</td>
<td>0.9838</td>
<td>0.9792</td>
</tr>
<tr>
<td>0.6</td>
<td>0.9742</td>
<td>0.9724</td>
<td>0.9735</td>
</tr>
<tr>
<td>0.7</td>
<td>0.9590</td>
<td>0.9827</td>
<td>0.9707</td>
</tr>
<tr>
<td>0.8</td>
<td>0.9559</td>
<td>0.9835</td>
<td>0.9695</td>
</tr>
<tr>
<td>0.9</td>
<td>0.9369</td>
<td>0.9812</td>
<td>0.9585</td>
</tr>
<tr>
<td>1.0</td>
<td>0.9287</td>
<td>0.9961</td>
<td>0.9612</td>
</tr>
</tbody>
</table>
5.6.3 Performance Evaluation in Different Web Sites

After the optimal similarity threshold of layout trees is determined, we conducted another experiment to evaluate the effectiveness of the proposed method in different web sites using the optimal similarity threshold. Since the area of blocks is not the same in different sites, we do not use the area of blocks to calculate the precision, recall, and F-measure in order to conduct a fair comparison. Given a page $P$, the total number of data record blocks is denoted $TotalRecordNum(P)$, and the total number of extracted blocks by the proposed method is denoted $ExtractNum(P)$, in which the number of true data record blocks is denoted $RecordNum(P)$. The $Precision(P)$, $Recall(P)$ and $F-Measure(P)$ were used as the evaluation criteria, and the definitions of them are shown as follows:

$$Precision(P) = \frac{RecordNum(P)}{ExtractNum(P)}$$

Figure 5.12: The diagram of average values and different thresholds
\[
Recall(P) = \frac{RecordNum(P)}{TotalRecordNum(P)}
\]

\[
F(P) = \frac{2 \times Precision(P) \times Recall(P)}{Precision(P) + Recall(P)}
\]

We set the threshold \( \alpha = 0.4 \) and calculate the average precision, recall and F-measure of 10 pages in each site. Table 5.3 shows the experiment result of each site.

Table 5.3: The average values of different sites

<table>
<thead>
<tr>
<th>Type</th>
<th>Site</th>
<th>Precision</th>
<th>Recall</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Shopping Sites</td>
<td>Amazon</td>
<td>0.9889</td>
<td>0.9528</td>
<td>0.9705</td>
</tr>
<tr>
<td></td>
<td>Rakuten</td>
<td>0.9814</td>
<td>0.9653</td>
<td>0.9732</td>
</tr>
<tr>
<td></td>
<td>Kakaku</td>
<td>0.9882</td>
<td>0.9712</td>
<td>0.9796</td>
</tr>
<tr>
<td>News Sites</td>
<td>Google news</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td>Yahoo news</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td>Goo news</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Video Sites</td>
<td>YouTube</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td></td>
<td>MSN Video</td>
<td>1.0000</td>
<td>0.9823</td>
<td>0.9910</td>
</tr>
<tr>
<td>SNS and Blog Sites</td>
<td>Twitter</td>
<td>0.9728</td>
<td>0.9712</td>
<td>0.9720</td>
</tr>
<tr>
<td></td>
<td>Laplog</td>
<td>0.9912</td>
<td>1.0000</td>
<td>0.9955</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>0.9923</strong></td>
<td><strong>0.9843</strong></td>
<td><strong>0.9882</strong></td>
</tr>
</tbody>
</table>

In the experiment result, the average values of the three news sites reached 1.0000 and there are no false blocks in the results of the three news sites. On the other hand, the average recalls of the three online shopping sites are relatively low. Especially, the average recall of Amazon is only 0.9528 which is the lowest average recall. Based on our observation, the layouts of data record blocks in three news sites are very simple, and almost every data record blocks have the identical layout. Thus the proposed method can work very well to recognize the data record blocks. Conversely, the layouts of data record blocks in the online shopping sites are very complex. For example, some product records may only include the product picture, product name and price, while some product records may include many additional contents, e.g. product details, user reviews, order information, etc. Thus the proposed method may cluster these blocks into different group by mistake. However, the average recalls of online shopping sites did not considerably decreased and they still reached more than 0.95. Based on the average values of all the 10 sites, we can conclude that the proposed method is
5.7 Conclusion

In this chapter, we proposed a novel method to recognize data record blocks, which is an important step of web page segmentation. These data record blocks always have independent and complete semantics. For example, they may be product records of an online shopping page and each block represents an individual product. Thus these blocks should be regarded as semantical segments. Since these data record blocks of a page are generated from database, they often have similar layout. The layout of a block describes the arrangement and relative positions of its leaf blocks. Therefore, we recognize the data record blocks by analyzing their layout. We use separators and leaf blocks to transform a block into a layout tree. By calculating the similarity of layout trees, we clustered the blocks that have similar layout. We used the LAMSB weight to measure these clusters. Finally, the cluster which has the highest LAMSB weight can be extracted as the data record block cluster. The experiments conducted drew the following conclusions:

(1) The optimal layout tree similarity threshold is 0.4.

(2) The proposed method is effective to recognize the data record blocks and has a good fault-tolerant and identification ability, although, its effectiveness may be different for different sites.
Chapter 6

Web Page Segmentation Using Visual Semantics

6.1 Introduction

In this thesis, the goal of web page segmentation is to transform the nested block structure into independent segments that are not nested. In other words, a segment cannot be a descendant or an ascendant of another segment. In this way, each segment can be indexed directly, and the nested search results can also be eliminated. In Chapter 5, we introduced a novel method to recognize the data record blocks, which is an important step for web page segmentation. In this chapter, we will introduce a novel method for web page segmentation using visual semantics.

The early techniques of web page segmentation are mainly based on machine learning algorithms [48][51][55][56] and rule-based heuristics [49][50][52][53][54][57][58][61][32]. Because of the small scale training data set, machine-learning-based methods can be only applied in some certain fields of web pages. The heuristics-based approaches involve simple rule-based heuristics either by interpreting the meaning of tag structures or visual analysis. While a heuristic approach might work well on small sets of pages, it isn’t suitable for large-scale sets of pages. For example tags such as <TABLE> and <P> are used not only for content markup but also for layout structure presentation. It is difficult to obtain the appropriate segmentation granularity. Visual heuristic-based approaches rely on visual cues from browser renderings. Most of the vision-based methods focus on the location, size or font features of elements. However, most of these methods involve some set of heuristics. These heuristics typically utilize many features present on a Web page. While a heuristic approach might work well on small sets of pages, it isn’t suitable for large-scale sets of pages [52].

In order to overcome the limitations mentioned above, this chapter introduces a formulated
method using visual semantics. Let us see an example that is shown in Figure 6.1.

Figure 6.1 shows the segments of a news web page. Intuitively, the news web page can be divided into five segments: (i) header, (ii) navigation bar, (iii) news, (iv) advertisement, and (v) related news list. Based on our observation, we noticed two facts:

1. Different segments always contain different contents that have different visual features. For example: in Figure 6.1, the navigation bar contains a list of short links; the news contains a long text; the header and advertisement are two big pictures with different size and position; the related news list contains a list of links (they are longer than the links of the navigation bar). Moreover, the font size in different segments is not the same.
The similar blocks in a segment are neatly arranged. For example: the blocks of navigation bar (they are the short links) are neatly arranged in horizontal direction; the blocks of news (they are the paragraphs) are neatly arranged in vertical direction; the blocks of related news list (they are the links) are neatly arranged in vertical direction. In other words, if a block contains similar blocks that are neatly arranged, then the block is likely to be a segment.

Due to the different visual features, humans can easily identify each of the segments without any descriptions. We call these visual features visual semantics. However, these semantics are intuitive and human friendly. In other words, they are not machine friendly and therefore difficult to be understood by computers. Therefore, we use two formulated measures to represent two visual semantics: Seam Degree is used to describe how neatly the blocks are arranged; Content Similarity is used to describe the visual similarity of contents between the blocks. It should be noted that data record blocks (see Chapter 5) do not have the two visual semantics. The data record blocks in a web page always have the similar contents and their child blocks are not always neatly arranged. The data record blocks can be considered as a special case. Because of this, we utilize layout tree to recognize these blocks in Chapter 5. Actually, layout tree is also a formulated measure to describe the visual semantic of layout feature. In this chapter, we will introduce the web page segmentation method based on the three visual semantics.

6.2 Related Work

Web page segmentation has a variety of benefits for potential web applications, such as: browsing web pages on mobile devices [48][49][50]; detecting duplicate web pages [51], information extraction [52][53][54], etc. Recognizing the importance of Web page segmentation, in the past few years, there has been plenty of work on automatic web page segmentation.

Some of the early approaches are based on machine learning algorithms [48][51][55][56]. These approaches segment pages by training the clues from DOM or simple vision cues. Machine-learning-based approaches can only be applied in some certain fields of web pages, because of the limitation of the training data set. Because these algorithms need to be trained, they can be regarded as semi-automatic approaches.

In order to automatically segment web pages, heuristics-based approaches were proposed.
Some of the heuristics-based approaches use HTML structure tags or DOM tree to segment a web page [50][57][58][61]. These methods also have some limitations, for example: these methods may falsely separate closely related contents and combine unrelated contents together. Some other heuristics-based approaches rely on visual cues from browser renderings [49][52][53][54][32]. Most of them focus on the location, size or font cues of web pages. Hereinto, VIPS [32] is considered to be the most representative visual-cue-based algorithm. It has three steps: first, a web page is recursively divided into blocks by using a number of heuristics; second, horizontal and vertical separators are determined; third, the structure of the page is constructed. These approaches can make good use of the visual features of web pages. However, heuristics are often based on simple models that cannot be generalized. In other words, even through a heuristic approach might work well on small sets of pages it isn’t suitable for large sets of pages.

Because of the limitation of heuristics-based approaches, many non-heuristics-based approaches [59][60][62][63] were proposed. X. Liu et al. [59] proposed a Gomory-Hu Tree based Web page segmentation algorithm. The algorithm firstly extracts vision and structure information from a web page to construct a weighted undirected graph, whose vertices are the leaf nodes of the DOM tree and the edges represent the visible position relationship between vertices. Then it partitions the graph with a Gomory-Hu tree based clustering algorithm. G. Hattori et al. J. Kong et al. [60] proposed Spatial Graph Grammar (SGG) to perform the semantic grouping and interpretation of segmented screen objects. Instead of analyzing HTML source codes, they applied an image processing technology to recognize atomic interface objects from the screenshot of an interface and produce a spatial graph, which records significant spatial relations among recognized objects. However, there are not enough quantified experiment results to indicate that SGG is effective to segment any kinds of web pages. J. Kang et al. [62] proposed repetition-based web page segmentation method. They consider the repetitive tag patterns to be key patterns in the DOM tree structure of a page. By detecting key patterns in a page and generating virtual nodes to correctly segment nested blocks, the method can segment pages into logical blocks. However, this method is only suitable for the pages that contain repetitive patterns. C. Kohlschütter et al. [63] utilized the notion of text-density as a measure to identify the individual text segments of a web page. Although, his method can reduce the problem to solving a 1D-partitioning task, it can be used for small-scale pages that have certain patterns.

Our work can be classified as a formulated and vision-based approach. Different from the visual-cue-based method, e.g. VIPS, our work formulates the visual feature as quantified and formulized measures. Based on these measures, the proposed approach can divide web pages
6.3 Seam Degree of Blocks

6.3.1 The Seam Degree of Two Adjacent Blocks

Because the blocks are visible rectangles in a web page, they are always arranged by certain rules. As introduced in Section 6.1, if a block contains similar blocks that are neatly arranged, then the block is likely to be a segment. For any two given blocks, their arrangements can be classified into three types as shown in Figure 6.2.

![Figure 6.2: Three arrangement types of two blocks](image)

(a) Not adjacent  (b) Partly adjacent  (c) Fully adjacent

In Figure 6.2, $a_1$ and $a_2$ are not adjacent, thus we consider them to be visually irrelevant. In Figure 6.2 (b) (c), $b_1$ and $b_2$, $c_1$ and $c_2$ are adjacent blocks. Intuitively, we consider $c_1$ and $c_2$ are neater than $b_1$ and $b_2$. Suppose there is a minimum rectangle that can just cover the two blocks in Figure 6.2 (b) and Figure 6.2 (c). $c_1$ and $c_2$ can fully fill up the minimum rectangle, but $b_1$ and $b_2$ cannot fill up it. It is known that each segment has a corresponding rectangle appearing in the page. We utilize seam degree to describe how neatly the two blocks are arranged. If two blocks are partly or fully adjacent, we call the length of adjacent section “seam length”.

For two given adjacent blocks $B_1$ and $B_2$, if they are upper-lower adjacent, the seam degree $SD(B_1, B_2)$ can be calculated as in equation (6-1):
where \( \text{SeamLength}(B_1, B_2) \) represents the seam length of \( B_1 \) and \( B_2 \), and \( \text{width}(B_i) \) represents the width of \( B_i \). Similarly, if \( B_1 \) and \( B_2 \) are left-right adjacent, the seam degree \( SD(B_1, B_2) \) can be calculated as in equation (6-2):

\[
SD(B_1, B_2) = \frac{\text{SeamLength}(B_1, B_2)^2}{\text{width}(B_1) \times \text{width}(B_2)}
\]

(6-2)

where \( \text{height}(B_i) \) represents the height of \( B_i \).

\( SD(B_1, B_2) \) is between 0 and 1. If \( SD(B_1, B_2) \) is closer to 1, then \( B_1 \) and \( B_2 \) are arranged more neatly.

6.3.2 The Average Seam Degree of Adjacent Child Blocks in a Block

If a block has child blocks, the average seam degree of adjacent child blocks can indicate how neatly the child blocks are arranged. For a given block \( B \), the set of child blocks in \( B \) is \( \text{Child}(B) = \{b_1, b_2, \ldots, b_n\} \). If two child blocks are adjacent, we count one pair. Let us assume that there are \( m \) pairs of adjacent child blocks. The averaging seam degree \( \text{AvgSD}(B) \) can be calculated as in equation (6-3):

\[
\text{AvgSD}(B) = \frac{\sum SD(b_i, b_j)}{m}
\]

(6-3)

where \( b_i \) and \( b_j \) are two adjacent child blocks. \( \text{AvgSD}(B) \) degree is also between 0 and 1. If it is closer to 1, child blocks of block \( B \) are arranged more neatly. Figure 6.3 shows two blocks with different average seam degree. There are four child block in both block \( A \) and \( B \). Block \( A \) contains three pairs of adjacent child blocks in: \( (a_1, a_2) \), \( (a_2, a_3) \), and \( (a_3, a_4) \). In block \( A \), since \( \text{SeamLength}(a_1, a_2) = \text{width}(a_1) = \text{width}(a_2) \), according to equation (6-1), \( SD(a_1, a_2) = 1 \). Similarly, \( SD(a_2, a_3) = SD(a_3, a_4) = 1 \). According to equation (6-3), \( \text{AvgSD}(A) = (1+1+1)/3 = 1 \). There are five pairs of adjacent child blocks in block \( B \): \( (b_1, b_2) \), \( (b_1, b_3) \), \( (b_2, b_3) \), \( (b_2, b_4) \), and \( (b_3, b_4) \), where \( \text{width}(b_1) = \text{width}(b_3) = \text{width}(b_4) = 1/2 \text{width}(b_1) \) and \( \text{height}(b_2) = \text{height}(b_4) = 1/2 \text{height}(b_2) \). According to equation (6-1), \( SD(b_1, b_2) = SD(b_1, b_3) = SD(b_2, b_3) = SD(b_2, b_4) = 0.5 \), and \( SD(b_3, b_4) = 1 \). According to equation (6-3), \( \text{AvgSD}(B) = (0.5+0.5+0.5+0.5+1)/5 = 0.6 \). Therefore, the child blocks of block \( A \) are arranged more neatly than that of block \( B \).
6.4 Content Similarity of Blocks

6.4.1 The Content Vectors of a Block

Blocks with different semantics always have different types of contents. For example, a navigation bar has a list of short link text; an advertisement has a big picture; a user registration form has some text boxes, pull-down menus, buttons, etc. These different contents have different visual features. If the contents of two blocks are similar, the two blocks have a high content coherent degree. We introduce the Content Similarity to describe the content coherent degree. We roughly classify the contents into four categories:

1. Text Contents (TC): all the text falls into this category, except the text that contains a hyper link.
2. Link Text Contents (LTC): the text that contains a hyper link can be classified into this category.
3. Image Contents (IMC): this category contains pictures, photos, icons, etc.
4. Input Contents (INC): this category includes elements that can accept user input, such as: text box, radio button, pull-down menus, etc.

For a given block $B$, if $B$ is a leaf block, then it can be classified one of the four content categories. This is because a leaf block can only contain one type of content. If $B$ is not a leaf block, the leaf block set of $B$ is denoted $Leaf(B) = \{ b_1, b_2, ..., b_n \}$. First, the leaf blocks can be classified into the four categories according to their content types. Then four categories of leaf block sets can be obtained, denoted $TCLeaf(B)$, $LTCLeaf(B)$, $IMCLeaf(B)$, and $INCLef(B)$. Obviously, the four leaf block sets are the subsets of $Leaf(B)$. If one of the subsets is $\emptyset$, it means that $B$ does not contain the corresponding type of the contents. In order

![Block Illustration](image)

(a) block $A$  
(b) block $B$

Figure 6.3: Two blocks with different seam degree
to determine the rate of each content type, we need to calculate area of each leaf block. We use $\text{Area}(c_i)$ to represent the area of a leaf block $b_i$. If the content of $b_i$ is a text content or link text content, we consider the area of characters is more accuracy than the area of a block. For example, Figure 6.4 shows two leaf blocks which have the same area, but their quantities of text are different. In this case, the area of blocks cannot be equal to the quantity of text.

![Figure 6.4: Two blocks with different quantity of text](image)

If the content of block $b_i$ is a text content or link text content, we approximately calculate the area of text $\text{TextArea}(b_i)$ to replace the area of block $b_i$ as in equation (6-4):

$$\text{TextArea}(b_i) = \text{Character}(b_i) \times \text{FontSize}(b_i)^2$$  \hspace{1cm} (6-4)

where $\text{Character}(b_i)$ represents the number of single-byte characters in $b_i$, $\text{FontSize}(b_i)$ represents the font size of text in $b_i$.

For a given leaf block subset: $\text{TCLLeaf}(B)$, $\text{LTCLeaf}(B)$, $\text{IMCLeaf}(B)$ or $\text{INCLeaf}(B)$, according to the area (or text area) of its leaf blocks, the leaf blocks of the given subset can be sorted from large to small area. By utilizing the sorted leaf block subsets, four area vectors can be obtained, denoted $V_{tc}(B)$, $V_{ltc}(B)$, $V_{imc}(B)$ and $V_{inc}(B)$. The value of each element in the four vectors is the area (text area) of corresponding leaf blocks. We call these four vectors “content vectors”. Let us see an example. Figure 6.5 shows three blocks A, B and C, where the boxes on the right side of the blocks indicate the content type and area (or text area) of their leaf blocks. The unit of area (or text area) is px*px. The table shows the content vectors of the three blocks. From the table, it is obvious that content vectors can describe the intuitive visual feature with a formulated way. By comparing the content vectors, it is easy to tell the
difference between two blocks.

![Top References](image1)

![It's finals week: Do you know what your teen is taking to study so hard?](image2)

![First/Given Name: Last/Family Name: Organization: Country: Email:](image3)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{tc})</td>
<td>(2744)</td>
<td>(\emptyset)</td>
<td>(2448, 2448, 1872, 1152, 864)</td>
</tr>
<tr>
<td>(V_{ltc})</td>
<td>(2541, 2299, 2178, 1573, 968)</td>
<td>(8712)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(V_{imc})</td>
<td>(\emptyset)</td>
<td>(19530)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(V_{inc})</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(3750, 3750, 3750, 3750, 2500)</td>
</tr>
</tbody>
</table>

Figure 6.5: Content vectors of three blocks

6.4.2 The Content Similarity of Two Blocks

If the content vectors of two given blocks are determined, the similarity of each content vector \((V_{tc}, V_{ltc}, V_{imc} \text{ and } V_{inc})\) can be calculated. There are many algorithms to calculate the similarity of two vectors, of which the cosine similarity is a simple and efficient algorithm [26]. Here we take the vector of the text content as an example to explain the calculation of cosine similarity.
similarity. For two given blocks $B_1$ and $B_2$, their text content vectors are $V_{tc}(B_1) = (u_1, u_2, \ldots, u_m)$ and $V_{tc}(B_2) = (v_1, v_2, \ldots, v_n)$. Let us assume that $V_{tc}(B_1) \neq \emptyset$, $V_{tc}(B_2) \neq \emptyset$, and $n > m$. Because the cosine similarity requires that the two vectors must have the same number of elements, we need to add $(n-m)$ elements whose value are 0 into $V_{tc}(B_1)$, denoted $V'_{tc}(B_1) = (u_1, u_2, \ldots, u_m, u_{m+1}, \ldots, u_n)$. The cosine similarity of $V'_{tc}(B_1)$ and $V_{tc}(B_2)$ can be calculated as in equation (6-5):

$$\cos(V'_{tc}(B_1), V_{tc}(B_2)) = \frac{\sum_{i=1}^{m} u_i \times v_i}{\sqrt{\sum_{j=1}^{m} (u_j)^2} \times \sqrt{\sum_{i=1}^{n} (v_i)^2}}$$

If both $V'_{tc}(B_1)$ and $V_{tc}(B_2)$ are $\emptyset$, $\cos(V'_{tc}(B_1), V_{tc}(B_2))$ is ill-formed. In this case, we define the $\cos(V'_{tc}(B_1), V_{tc}(B_2))$ to be zero. Similarly, the cosine similarity of other content vectors (including $V_{ltc}$, $V_{imc}$ and $V_{inc}$) can also be determined.

Additionally, the four types of contents may have different weight in $B_1$ and $B_2$. Also, we take the text content as an example to explain the calculation of weight. For two given blocks $B_1$ and $B_2$, their text content vectors are $V_{tc}(B_1) = (u_1, u_2, \ldots, u_m)$ and $V_{tc}(B_2) = (v_1, v_2, \ldots, v_n)$. The weight of text content can be calculated as in equation (6-6):

$$\text{Weight}(TC(B_1, B_2)) = \frac{\sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j}{\text{Area}(B_1) + \text{Area}(B_2)}$$

where the $\text{Area}(B_i)$ represents the total area (and text area) of leaf blocks in block $B_i$. It means that the greater the area of the corresponding type of contents is, the higher its weight will be.

After the cosine similarity and weight of each content area vector are determined, the content similarity $CS(B_1, B_2)$ of $B_1$ and $B_2$ can be calculated as in equation (6-7):

$$CS(B_1, B_2) = W_{tc} \times \cos_{tc} + W_{ltc} \times \cos_{ltc} + W_{imc} \times \cos_{imc} + W_{inc} \times \cos_{inc}$$

where $W_{tc}$, $W_{ltc}$, $W_{imc}$, and $W_{inc}$ represent the weight of four types of contents ($TC$, $LTC$, $IMC$ and $INC$) respectively, and $\cos_{tc}$, $\cos_{ltc}$, $\cos_{imc}$, and $\cos_{inc}$ represent the cosine similarity of the corresponding content vector ($V_{ltc}$, $V_{ltc}$, $V_{imc}$ and $V_{inc}$) respectively. $CS(B_1, B_2)$ is
between 0 and 1.

6.4.3 The Average Content Similarity of Adjacent Child Blocks in a Block

If a block has child blocks, the average content similarity of adjacent child blocks can indicate the content coherent degree of the child blocks in the block. It should be noted that only the content similarity of adjacent child blocks is considered. For a given block $B$, the set of child blocks in $B$ is $\text{Child}(B) = \{b_1, b_2, \ldots, b_n\}$. If two child blocks are adjacent, we count 1 pair. Let us assume that there are $m$ pairs of adjacent child blocks. The average seam degree $\text{AvgCS}(B)$ can be calculated as in equation (6-8):

$$\text{AvgCS}(B) = \frac{\sum CS(b_i, b_j)}{m}$$

(6-8)

where $b_i$ and $b_j$ are adjacent child blocks. $\text{AvgCS}(B)$ is also between 0 and 1. If it is closer to 0, the content coherent degree of child blocks is lower. If it is closer to 1, the content coherent degree of child blocks is higher. Let us see an example to explain the average content similarity of adjacent child blocks in a block.

<table>
<thead>
<tr>
<th></th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{tc}$</td>
<td>(1584)</td>
<td>(15552, 14976, 3456, 2304)</td>
<td>(2592)</td>
</tr>
<tr>
<td>$V_{ite}$</td>
<td>(1872, 1728, 1440, 1296)</td>
<td>(6624, 5760)</td>
<td>(6336, 5760)</td>
</tr>
<tr>
<td>$V_{imc}$</td>
<td>(7056, 7056, 7056, 7056)</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$V_{inc}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td><strong>Total Area</strong></td>
<td>36414</td>
<td>48672</td>
<td>14688</td>
</tr>
</tbody>
</table>

Figure 6.6: Average content similarity of adjacent child blocks in a block
Figure 6.6 shows a block which contains three child blocks $b_1$, $b_2$, and $b_3$. The table shows the content vectors and total area of each child block. In this example, there are two pairs of adjacent child blocks: $(b_1, b_2)$, and $(b_2, b_3)$. According to equation (6-5), (6-6) and (6-7), we can calculate the content similarity between each pair of adjacent child blocks. Here the results are: $CS(b_1, b_2) = 0.4898$, and $CS(b_2, b_3) = 0.8204$. Since $b_1$ and $b_3$ are not adjacent child blocks, we do not calculate the content similarity between $b_1$ and $b_3$. According to equation (6-8), we can calculate the average content similarity of adjacent child blocks. Here, the result is $AvgCS(B) = (0.4898 + 0.8204) / 2 = 0.6551$.

### 6.5 Segment Web Pages Using Visual Semantics

**Table 6.1: Steps for judging a block whether should be divided**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>If the current block is a leaf block, then do not divide it. Otherwise go to Step 2.</td>
</tr>
<tr>
<td>Step 2</td>
<td>If the current block contains data record blocks, then divide it. Otherwise go to Step 3.</td>
</tr>
<tr>
<td>Step 3</td>
<td>If the current block is one of data record blocks, then do not divide it. Otherwise go to Step 4.</td>
</tr>
<tr>
<td>Step 4</td>
<td>If the current block contains only one child block, then divide it. Otherwise go to Step 5.</td>
</tr>
<tr>
<td>Step 5</td>
<td>If the $AvgSD(B)$ of the current block is less than $\alpha$, then divide it. Otherwise go to Step 6.</td>
</tr>
<tr>
<td>Step 6</td>
<td>If the $AvgCS(B)$ of the current block is less than $\beta$, then divide it. Otherwise go to Step 7.</td>
</tr>
<tr>
<td>Step 7</td>
<td>If the area of the current block is greater than the half of client area, then divide it. Otherwise go to Step 8.</td>
</tr>
<tr>
<td>Step 8</td>
<td>If the current block does not satisfy all of the above conditions, then do not divide it.</td>
</tr>
</tbody>
</table>

First we use the method that was introduced in Chapter 2 to transform a web page into a pruned DOM tree. As introduced in Section 6.1, data record blocks are special blocks. Since their child blocks are arranged neatly and two data record blocks have similar contents, thus seam degree and content similarity are not suitable for data record blocks. Therefore we need to recognize and mark the data records by using the proposed method in Chapter 5. If a web page does not contain any data record blocks, then no blocks will be marked.
Our web page segmentation algorithm is a top-down method. It begins from the root node of the DOM tree and which is set to be the current node. The corresponding block of the current node will be judged according to the steps shown in Table 6.1. If the current node should be divided, then its child blocks will be judged as well. If the current node should not be divided, then it will be pushed into an array of segments and its child blocks will not be judged anymore. Here, we introduce \( \alpha \) to be the threshold of \( \text{AvgSD} \), and \( \beta \) to be the threshold of \( \text{AvgCS} \). The detailed algorithm is shown in Figure 6.7.

---

Input: a DOM tree of a web page \( T \)
Output: an array of segments \( \text{SegmentSet} \)

**Begin**

1. \( \text{SegmentSet} = \emptyset \)
2. \( \text{CurrentNode} = T \rightarrow \text{Root} \)
3. \( \text{Segment(CurrentNode)} \) {
    
    4. \quad \text{if CurrentNode is NOT divisible then}
    5. \quad \quad \text{push CurrentNode into SegmentSet}
    6. \quad \text{end if}
    7. \quad \text{if CurrentNode is divisible then}
    8. \quad \quad \text{ChildList = CurrentNode} \rightarrow \text{Children}
    9. \quad \quad \text{for each Child}_i \in \text{ChildList}
    10. \quad \quad \quad \text{Segment(Child}_i)\)
    11. \quad \text{end for}
    12. \quad \text{end if}
    13. } \)
14. \quad \text{return SegmentSet}

**End**

---

**Figure 6.7: Algorithm for web page segmentation**

6.6 Experiment and Evaluation

6.6.1 Data Set

We submitted 10 queries to Google, from which we randomly collected 10 pages from the search results as test pages. As a result, 100 pages are collected as the experiment data.
6.6.2 Evaluation of Seam Degree and Content Similarity

We set the thresholds $\alpha$ and $\beta$ to be 0 to 1 respectively, where the step is 0.1, and obtained a set of 121 $\alpha$ and $\beta$ pairs \{(0, 0), (0, 0.1), \ldots, (1, 0.9), (1, 1)\}. Using the 121 threshold pairs, we use our method to segment each page 121 times. Each time we recorded the segment numbers of each page, denoted $n_i$. For a given web page, the set of segment number is \{\text{n}_1, \text{n}_2, \ldots, \text{n}_{121}\}. Since the segment number of each page is different, the set of segment numbers should be normalized as follows:

$$\text{Normalize}(n_i) = \frac{n_i}{\text{Max}\{n_1, n_2, \ldots, n_{121}\}}$$

where $\text{Max}\{n_1, n_2, \ldots, n_{121}\}$ represents the largest value of the set.

For a given threshold pair $(\alpha_i, \beta_j)$, We calculated the average values of the normalized segment number of the 100 pages, denoted \{\text{N}_1, \text{N}_2, \ldots, \text{N}_{121}\}.

Table 6.2 shows the average results of the 121 threshold pairs and Figure 6.8 shows the coordinate graph of these results. This is a three-dimensional coordinate where the two horizontal axes represent the two thresholds $\alpha$ and $\beta$, and the vertical axis is the average normalized segment number of 100 pages.

According to Figure 6.8, the following inferences can be drawn:

(1) According to section 6.5, the blocks whose average seam degrees are less than $\alpha$ will be divided. Thus, if $\alpha = 0$, the Step 5 in Table 6.1 will be invalid. In other words, only the average content similarity is effective to determine whether divide a block or not. In this case, $\beta$ and the normalized segment number are approximately proportional. We can infer that the averaging content similarities of all the blocks are approximate uniform distribution.

(2) According to section 6.5, if $\alpha = 1$, most of the blocks should be divided, and the normalized segment number should be close to 1 no matter how $\beta$ changes. However, that was not the case. $\beta$ and the normalized segment number are still approximately proportional. We can infer that the average seam degrees of most blocks are 1.

(3) If $\alpha$ is constant, the curve increases steeply along with $\beta$. Conversely, if $\beta$ is constant, the curve increases gradually along with $\alpha$. We can infer that the averaging content similarity plays a main role to determine whether a block should be divided or not, and the average seam degree plays a supplementary role.
Table 6.2: Average number of segments of 121 threshold pairs

<table>
<thead>
<tr>
<th>α</th>
<th>0.0</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>β</td>
<td>0</td>
<td>0.017</td>
<td>0.023</td>
<td>0.024</td>
<td>0.025</td>
<td>0.034</td>
<td>0.057</td>
<td>0.059</td>
<td>0.075</td>
<td>0.102</td>
<td>0.197</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.041</td>
<td>0.041</td>
<td>0.043</td>
<td>0.045</td>
<td>0.055</td>
<td>0.087</td>
<td>0.098</td>
<td>0.115</td>
<td>0.142</td>
<td>0.242</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>0.054</td>
<td>0.055</td>
<td>0.059</td>
<td>0.063</td>
<td>0.081</td>
<td>0.112</td>
<td>0.117</td>
<td>0.145</td>
<td>0.169</td>
<td>0.267</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.085</td>
<td>0.086</td>
<td>0.097</td>
<td>0.099</td>
<td>0.121</td>
<td>0.160</td>
<td>0.167</td>
<td>0.190</td>
<td>0.207</td>
<td>0.314</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.140</td>
<td>0.141</td>
<td>0.149</td>
<td>0.155</td>
<td>0.183</td>
<td>0.247</td>
<td>0.269</td>
<td>0.300</td>
<td>0.320</td>
<td>0.434</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.229</td>
<td>0.230</td>
<td>0.230</td>
<td>0.244</td>
<td>0.249</td>
<td>0.277</td>
<td>0.328</td>
<td>0.357</td>
<td>0.391</td>
<td>0.417</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>0.381</td>
<td>0.382</td>
<td>0.396</td>
<td>0.399</td>
<td>0.413</td>
<td>0.436</td>
<td>0.464</td>
<td>0.501</td>
<td>0.534</td>
<td>0.639</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>0.523</td>
<td>0.524</td>
<td>0.524</td>
<td>0.538</td>
<td>0.541</td>
<td>0.557</td>
<td>0.570</td>
<td>0.604</td>
<td>0.637</td>
<td>0.675</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>0.685</td>
<td>0.685</td>
<td>0.686</td>
<td>0.694</td>
<td>0.696</td>
<td>0.722</td>
<td>0.740</td>
<td>0.771</td>
<td>0.791</td>
<td>0.815</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>0.829</td>
<td>0.830</td>
<td>0.830</td>
<td>0.839</td>
<td>0.840</td>
<td>0.863</td>
<td>0.866</td>
<td>0.878</td>
<td>0.896</td>
<td>0.906</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.944</td>
<td>0.944</td>
<td>0.945</td>
<td>0.946</td>
<td>0.947</td>
<td>0.955</td>
<td>0.957</td>
<td>0.960</td>
<td>0.974</td>
<td>0.979</td>
</tr>
</tbody>
</table>

Figure 6.8: The coordinate graph of page segment results

6.6.3 Comparison with VIPS

Direct comparison of the proposed method with all of the related work describe in Section 6.2 is difficult, since the data set and evaluation program used are not open to the public. Therefore we implemented the VIPS [32] algorithm as the comparison baseline. VIPS is a
popular page segment method that is often taken as a comparison baseline by other work. We conducted a questionnaire experiment to compare the proposed method and VIPS. There are 14 participants who evaluated the segment results of 100 pages. We use 1~5 five criteria to evaluate the results where 5 is best and 1 is worst. The detailed descriptions of the five criteria are shown in Table 6.3. Here, if a block that should not be divided was divided or a block that should be divided was not divided, we call these segments “bad segments”.

Table 6.3: The five criteria of evaluation

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Categories</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Perfect</td>
<td>There are no bad segments.</td>
</tr>
<tr>
<td>4</td>
<td>Good</td>
<td>There are few bad segments, and these bad segments have little effect on segment results.</td>
</tr>
<tr>
<td>3</td>
<td>Fair</td>
<td>There are some bad segments, and these bad segments have an effect on segment results.</td>
</tr>
<tr>
<td>2</td>
<td>Bad</td>
<td>There are a lot bad segments, and the segment results are not acceptable.</td>
</tr>
<tr>
<td>1</td>
<td>Too Bad</td>
<td>Almost all the segments are bad segments.</td>
</tr>
</tbody>
</table>

Table 6.4 and Figure 6.9 show the results of questionnaire evaluation. Within the five criteria, we consider that 5 (perfect) is the most significant category. Let us see an example. Suppose there are two page segment approaches (A1 and A2), and they use the same data set (10 web pages). A1 can divide each page into ten segments, but every page has one bad segment. So, the average rate of bad segments is 10%. A2 can also divide each page into ten segments. Nine pages of them have no bad segments and the tenth page has ten bad segments. Obviously, the average rate of bad segments is also 10%. Based on the average rate of bad segments, A1 and A2 have the same performance. But in real applications, A2 may be the better choice. To make the rate of bad segments 0%, A1 has to be manually tuned for all the 10 pages, while A2 just needs to be manually tuned for only one page. In other words, though A1 and A2 have the same rate of bad segments, A1 cannot perfectly segment any pages while A2 can perfectly segment nine pages. In this case, we consider A2 is better than A1. The perfectly segmented pages of VIPS account for only 31% while the perfectly segmented pages of our method account for 70%. It indicates that our method needs less manual intervention in order to make 100% perfectly segmented pages. If we consider both the results of criteria 4 (good) and 5 (perfect) are acceptable results, then VIPS has only 56% acceptable results while our method has 88% acceptable results.
Table 6.4: The results of questionnaire evaluation

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Perfect</td>
<td>Good</td>
<td>Fair</td>
<td>Bad</td>
<td>Too Bad</td>
</tr>
<tr>
<td>VIPS</td>
<td>31%</td>
<td>25%</td>
<td>19%</td>
<td>16%</td>
<td>9%</td>
</tr>
<tr>
<td>Proposed Method</td>
<td>70%</td>
<td>18%</td>
<td>9%</td>
<td>2%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Figure 6.9: The graph of questionnaire results

6.7 Conclusion

As a solution of nested search results, in this chapter, we proposed a formulated page segmentation method using visual semantics. Page segmentation aims to divide a web page into visually and semantically segments that are not nested. In this way, each segment can be indexed directly without considering the case of nested blocks. As a result, the problem of nested search results can be avoided. However, early techniques of web page segmentation are mainly based on machine learning algorithms and rule-based heuristics, which cannot be used for large-scale page segmentation. In order to overcome these limitations, instead of
analyzing the visual cues of web pages, the proposed method utilized the following three measures to formulate the visual semantics: Seam Degree was used to describe how neatly the blocks are arranged; Content Similarity was used to describe the content distance between the blocks. The data record blocks (see Chapter 5) do not have the two visual semantics. The data record blocks in a web page always have the similar contents and their child blocks are not neatly arranged. The data record blocks can be considered as a special case. Because of this, we utilized layout tree to recognize these blocks in Chapter 5. Actually, layout tree is also a formulated measure to describe the visual semantic of layout feature. Based on these visual semantics, the proposed method judges the DOM tree nodes top-down. An experiment was conducted to determine the relationship between the two visual semantics (seam degree and content similarity) and the number of segment result. Compared with VIPS, the experiment results show that the proposed method can divide a web page into appropriate semantic segments with few bad segments. The perfectly segmented pages of the proposed method are also more than that of VIPS.
Chapter 7

Conclusion and Future Work

7.1 Conclusion

Existing search engine, such as Google, Bing and Yahoo, has become an increasingly important tool. Finding information has become an easy exercise by just typing a few key words into the search engine text-box. However, the information found by most search engines today is located on web pages. For finding the relevant part of a web page, user has to spend extra time in reading the irrelevant information. Therefore, a block-level search engine is necessary. A block-level web search can locate the relevant blocks of web pages, which can help people to find relevant information quickly and filters out irrelevant information. However, a major obstacle of realizing a block-level search engine is that blocks in a web page are not completely independent and they are nested. Building an index for the nested blocks may lead to nested search results which are redundant and difficult to understand. Therefore, the cornerstone of realizing a block-level web search engine is to build a proper index in order to avoid the nested results.

In this thesis, there are two solutions to build a proper index. One solution is to build an index for the blocks that do not include other blocks. In this thesis, these blocks are called leaf blocks. However, indexing the leaf blocks may ignore the structure of web pages and make the search results too scattered. Therefore, it is necessary to extract the proper blocks that people really require after the relevant leaf blocks are determined by analyzing the structure of web pages. This process is called block extraction. Another solution is to divide a web page into segments that are not nested and have independent semantics. This process is called page segmentation. Each segment can be regarded as an independent sub-page and they can be indexed directly. In this case, nested search results will not occur. Therefore, this thesis
provides two key contributions: block extraction and page segmentation. The contributions of this thesis are as follows:

1. In order to extract the required blocks, we analyzed the features of blocks in order to determine which feature is most effective to detect the required blocks. For a single block, we introduced three categories of features: text quantity, query term occurrence frequency, and DOM tree structure feature. For blocks in a parent-children group, we introduced the Gap feature to the relationship between the largest child block and the parent block. We manually labeled the required blocks from a set of web pages and utilized SVM to train these features. The results of the experiments show that the features of DOM tree structure, especially the depth of a block (including the depth from the root block to the block and the depth from the block to its deepest leaf block) are most effective to distinguish the required blocks. The other features can also play good supplement roles. Although, Gap feature is not the most effective feature, it can still improve the precision of a classifier for detecting the required blocks.

2. We proposed a method to automatically extract the required block from nested blocks in a web page. This method both focuses on the DOM tree depth and the total word weight of blocks. Based on the DOM tree depth and total word weight, each block will be scored. The block that has the highest score is extracted as a required block. The result of an evaluation experiment indicated that the proposed method is effective to extract the required block in most cases, and the result of a usability experiment indicated that extracting the required blocks by using the proposed method is useful to improve user’s effectiveness when a user is seeking the information from web pages.

3. We proposed a novel method to recognize data record blocks, which is an important step of web page segmentation. These data record blocks always have independent and complete semantics. Since these data record blocks of a page are generated from database, they often have similar layout. We used separators and leaf blocks to transform a block into a layout tree. By calculating the similarity of layout trees, we clustered the blocks that have similar layout. We used the LAMSB weight to measure these clusters. Finally, the cluster which has the highest LAMSB weight can be extracted as the data record block cluster. The experiment results indicated that (i) the optimal layout tree similarity threshold is 0.4; (ii) the proposed method is effective to recognize the data record blocks and has a good fault-tolerant and identification ability, although, its effectiveness may be different for different sites.

4. We proposed a formulated page segmentation method using visual semantics. Page segmentation aims to divide a web page into visually and semantically segments that are not nested. In this way, each segment can be indexed directly without considering the case of nested blocks. As a result, the problem of nested search results can be avoided. The proposed
method utilized the following two measures to formulate the visual semantics: Seam Degree was used to describe how neatly the blocks are arranged; Content Similarity was used to describe the distance between the blocks. The data record blocks (see Chapter 5) do not have the two visual semantics. The data record blocks in a web page always have the similar contents and their child blocks are not neatly arranged. The data record blocks can be considered as a special case. Because of this, we recognized and marked these blocks in advance. Based on the visual semantics, the proposed method judges the DOM tree nodes top-down. An experiment was conducted to determine the relationship between the two visual semantics (Seam Degree and Content Similarity) and the number of page segment. Compared with VIPS, the experiment results show that the proposed method can divide a web page into appropriate semantic segments with few bad segments. The perfectly segmented pages of the proposed method are also more than that of VIPS.

7.2 Future Work

For realizing a block-level, we considered two solutions to build proper index to avoid the nested search results. We need to determine the optimal one from the two solutions. The theoretical analysis and experiments indicated that both two solutions are effective to solve the nested search result problem. However, practical analysis is also necessary to detect the potential problems of solution. For each solution, we will classify these potential problems into two types: solvable problem and unsolvable problem. By analyzing the solvable and unsolvable problems, we can determine which one is the optimal solution.

Once the optimal solution is determined, we are planning to realize a real block-level search engine under the laboratory environment. A real world search engine mainly consists three parts: web crawling, indexing and searching. However, in the laboratory environment, we do not consider the web crawling. We can build a block-level web search engine on a particular type of web pages, such as: online shopping web pages, blog pages or news pages. If the block-level web search engine is practicable, we can build a block-level on large-scale of web pages. Our ultimate goal is to build a real world block-level search engine which may overthrow the concept of web search engines.
Bibliography


Proc. 9th ACM SIGKDD Int’l Conf. on Knowledge Discovery and Data Mining (KDD ’03), Washington, USA, pp. 296-305, Aug. 2003.


Wide Web (WWW '03), Budapest, Hungary, pp. 11-18, May 2003.


extraction tools,” *SIGMOD Record*, vol. 31, no. 2, pp. 84-93, Jun. 2002.


[38] K. Simon and G. Lausen, “ViPER: augmenting automatic information extraction with visual perceptions,” *Proc. 14th ACM Int’l Conf. on Information and Knowledge*


Advanced Applied Informatics (IIAI AAI ’13), Matsue, Japan, Sept. 2013. (in Press)