LTDE: A Layout Tree Based Approach for Deep Page Data Extraction

Jun ZENG\(^{[a]}\), Feng LI\(^{1}\), Brendan FLANAGAN\(\dagger\dagger\), Nonmembers, and Sachio HIROKAWA\(\dagger\dagger\dagger\), Member

SUMMARY  Content extraction from deep Web pages has received great attention in recent years. However, the increasingly complicated HTML structure of Web documents makes it more difficult to recognize the data records by only analyzing the HTML source code. In this paper, we propose a method named LTDE to extract data records from a deep Web page. Instead of analyzing the HTML source code, LTDE utilizes the visual features of data records in deep Web pages. A Web page is considered as a finite set of visual blocks. The data records are the visual blocks that have similar layout. We also propose a pattern recognizing method named layout tree to cluster the similar layout visual blocks. The weight of all clusters is calculated, and the visual blocks in the cluster that has the highest weight are chosen as the data records to be extracted. The experiment results show that LTDE has higher effectiveness and better robustness for Web data extraction compared to previous works.

key words: deep Web data extraction, layout tree, visual features, Web mining

1. Introduction

Deep Web is a large part of data available on the Web. Deep Web (also called hidden Web or invisible Web)\(\cite{1, 5}\) differs from the surface Web which can be indexed by standard search engines. The pages of deep Web are generated dynamically from a database and are difficult for standard crawler-based search engines to index. Thus, these types of Web pages are termed deep Web pages. Some search engines, such as Google, offer various deep Web sources, including images, maps, news, video, and so on. However, even a search engine as far reaching as Google provides access to only a very small part of the deep Web. Therefore, most of the time people cannot obtain the information they require from deep Web pages just typing some keywords into a Web search engine.

Deep Web pages always represent retrieved information in the form of data records. Data records generated from databases are useful in meta search engines. However, in order to be used by a meta search engine, data records need to be extracted from deep Web pages. This data extraction process is so called Information Extraction (IE). According to the IE task domain, the extraction task can be classified into 3 categories\(\cite{1}\): record-level, page-level, and site level. The record-level IE task can be further classified into two categories according to the input page: IE from a single page and IE from multiple pages. In this paper, we will study the problem of automatically extracting the data records from a single deep Web page.

The problem of deep Web data extraction has received great attention in recent years. Like hand-written Web pages, deep Web pages are also mainly written in HTML. Thus, most of the proposed data extraction solutions are based on analyzing the HTML source code of Web pages. However, these solutions have some limitations. First, with the reversion of HTML, some new elements will be introduced and some elements may be deprecated. Once the version of HTML changes, the previous works may not be able to adapt to the new version of the language. Second, Web pages are not written only in HTML, they also need the support of some scripting languages such as JavaScript and Cascading Style Sheets (CSS). Moreover, some web pages are dynamically generated by scripting languages while the browser is rendering the Web pages. Most previous works have not taken scripting languages into consideration.

In order to overcome these limitations, some researchers proposed vision-based methods that rely on visual cues from browser renderings. Most of the vision-based methods\(\cite{8, 12, 22}\) focus on the location, size or font features of elements in data records. For example, these works cluster the data records through analyzing the similarity of position, image size and font size of the elements or consider that the main contents or data records are always in the middle of a Web page. Even though such assumptions are important for the success of the algorithm, it is hard to see how the proposed approaches can be used for pages with other semantic structures. In this paper, besides HTML structure and visual cues, we also consider the relative positions of HTML elements as an important feature to extract data records. Relative position is different from other visual properties. Visual properties, such as: position, size and font size only refer to just only one element, but relative position refers to at least two elements. In other words, visual properties like position, size and font size are absolute and a single tuple, but the relative position is relative and therefore a double tuple. The relative position of two elements \(x_1\) 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 \(x_2\). Obviously, given two relative position \(RP_1(x_1, x_2) = (x_1, r_1, x_2)\) and

Manuscript received September 5, 2016.
Manuscript revised January 9, 2017.
Manuscript publicized February 21, 2017.

\(^1\)The authors are with the School of Software Engineering, Chongqing University, Chongqing, 400044 China.
\(^1\dagger\dagger\)The author is with the Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka-shi, 819–0395 Japan.
\(^1\dagger\dagger\dagger\)The author is with Research Institute for Information Technology, Kyushu University, Fukuoka-shi, 819–0395 Japan.

\(\dagger\)E-mail: zengjun@cqu.edu.cn
DOI: 10.1587/transinf.2016EDP7375
We can also detect the data records from multiple pages. We level extraction task from a single input page. (STEM [11] as MDR [15], TPC [13], STEM [11], RST [10], ViNTs [19], fully automatic methods have been proposed. Methods such and time-consuming. To overcome the above limitations, semi-automatic These works are di

difficult to maintain and have low efficiency. In order to tackle these problems, semi-automatic approaches [14], [16], [20], [24], [25] 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 [15], TPC [13], STEM [11], RST [10], ViNTs [19], ViPER [22], and ViDE [12] were designed to tackle record-level extraction task from a single input page. (STEM [11] can also detect the data records from multiple pages.) We

$$RP_2(y_1, y_2) = (y_1, r_2, y_2),$$ if and only if 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 much stricter to describe the similarity of any two HTML elements.

If there are three elements $x_1$, $x_2$ and $x_3$, we can first consider $x_2$ and $x_3$ as one node $x_2, x_3$ and get the relative position $RP_1(x_1, x_2, x_3)$. And then we can get the relative position $RP_2(x_2, x_3)$. Thus, a tree is formed as shown in Fig. 1. 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 Sect. 4.

In this paper, we propose a method named Layout Tree based Data Extraction (LTDE) for extracting data records from the deep Web pages. This method analyzes the layout feature of data records, and transforms the layout into a binary tree called layout tree. We calculate the distance of the layout trees of visual blocks (see Sect. 3.1 for the definition of visual block), and cluster the visual blocks with similar layout trees together. Finally the data records will be extracted from the clusters using visual features.

The rest of the paper is organized as follows: Related works are reviewed in Sect. 2. Visual block and visual features of deep Web data records are introduced in Sect. 3. The layout tree of visual blocks is introduced in Sect. 4. Our solution for data record extraction is described in Sect. 5. Experimental results are reported in Sect. 6. Finally, the conclusion and future work are given in Sect. 7.

2. Related Work

Data extraction from Web pages (including deep Web pages) is attracting major research interest. Good surveys of works on Web information extraction can be found in [1] and [2]. Early techniques used were manual approaches, such as in Minerva [27], TSIMMIS [21], and Web-OQL [18]. These works are difficult to maintain and have low efficiency. In order to tackle these problems, semi-automatic approaches [14], [16], [20], [24], [25] 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 [15], TPC [13], STEM [11], RST [10], ViNTs [19], ViPER [22], and ViDE [12] were designed to tackle record-level extraction task from a single input page. (STEM [11] can also detect the data records from multiple pages.) We

roughly divide these approaches into two groups: HTML-based approaches [5], [7], [10], [11], [13], [15] and vision-based approaches [3], [4], [8], [12], [19], [22]. Our method named LTDE is a vision-based method, thus, more detailed about vision-based approaches will be discussed in this section.

HTML-based approaches transform 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. Vision-based approaches consider that when a page is presented to the user, the spatial and visual features play a very important role because they help the user to unconsciously divide the webpage into several semantic parts. Therefore, many vision-based approaches have been proposed. These approaches rely on visual cues from browser renderings. ViNTs [19] 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 [22] also incorporates visual information and tag structure on a Web page for data record extraction. In both of the approaches, tag structures are still the primary information utilized, while visual information plays a supplementary role. ViDE [12] can be considered as a pure visual feature based approach. It is effective in extracting records from pages with well-organized visual features. Paper [8] proposed a method to extract the content structure for a web page based on ViPS (Vision-based Page Segmentation). ViPS can divide a web page into blocks by using the heuristics of visual-cue. Paper [3] also extracts the data records by using the visual feature, such as: position, layout, appearance and contents. Paper [4] proposed an approach that is based on structural regularity, visual and content similarity between data records displayed on the query result page. However, the existing vision-based methods are often based on simple models or heuristics that cannot be generalized.

The proposed method named LTDE is also a vision-based method. LTDE not only makes use of the visual information of HTML elements, but also considers the layout of these elements. The proposed method can be combined with other data item extraction methods to realize the entire data extraction process.

3. Visual Block and Visual Features of Data Records

3.1 Visual Block and Visual Block Tree

Paper [17] considers a Web page to be made up of a finite number of blocks. Each block can be recursively view as a sub-Web-page with sub-structure induced from the whole page structure. The blocks also have many other names: paper [9], [28] refers to blocks of Web pages as Web page components; paper [26] refers to blocks as co-herent regions; paper [27] refers to blocks as visual blocks. Similar to pa-
per [27], we also refer to these blocks as visual blocks or blocks for short. In addition, some works (e.g. paper [17]), considers that all these blocks are not overlapped. In our paper, we consider that the blocks are nested.

The detailed definition of a visual block is as follows:

**Definition 3-1**: Visual block $VB = (E, R)$, where $E$ is an Element object that is defined by the HTML DOM based on W3C standard, and $R$ represents the visible rectangular region where $VB$ appears in Web page.

According to W3C standard, a web page can be transformed into a DOM tree, and each DOM object has a corresponding element in the web page. If an element is visible, it will be displayed within a rectangular region in a web page. Therefore, a DOM object and its rectangular region (if it is visible) represent a block in a web page. Moreover, we define the child block and leaf block as follows:

**Definition 3-2**: For two given visual blocks $VB_1 = (E_1, R_1)$ and $VB_2 = (E_2, R_2)$, if $E_1$ is a descendant of $E_2$, then $VB_2$ includes $VB_1$, denoted $VB_1 \subseteq VB_2$.

**Definition 3-3**: If a visual block $VB = (E, R)$ does not include any other visual blocks, then $VB$ is a leaf visual block, denoted $VB$ : leaf. Otherwise $VB$ is not a leaf visual block, denoted $VB$ : non-leaf.

According to the inclusion relation of visual blocks, a Web page can be transformed into a Visual Block Tree. Figure 2 (b) shows the visual block tree of a Web page shown in Fig. 2 (a). The visual block tree is an ordered tree. The blocks with the same parent have the same order as the corresponding HTML elements. Contrary to DOM tree, visual block tree removes the HTML elements that are not visible. Thus the visual block tree can be viewed as a pruned tree of DOM tree.

### 3.2 Visual Feature of Data Records

The visual block and visual block tree are common features that can be applied to any kind of Web pages. In this section, the features of deep Web pages are introduced. Deep Web pages often represent retrieved information in the form of data records. Figure 3 shows two examples of deep Web page from “Amazon” and “Rakuten”. The rectangles indicate the data records of the two deep Web pages. Obviously, each data record has a corresponding visual block, thus a data record can be represented as a visual block. (In rare cases, a data record is made up of several sibling blocks. In Sect. 5.3, these cases will be discussed.) Through the observation of the visual information of deep Web pages, the following features of data records were discovered.

1. **Similar Layout (SL) feature**

   Here only the leaf blocks are considered. This is because the leaf blocks contain contents such as text, images etc. The other intermediate visual blocks are only the structure of the visual blocks tree, and they do not contain content. On the other hand, if the intermediate visual blocks are considered, the visual blocks may overlap each other.

   Figure 4 shows two data records of tablet computers. Although the contents of two records are not all the same, the main layout is similar. In both records, the picture is on the top of records; the product names are under the pictures; the prices are under the product names; evaluations are at the bottom. Record (a) contains some additional contents, but the layout of “picture”, “name”, “price” and “evaluation” is the same in both (a) and (b). Therefore, this feature is the essence of our research to identify the data records from other visual blocks.

   (2) **In a deep Web page, the visual blocks of data records have similar shapes and coordinates.** If two visual blocks have similar width or height, we consider that they have the similar shapes. If two visual blocks have similar horizontal or vertical coordinates, we consider that they...
have the similar coordinates. We call this feature Similar Shape and Coordinate (SSC) feature for short.

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

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 visual blocks.

4. Layout Tree of Visual Blocks

4.1 The Layout of a Visual Block

In Sect. 3.1, the similar layout feature has been mentioned as a key feature for distinguishing data records from other visual blocks. In this section, the description of layout and the creation of layout tree will be introduced.

The layout of a visual block can be described as follows: Given a visual block \( B \), where \( B \) is not a leaf block, the layout of \( B \) can be represented as two-tuples \( \text{Layout}(B) = (LB, SL) \). \( LB = (b_1, b_2, \ldots, b_n) \) is a sequence of leaf blocks that are included by \( B \). \( SL = (S_1, S_2, \ldots, S_{n-1}) \) is a sequence of split lines, including horizontal split lines and vertical split lines. The direction of a split line is a simple and effective way to describe the relative position of leaf blocks in a data record.

Figure 5 (a) shows an example of the layout of a visual block, and Fig. 5 (b) shows the visual block tree of Fig. 5 (a). In Fig. 5 (a), the rectangles \( (b_1, b_2, b_3, b_4) \) represent the leaf blocks and dotted lines \( (S_1, S_2, S_3) \) represent the split lines. In Fig. 5 (b), the nodes with dotted line represent the intermediate nodes of the visual block tree. All the intermediate nodes \( \{n_1, \ldots, n_5\} \) are ignored, because if they are considered, the visual blocks may overlap each other, which will make it difficult to determine the split lines. Therefore only the leaf blocks are considered to describe the layout of a visual block. As mentioned in Sect. 3.1, the leaf blocks are ordered, thus the order of leaf blocks need to be determined. Because the visual block tree is ordered, the order of leaf blocks will be determined by depth-first traversal of the visual block tree. After removing the root and intermediate nodes, the sequence of leaf blocks can be obtained.

The importance of different leaf blocks for the layout is different. For example in Fig. 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 refer this importance as “weight”. Given a visual block \( B \), and \( b_i \) is a leaf block of \( B \). The weight of \( b_i \) is calculated as in formula (1):

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

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

4.2 Determination of Split Lines

The determination of split lines is more complex. When determining split lines, two conditions must be followed:

**Condition 4-1**: A split line never crosses any blocks;

**Condition 4-2**: There must be blocks on both sides of every split line;

Based on the conditions, here is an example to introduce the algorithm for determining split lines.

Figure 6 shows how to determine the split lines of the visual block shown in Fig. 5 (a). As shown in Fig. 6 (a), the four edges of the first leaf block \( b_1 \) are extended. The four extended edges are the candidate split lines. According to condition 4-2, \( S_{1,1} \), \( S_{1,2} \) 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 side of \( S_{1,2} \), \( b_2 \), \( b_3 \) and \( b_4 \) are on the lower side of \( S_{1,2} \)). Thus \( S_{1,2} \) is the first split line, and will be saved into the sequence of split lines. In Fig. 6 (b), the split line \( S_{1,2} \) divides
the visual block into two parts $P_1$ and $P_2$. Because $P_1$ only contains one leaf block, there cannot be any split lines in $P_1$. Hence, we consider the leaf blocks in $P_2$. In $P_2$, $b_2$ is the first leaf block. Similarly, the four edges of $b_2$ are extended. According to condition 4-2, $S_{2,3}$ and $S_{3,4}$ cannot be a split line. Therefore, the layout of a visual block 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 the split line is horizontal, the upper part is the left subtree and lower part is the right subtree. If the split line is vertical, the left part is the right subtree and right part is the right subtree. Therefore, the layout of a visual block can be regarded as a tree. We refer the tree as “layout tree”. In this section, the layout tree will be introduced in detail.

We take the visual block shown in Fig. 5 (a) as an example to introduce the process to generating a layout tree as shown in Fig. 8. Let us suppose that the sequence of the split lines ($S_1$, $S_2$, $S_3$) has been determined. In Fig. 8, each split line can split current visual block into two smaller parts. The split lines can be considered as a root node, and the two smaller parts can be considered as the left subtree and the right subtree. If the smaller part contains only one leaf block, it will be further split. Thus, the split line and leaf blocks can form a tree. In this paper, this tree is named layout tree. Figure 8 (d) shows the final layout tree of the given visual block. By observing the structure of the layout tree, we discover that a layout tree has the following features:

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

**Feature 2:** In a layout tree, the root node and intermediate nodes represent the split lines, and the leaf nodes represent the leaf blocks.

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

Feature 3 is the most important feature. It gives layout tree the ability to accurately describe the layout of leaf blocks in a visual block. According to the SL feature (see Sect. 3.2), we can calculate the distance of two layout trees in order to compare the similarity of the layouts of two visual blocks. The smaller the distance between two layout trees, the higher the similarity of two blocks will be. Figure 9 shows the core concept for calculating the similarity of two visual blocks. The edit distance between trees, of which the Tree Edit Distance (TED) is, the higher the similarity of two blocks will be. The smaller the distance between two layout trees, the higher the similarity of two blocks will be.

4.4 The Distance between Two Layout Trees

Due to paucity of space, we only introduce the TED algorithm roughly. The detailed algorithm can be found in paper [23]. The edit distance, denoted \( \delta(F, G) \), between two trees \( F \) and \( G \) 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 \( n_1 \) is an actual node or an empty node denoted by \( \epsilon \). The operation is a node replacement if \( n_1 \neq \epsilon \) and \( n_2 \neq \epsilon \), a node deletion if \( n_2 = \epsilon \), and a node insertion if \( n_1 = \epsilon \). Given a 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 (3):

\[
\delta(\phi, \phi) = 0
\]

\[
\delta(F, \phi) = \delta(F - v, \phi) + \gamma(v \rightarrow \epsilon)
\]

\[
\delta(\phi, G) = \delta(\phi, G - w) + \gamma(\epsilon \rightarrow w)
\]

if \( F \) is not a tree or \( G \) is not a tree:

\[
\delta(F, G) = \min \left\{ \delta(F - v, G) + \gamma(v \rightarrow \epsilon), \delta(F, G - w) + \gamma(\epsilon \rightarrow w), \delta(F_v, G_w) + \delta(F - F_v, G - G_w) \right\}
\]

if \( F \) is a tree and \( G \) is a tree:

\[
\delta(F, G) = \min \left\{ \delta(F - v, G) + \gamma(v \rightarrow \epsilon), \delta(F, G - w) + \gamma(\epsilon \rightarrow w), \delta(F - v, G - w) + \gamma(v \rightarrow w) \right\}
\]

where \( v \) and \( w \) are either both the left most or right most root nodes of the respective tree. \( F_v \) 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 tree obtained by deleting \( w \) from \( G \).

According to the TED algorithm and the three features of a layout tree, we introduce the cost functions to calculate the cost of operations. Formula (4) and formula (5) show the cost functions of insertion and deletion operations:

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

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

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

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

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: split line nodes and leaf block nodes. Moreover, there are two directions of
split line 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 different:

**Rule 4-1:** If node \( n_1 \) and node \( n_2 \) are different types (one is a split line node and the other one is a leaf block node), then \( n_1 \ text{diff} n_2 \).

**Rule 4-2:** If both node \( n_1 \) and \( n_2 \) are split line nodes, and the directions of \( n_1 \) and \( n_2 \) are different (one is horizontal and the other one is vertical) then \( n_1 \ text{diff} n_2 \). Otherwise \( n_1 \ sim n_2 \).

**Rule 4-3:** 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 \( n_1 \ text{diff} n_2 \).

**Rule 4-4:** If both node \( n_1 \) and \( n_2 \) are image nodes, then \( n_1 \ sim n_2 \).

**Rule 4-5:** 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 \( n_1 \ sim n_2 \). Otherwise \( n_1 \ text{diff} n_2 \).

After the edit distance of two layout trees are figured out, the distance 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 distance of \( T_1 \) and \( T_2 \) can be calculated as in formula (7):

\[
\text{Dist}(T_1, T_2) = \frac{\delta(T_1, T_2)}{\max\{\sum \text{Weight}(n_i), \sum \text{Weight}(m_i)\}}
\]

where \( n_i \) is a node in \( T_1 \) and \( m_i \) is a node in \( T_2 \). The denominator of formula (7) represents the greater one of total weight of the layout trees \( T_1 \) and \( T_2 \). The distance of \( T_1 \) and \( T_2 \) has the following features:

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

We introduce a distance threshold \( \alpha \). If \( \text{Dist}(T_1, T_2) \leq \alpha \), then \( T_1 \) and \( T_2 \) are similar, otherwise they are different.

Given two visual blocks \( VB_1 \) and \( VB_2 \), their layout trees are \( T_1 \) and \( T_2 \). The distance of the layout of \( VB_1 \) and \( VB_2 \) can be calculated as in formula (8):

\[
\text{LayoutDist}(VB_1, VB_2) = \text{Dist}(T_1, T_2)
\]

## 5. Data Record Extraction from Deep Web Page

In this section, we introduce the data record extraction method using the layout tree and other visual features mentioned in the previous section. Our method can be roughly divided into four steps:

1. **Step 1:** Generate the visual block tree of an input Web page;
2. **Step 2:** Cluster the similar layout visual blocks;
3. **Step 3:** Refine the clusters of Step 2;
4. **Step 4:** Identify the data records from the visual block clusters.

### 5.1 Generation of the Visual Block Tree of a Web Page

According to **Definition 3-1**, each visual block has a corresponding DOM Element object. Thus, we first obtain the DOM tree of the web page.

In the DOM tree, the nodes can be roughly classified into two categories: visible nodes and invisible nodes. Since the visual block tree only contain the visible nodes, thus we need to prune the DOM tree. The detailed pruning rules are shown in Table 1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1</td>
<td>The Attribute Nodes, Comment Nodes, and Document Nodes should be cut.</td>
</tr>
<tr>
<td>5-2</td>
<td>The invisible nodes should be cut, including the nodes such as (&lt;\text{head}&gt;), (&lt;\text{script}&gt;), (&lt;\text{meta}&gt;), etc., and the nodes whose \text{Width} and \text{Height} properties are zero and \text{Display} property is “none”</td>
</tr>
<tr>
<td>5-3</td>
<td>If a node has only one Text Node, then the Text Node should be cut.</td>
</tr>
<tr>
<td>5-4</td>
<td>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.</td>
</tr>
</tbody>
</table>

Fig. 10 Clustering the similar layout blocks of the visual block tree

### 5.2 Clustering of Similar Layout Visual Blocks

Block clustering is the most important part of data extraction, which can influence the precision of the result. In this section, we will introduce the detailed algorithm to cluster the similar layout blocks. In Fig. 10, the polygons with the same shape present the similar layout blocks. The computation of layout tree distance 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 has introduced in Sect. 3.2. This SSC feature has two aspects: similar shape and similar coordinate. Given two visual block \( VB_1 \) and \( VB_2 \), the \( \text{SSC}(VB_1, VB_2) \) can be calculated as in formula (9):

\[
\text{SSC}(VB_1, VB_2) = \text{ShapeSim}(VB_1, VB_2) \times \text{CoorSim}(VB_1, VB_2)
\]

where \( \text{ShapeSim}(VB_1, VB_2) \) represents the similarity of the shape, and \( \text{CoorSim}(VB_1, VB_2) \) represents the similarity of the coordinate. As introduced in Sect. 3.2, if two visual blocks have similar height or width, we consider them to have a similar shape. Similarly, if two visual blocks have similar horizontal or vertical coordinate, we consider that...
they have similar coordinate. The \( \text{ShapeSim}(VB_1, VB_2) \) and \( \text{CoorSim}(VB_1, VB_2) \) can be calculated as in formula (10) and (11):

\[
\text{ShapeSim}(VB_1, VB_2) = 1 - \min \left\{ \frac{|Rh(VB_1) - Rh(VB_2)|}{\max(Rh(VB_1), Rh(VB_2))}, \frac{|Rw(VB_1) - Rw(VB_2)|}{\max(Rw(VB_1), Rw(VB_2))} \right\}
\]

(10)

\[
\text{CoorSim}(VB_1, VB_2) = 1 - \min \left\{ \frac{|Rx(VB_1) - Rx(VB_2)|}{\max(Rx(VB_1), Rx(VB_2))}, \frac{|Ry(VB_1) - Ry(VB_2)|}{\max(Ry(VB_1), Ry(VB_2))} \right\}
\]

(11)

where \( Rh(VBi) \) is the height of \( VBi \), \( Rw(VBi) \) is the width of \( VBi \), \( Rx(VBi) \) is the horizontal coordinate of \( VBi \), and \( Ry(VBi) \) is the vertical coordinate of \( VBi \). It should be noted that \( \text{CoorSim}(VB_1, VB_2) \) also uses the \( Rh(VBi) \) and \( Rw(VBi) \) as denominators to calculate the similarity of coordinates. Thus the \( \text{ShapeSim}(VB_1, VB_2) \) and \( \text{CoorSim}(VB_1, VB_2) \) can be normalized in a same range. Empirically, if \( \text{SSC}(VB_1, VB_2) > 0.9 \), we consider \( VB_1 \) and \( VB_2 \) have similar shape and coordinate.

Figure 11 shows the detailed algorithm for clustering visual blocks with similar layout. In line 13, \( \text{LayoutDist}(VB_1, VB_2) \) has been defined in formula (8), which can calculate the distance of two layout trees of two visual blocks. The clusters of similar layout visual blocks can be obtained.

5.3 Refine the Clusters of Similar Layout Blocks

After the clusters are obtained, we need to refine them. Generally, each data record has a corresponding visual block. In other words, for a data record there is a visual block that contains all the contents of this data record. However, in rare cases, a data record may need two or more blocks to cover its contents. In other words, a data record consists of several sibling blocks. Some vision-based methods (e.g. ViDE [12]) first detects the data record region, and then regroups these sibling blocks in the region. In other words, ViDE considers all the regrouped sibling blocks as data records. In this case, if the data record region is wrong, the regrouped blocks will become meaningless. In our paper, we use a virtual block to describe a data record. A virtual block is not a real visual block. It only plays the role of combining different blocks together. Different from ViDE, we don’t consider the virtual block as data records. According to Sect. 5.4, we detect the data record region from the block clusters. If these virtual blocks do not meet the conditions of Sect. 5.4, they cannot be data records.

Figure 12 (a) shows an example of this case. A record is made up of \( a_i \) and \( b_i \), where \( a_i \) is required and \( b_i \) is optional. Here we only consider the case that at least one block is required in a data record. According to the algorithm introduced in Sect. 5.1, \( a_i \) and \( b_i \) are grouped into different clusters. We introduce a parameter called \( \text{Rect}(C) \), which is a minimum rectangle that can cover all blocks in cluster \( C \). We analyze the \( \text{Rect}(C) \) of each cluster. If the \( \text{Rect}(\text{Cluster}_i) \) and \( \text{Rect}(\text{Cluster}_j) \) are overlapping, we consider the two clusters as candidates, denoted \( \text{Cluster}_i \) and \( \text{Cluster}_j \). Then all the blocks in the two clusters are sorted according to the horizontal and vertical coordinate to determine the arrangement of these blocks. We check each block
top-down (or left-right). For example, in Fig. 12 (a), first the
a1 is checked and a pointer is generated to point to a1. If the
current block is from the same cluster of the block that the
pointer is pointing to, the pointer moves to the current block.
Because a1 is a block of Clusteri and a1 is the first block, a
virtual block c1 is created and a1 is set as a child node of c1.
Then b1 is checked. Because b1 is from cluster Clusterj that
is different from a1, b1 is set as a child node of c1 and the
pointer does not moved. The next block a2 is from the same
cluster as a1, so the pointer moves to a2 and another virtual
block c2 is created. Similarly, b2 is set as a child node of
c2 and pointer does not move. Next, the pointer moves to
a3 because it is from the same cluster as a2, and a3 is set as
a child node of c3. After all blocks are checked, c1, c2 and
c3 are set as child nodes of the parent node of a1 and b1, as
shown in Fig. 12 (b). Finally, c1 are grouped into a cluster
and Clusteri, and Clusterj will be deleted.

After all the clusters are regrouped, we refine the clus-
ters as the following steps:

Step 1: Given a cluster C = {VB1, ... , VBn}, where VB is
not a virtual block. If ∃ VB ∈ C and VB; leaf, then delete C.

A data record consists of more than one HTML ele-
ments, such as picture, product name, price etc. Thus, a leaf
block cannot be a data record. Therefore, the clusters that
consist of only leaf blocks are meaningless and need to be
removed.

Step 2: Given two clusters C1 = {VB1, ... , VBn} and
C2 = {VB1, ... , VBn}, where VB1 and VB2 are not vir-
tual blocks. If ∃ VB1 ∈ C1, VB2 ∈ C2, and VB1 < VB2, then
we remove VB2 from C2.

Step 3: Given two clusters C1 = {VB1, ... , VBn} and
C2 = {VB1, ... , VBn}, where VB1 and VB2 are not vir-
tual blocks. If ∃ VB1 ∈ C1, VB2 ∈ C2, and
LayoutDist(VB1, VB2) < α, then we merge C1 and C2.

In Sect. 5.2, only the similar layout blocks in the same
depth are clustered together. In Step 3, we can merge two
clusters which consist of similar visual blocks, even if they
are in different depths.

5.4 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 introduced in Sect. 3.1. The block
cluster that has the largest area and contains the most similar
layout blocks can be regarded as having the LAMSB fea-
ture. Given a set of block clusters S = {C1, ... , Cn}, we use
AreaWeight(C1) to measure the area weight of Ci and use
NumWeight(C1) to measure the number weight of Ci. We
calculate the two weight measures as in formula (12) and
(13):

\[
\text{AreaWeight}(C_i) = \frac{\text{Area}(C_i)}{\text{TotalArea}(S)} \quad (12)
\]
\[
\text{NumWeight}(C_i) = \frac{\text{Num}(C_i)}{\text{TotalNum}(S)} \quad (13)
\]

where Area(Ci) is the total area of blocks in cluster Ci,
TotalArea(S) is the total area of all the blocks in S. It should
be noted that if the blocks in Ci are virtual blocks then
Area(Ci) is the total area of the virtual blocks’ child nodes.
TotalNum(S) is the total number of blocks (including visual
blocks and virtual blocks) of all the clusters in S. Num(Ci)
is the total number of blocks in cluster Ci. Here, we intro-
duce a harmonic function Weight(Cj) of AreaWeight(Ci) and
NumWeight(Ci). We call this harmonic function LAMSB
weight. It can be calculated as in formula (14):

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

(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
records.

6. Experiment and Evaluation

In this section, experiments are conducted to evaluate the
effectiveness of LTDE. The contents of the experiments are
as follows:

(1) Determining the optimal distance threshold α that is
introduced in Sect. 4.4;
(2) Evaluating the effectiveness of the SSC feature and the
LAMSB feature that are introduced Sect. 3.2;
(3) Comparing our method LTDE with existing methods.

6.1 Data Set

In the experiment, two testbeds were used. The first testbed
was proposed by Yamada et al. [29] as the data set, de-
noted DataSet1. This data set includes frozen results pages
from 51 engines selected randomly from 114,540 pages with
search forms and manually identified target information to
be extracted. There are 253 deep Web pages in this data set.
The pages from 51 different engines have different visual
and layout features. The diversity of the data set can test
the robustness of LTDE. DataSet1 only contains the HTML
files of the Web pages. All the pages in DataSet1 can be
correctly displayed by the Web browser, although the CSS
and Javascript files have been removed.

We also manually collected 5000 pages from 50 dif-
f erent web sites as the second data set, denoted DataSet2.
The 50 web sites can be roughly divided into four types: on-
line shopping sites, news sites, video sites, SNS and blog
sites. Every site supports search function, and can generate
search forms and manually identified target information to
be extracted. There are 253 deep Web pages in this data set.
The pages from 51 different engines have different visual
and layout features. The diversity of the data set can test
the robustness of LTDE. DataSet2 contains all the CSS and Javascript files of pages. Moreover, many
pages of DataSet2 are dynamically generated by Javascript
after the pages are rendered by the Web browser. Therefore,
DataSet2 can determine the robustness of LTDE for dynamical pages.

6.2 Performance Measures

Almost all previous works use precision and recall to evaluate their experiment results [30]. In our experiment, we adopted precision and recall as the evaluation criteria. The data set is a finite set \( D = \{ P_1, \ldots, P_n \} \), where \( P_i \) is a Web page. \( \text{Record}(P_i) \) is a set of data records in a Web page \( P_i \); \( \text{Extract}(P_i) \) is a set of output data records that is extracted from page \( P_i \) by an IE method. The \( \text{Precision}(D) \), \( \text{Recall}(D) \) can be calculated as in formula (15) and (16):

\[
\text{Precision}(D) = \frac{1}{n} \sum \left| \frac{\text{Record}(P_i) \cap \text{Extract}(P_i)}{|\text{Extract}(P_i)|} \right|
\]

(15)

\[
\text{Recall}(D) = \frac{1}{n} \sum \left| \frac{\text{Record}(P_i) \cap \text{Extract}(P_i)}{|\text{Record}(P_i)|} \right|
\]

(16)

We also use F-measure as an evaluation criterion which is the weighted harmonic mean of precision and recall. F-measure of data set \( D \) can be calculated as in formula (17):

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

(17)

6.3 Evaluation of the Distance Threshold

In this experiment, we aim to determine the optimal distance threshold of layout trees. We set the distance threshold \( \alpha \) to be \( \{0, 0.1, 0.2, \ldots, 0.9, 1.0\} \), and extract the data records from the pages using proposed method. Figure 13 shows the \( \text{Precision}(D) \), \( \text{Recall}(D) \) and \( F(D) \) with different thresholds. In Fig. 13, the horizontal axis is the value of threshold \( \alpha \), and the vertical axis is the scores of precision, recall and F-measure. From Fig. 13, we can make the following observations.

Because the curves changed along with the change of \( \alpha \), it’s obvious that the distance threshold \( \alpha \) can influence the precision and recall of the proposed method. It indirectly proves that layout tree is effective to extract the data records with similar layouts. In other words, if the layout tree method is completely ineffective, the curves would be horizontal straight lines, no matter how \( \alpha \) changes.

Moreover, both the F-measures of DataSet1 and DataSet2 reach the maximum when \( \alpha \) is 0.4. Therefore, \( \alpha = 0.4 \) is an important point for both \( \text{Precision}(D) \) and \( \text{Recall}(D) \). In addition, both \( \text{Precision}(D) \) and \( \text{Recall}(D) \) of two data sets have the similar curve shapes. When \( \alpha \) is in the interval \([0, 0.4]\), both \( \text{Recall}(D) \) of two data sets increase steeply. When \( \alpha \) is in the interval \([0.4, 1]\), both \( \text{Recall}(D) \) of two data sets changes gradually. This means that when \( \alpha \) is less than 0.4, the ability of the layout tree to recognize the similar layout blocks significantly improves while \( \alpha \) increases, and when \( \alpha \) is greater than 0.4, the ability of layout tree to recognize layout similar blocks improves gradually. Meanwhile, \( \text{Precision}(D) \) of DataSet2 is increasing when \( \alpha \) is in the interval \([0, 0.4]\), and it is decreasing steeply when \( \alpha \) is in the interval \([0.4, 1]\). Although \( \text{Precision}(D) \) of DataSet1 is monotonic decreasing, however, when \( \alpha \) is in the interval \([0, 0.4]\), \( \text{Precision}(D) \) of DataSet1 decreases gradually. While \( \alpha \) is in the interval \([0.4, 1]\), \( \text{Precision}(D) \) of DataSet1 decreases steeply. In other words, the ability of the layout tree to eliminate layout different blocks decreases steeply when \( \alpha \) is greater than 0.4. In summary, we conclude that \( \alpha = 0.4 \) is the optimal distance threshold.

6.4 Comparison with Other Methods

For evaluating the performance of LTDE, we conducted comparisons with existing methods. In this experiment, MDR [15] was employed as the comparison baseline. This experiment also compared our method with six existing methods, namely, TPC [13], STEM [11], RST [10], ViPER [22], VIPS [8] and ViDE [12]. The program of MDR is open on the Web, and we reproduced the other six methods. We used DataSet1 and DataSet2 to test the eight methods respectively. Since DataSet1 is an open data set, the experiment results of TPC, STEM and RST on the DataSet1 are available. We set the distance threshold \( \alpha \) to be 0.4. Table 2 shows the experiment results of different methods.
Table 2 The experiment results of applying different methods to the two data sets

<table>
<thead>
<tr>
<th></th>
<th>DataSet1</th>
<th>DataSet2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P (D)  R (D)  F(D)</td>
<td>P (D)  R (D)  F(D)</td>
</tr>
<tr>
<td>MDR</td>
<td>0.679  0.641  0.659</td>
<td>0.553  0.503  0.527</td>
</tr>
<tr>
<td>TPC</td>
<td>0.962  0.931  0.946</td>
<td>0.893  0.874  0.883</td>
</tr>
<tr>
<td>STEM</td>
<td>0.970  0.960  0.965</td>
<td>0.902  0.893  0.897</td>
</tr>
<tr>
<td>RST</td>
<td>0.968  0.974  0.971</td>
<td>0.912  0.909  0.906</td>
</tr>
<tr>
<td>ViPER</td>
<td>0.972  0.967  0.969</td>
<td>0.935  0.931  0.933</td>
</tr>
<tr>
<td>ViPS</td>
<td>0.977  0.953  0.965</td>
<td>0.924  0.941  0.932</td>
</tr>
<tr>
<td>ViDE</td>
<td>0.962  0.952  0.957</td>
<td>0.949  0.937  0.943</td>
</tr>
<tr>
<td>LTDE</td>
<td>0.993  0.977  0.985</td>
<td>0.989  0.961  0.975</td>
</tr>
</tbody>
</table>

The result data of TPC, STEM and RST in DataSet1 are cited from [10], [11], [13]. In the results, the precision of our method is better than that of any other methods. The precision of LTDE outperforms MDR by more than 30%. The recall of LTDE is also better than that of the other methods. Moreover, we calculated the F-measure of these methods. Because F-measure takes into account both the precision and the recall of the results to compute the score, it is a significant criterion to evaluate the performance of these methods. The F-measure of LTDE is much better than any of the other methods. The results show that LTDE can improve the precision significantly on the premise of guaranteeing that the recall stays stable. Therefore, we can conclude that the effectiveness of LTDE is better than other existing methods.

It also should be noted that the precision and recall of HTML-based methods (MDR, TPC, STEM and RST) on DataSet2 were much lower than the results on DataSet1. In contrast, the results of vision-based methods (ViDE and LTDE) change little between DataSet1 and DataSet2. That is because DataSet2 contains many dynamical pages which are generated by Javascript after the pages are rendered by browser. Since the vision-based methods analyze the visual information from the browser renderings, these methods can obtain the complete information of web pages. Contrary to the vision-based methods, the HTML-based methods analyze the HTML files before the pages are completely generated, they can only get the part information of Web pages. The vision-based methods (ViDE and LTDE) have better robustness on DataSet2 than HTML-based methods. Meanwhile, the results of LTDE on DataSet1 is much better than that of ViDE. We can conclude that LTDE have better robustness than other methods.

7. Conclusion and Future Work

Due to the explosive growth and the popularity of the deep web, information extraction from deep web page has gained more and more attention. However, the HTML structure of web documents has become more complicated, making it more difficult to recognize the target content by only analyzing the HTML source code. In this paper, we proposed a method named LTDE to extract data records from a deep web. We consider the rectangular region of an HTML element in a page as a visual block. Three visual features of data records are used: the data records have similar layout (SL); the data records have similar shape and coordinate (SSC); the data record region has the largest area and most similar layout blocks (LAMSB). Based on these features, we transformed the elements’ layout of a visual block into a layout tree. By calculating the distance of layout trees, we clustered the visual blocks that have similar layout features. 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 cluster. From the experiments, we drew the following conclusions: (1) the optimal layout tree distance threshold is 0.4; (2) compared with other methods, LTDE has better effectiveness and robustness.

In the future, we plan to use the layout tree method to identify the layout template of a Web site. By using the template, the data extraction from multiple pages can be realized.

Acknowledgements

This research is supported by Supported by the National Natural Science Foundation of China (Grant No. 61502062), the China Postdoctoral Science Foundation under Grant 2014M560704, the Scientific Research Foundation for the Returned Overseas Chinese Scholars (State Education Ministry), the Fundamental Research Funds for the Central Universities Project No. 2015CDJX1Y, and the Basic and Advanced Research Program of Chongqing under Grant Nos. cstc2013jcyjA1658 and cstc2014jcyjA40054.

References


Jun Zeng received the BS and MS degree in software engineering from Chongqing University in 2007 and 2010. He received Ph.D. degree in advanced information technology of Kyushu University in 2013. Since 2013, he has been a lecturer in the school of software engineering of Chongqing University. His research interest includes data mining, search engine and service computing.

Brendan J. Flanagan received a BS in Information Technology (Computing Studies) from RMIT University in 2010. Since 2013, has been a graduate student in software engineering of Chongqing University. Research interests include: data mining and recommender system.

Feng Li received a BS in software engineering from Chongqing University in 2015. Since 2015, has been a graduate student in software engineering of Chongqing University. Research interests include: text mining, search engines, and e-learning.

Sachio Hirokawa received the BS and MS degree in Mathematics and PhD degree in Interdisciplinary Graduate School of Engineering Sciences from Kyushu University in 1977, 1979, and 1992. Since 1997, he has been a professor in research institute for information technology of Kyushu University. His research interest includes search engine, text mining, and computational logic.