ViDE: A Vision-based Approach for Deep Web Data Extraction

Wei Liu, Xiaofeng Meng, *Member, IEEE,* and Weiyi Meng, *Member, IEEE*

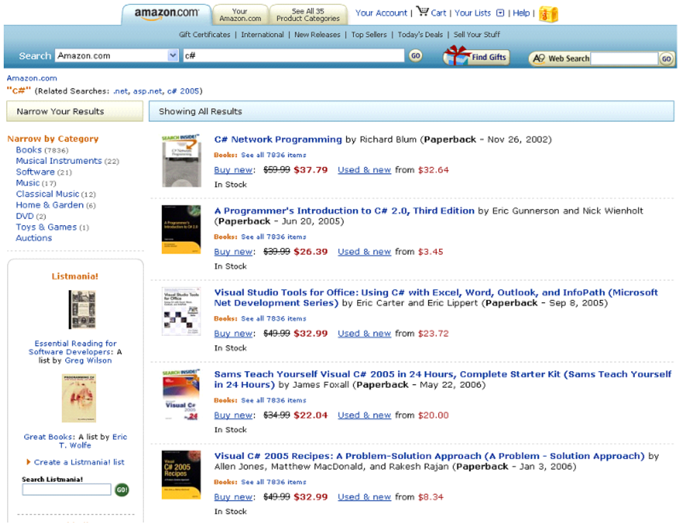
Abstract—Deep web contents are accessed by queries submitted to web databases and the returned data records are enwrapped in dynamically generated web pages (they will be called *deep web pages* in this paper). Extracting structured data from deep web pages is a challenging problem due to the underlying intricate structures of such pages. Until now, a large number of techniques have been proposed to address this problem, but all of them have inherent limitations because they are web page programming language dependent. As the popular two-dimensional media, the contents on web pages are always displayed regularly for users to browse. This motivates us to seek a different way for deep web data extraction to overcome the limitations of previous works by utilizing some interesting common visual features on the deep web pages. In this paper, a novel vision-based approach that is web page programming language independent is proposed. This approach primarily utilizes the visual features on the deep web pages to implement deep web data extraction, including data record extraction and data item extraction. We also propose a new evaluation measure *revision* to capture the amount of human effort needed to produce perfect extraction. Our experiments on a large set of web databases show that the proposed vision-based approach is highly effective for deep web data extraction.

Index Terms—Web mining, web data extraction, visual features of deep web pages, wrapper generation.

1 Introduction

The World Wide web has more and more online web databases which can be searched through their web query interfaces. The number of web databases has reached 25 millions according to a recent survey [21]. All the web databases make up the deep web (hidden web or invisible web). Often the retrieved information (query results) is enwrapped in web pages in the form of data records. These special web pages are generated dynamically and are hard to index by traditional crawler-based search engines, such as Google and Yahoo. In this paper, we call this kind of special web pages *deep web pages*. Each data record on the deep web pages corresponds to an object. For instance, Fig. 1 shows a typical deep web page from Amazon.com. On this page, the books are presented in the form of data records, and each data record contains some data items such as title, author, etc. In order to ease the consumption by human users, most web databases display data records and data items regularly on web browsers.

However, to make the data records and data items in them machine processable, which is needed in many applications such as deep web crawling and meta-searching, the structured data need to be extracted from the deep web pages. In this paper, we study the problem of automatically extracting the structured data, including data records and data items, from the deep web pages.



C# Network Programming by Richard Blum (**Paperback** - Nov 26, 2002)
Books: See all 7836 items
Buy new: ~~$59.99~~ $37.79 Used & new from $32.64
In Stock

A Programmer's Introduction to C# 2.0, Third Edition by Eric Gunnerson and Nick Wienholt (**Paperback** - Jun 20, 2005)
Books: See all 7836 items
Buy new: ~~$39.99~~ $26.39 Used & new from $3.45
In Stock

Visual Studio Tools for Office: Using C# with Excel, Word, Outlook, and InfoPath (Microsoft Net Development Series) by Eric Carter and Eric Lippert (**Paperback** - Sep 8, 2005)
Books: See all 7836 items
Buy new: ~~$49.99~~ $32.99 Used & new from $23.72
In Stock

Sams Teach Yourself Visual C# 2005 in 24 Hours, Complete Starter Kit (Sams Teach Yourself in 24 Hours) by James Foxall (**Paperback** - May 22, 2006)
Books: See all 7836 items
Buy new: ~~$34.99~~ $22.04 Used & new from $20.00
In Stock

Visual C# 2005 Recipes: A Problem-Solution Approach (A Problem - Solution Approach) by Allen Jones, Matthew MacDonald, and Rakesh Rajan (**Paperback** - Jan 3, 2006)
Books: See all 7836 items
Buy new: ~~$49.99~~ $32.99 Used & new from $8.34
In Stock

Fig. 1. An example deep web page from Amazon.com

The problem of web data extraction has received a lot of attention in recent years and most of the proposed solutions are based on analyzing the HTML source code or the tag trees of the web pages (please see Section II for a review of these works). These solutions have the following main limitations: First, they are web page programming language dependent, or more precisely, HTML dependent. As most web pages are written in HTML, it is not surprising that all previous solutions are based on analyzing the HTML source code of web pages. However, HTML itself is still evolving (from version 2.0 to the current version 4.01, and version 5.0 is being drafted [14]) and when new versions or new tags are introduced, the previous works will have to be amended repeatedly to adapt to new versions or new tags. Furthermore, HTML is no longer the exclusive web

- *Wei Liu is with the School of Information, Renmin University of China, China. E-mail: E-mail: gue1976@ gmail.com*
- *Xiaofeng Meng is with the School of Information, Renmin University of China, China. E-mail: xfmeng@ruc.edu.cn.*
- *Weiyi Meng is with Department of Computer Science, SUNY at Binghamton Binghamton, New York. E-mail: meng@cs.binghamton.edu.*

page programming language, and other languages have been introduced, such as XHTML and XML (combined with XSLT and CSS). The previous solutions now face the following dilemma: should they be significantly revised or even abandoned? Or should other approaches be proposed to accommodate the new languages? Second, they are incapable of handling the ever-increasing complexity of HTML source code of web pages. Most previous works have not considered the scripts, such as JavaScript and CSS, in the HTML files. In order to make web pages vivid and colorful, web page designers are using more and more complex JavaScript and CSS. Based on our observation from a large number of real web pages, especially deep web pages, the underlying structure of current web pages is more complicated than ever and is far different from their layouts on web browsers. The makes it more difficult for existing solutions to infer the regularity of the structure of web pages by only analyzing the tag structures.

Meanwhile, to ease human users' consumption of the information retrieved from search engines, good template designers of deep web pages always arrange the data records and the data items with visual regularity to meet the reading habits of human beings. For example, all the data records in Fig. 1 are clearly separated, and the data items of the same semantic in different data records are similar on layout and font.

In this paper, we explore the visual regularity of the data records and data items on deep web pages and propose a novel vision-based approach, ViDE (Vision-based Data Extractor), to extract structured results from deep web pages automatically. ViDE is primarily based on the visual features human users can capture on the deep web pages while also utilizing some simple non-visual information such as data types and frequent symbols to make the solution more robust. ViDE consists of two main components, ViDRE (Vision-based Data Record extractor) and ViDIE (Vision-based Data Item extractor). By using visual features for data extraction, ViDE avoids the limitations of those solutions that need to analyze complex web page source files.

Our approach employs a four-step strategy. First, given a sample deep web page from a web database, obtain its visual representation and transform it into a Visual Block tree which will be introduced later; second, extract data records from the Visual Block tree; third, partition extracted data records into data items and align the data items of the same semantic together; fourth, generate visual wrappers (a set of visual extraction rules) for the web database based on sample deep web pages such that both data record extraction and data item extraction for new deep web pages that are from the same web database can be carried out more efficiently using the visual wrappers.

To our best knowledge, although there are already some works [3][4][23][26][28] that pay attention to the visual information on web pages, our work is the first to perform deep web data extraction using primarily visual features. Our approach is independent of any specific web page programming language. Although our current implementation uses the VIPS algorithm [4] to obtain a deep web page's Visual Block tree and VIPS needs to analyze the HTML source code of the page, our solution is independent of any specific method used to obtain the Visual Block tree in the sense that any tool that can segment the web pages into a tree structure based on the visual information, not HTML source code, can be used to replace VIPS in the implementation of ViDE.

In this paper, we also propose a new measure, *revision*, to evaluate the performance of web data extraction tools. It is the percentage of the web databases whose data records or data items cannot be perfectly extracted (i.e., at least one of the precision and recall is not 100%). For these web databases, manual revision of the extraction rules is needed to achieve perfect extraction.

In summary, this paper has the following contributions. (1) A novel technique is proposed to perform data extraction from deep web pages using primarily visual features. We open a promising research direction where the visual features are utilized to extract deep web data automatically. (2) A new performance measure, *revision*, is proposed to evaluate web data extraction tools. This measure reflects how likely a tool will fail to generate a perfect wrapper for a site. (3) A large data set consisting of 1,000 web databases across 42 domains is used in our experimental study. In contrast, the data sets used in previous works seldom had more than 100 web databases. Our experimental results indicate that our approach is very effective.

The rest of the paper is organized as follows. Related works are reviewed in Section 2. Visual representation of deep web pages and visual features on deep web pages are introduced in Section 3. Our solutions to data record extraction and data item extraction are described in Sections 4 and 5, respectively. Wrapper generation is discussed in Section 6. Experimental results are reported in Section 7. Finally, concluding remarks are given in Section 8.

2 RELATED WORK

A number of approaches have been reported in the literature for extracting information from web pages. Good surveys about previous works on web data extraction can be found in [16] and [5]. In this section, we briefly review previous works based on the degree of automation in web data extraction, and compare our approach with fully automated solutions since our approach belongs to this category.

2.1 Manual Approaches

The earliest approaches are the manual approaches in which languages were designed to assist programmer in constructing wrappers to identify and extract all the desired data items/fields. Some of the best-known tools that adopt manual approaches are Minerva [7], TSIMMIS [11] and Web-OQL [1]. Obviously, they have low efficiency and are not scalable.

2.2 Semi-automatic Approaches

Semi-automatic techniques can be classified into sequence-based and tree-based. The former, such as WIEN [15], Soft-Mealy [12] and Stalker [22], represent documents as sequences of tokens or characters, and generate delimiter-based extraction rules through a set of training examples. The latter, such as W4F [24] and XWrap [19], parse the document into a hierarchical tree (DOM tree), based on which they perform the extraction process. These approaches require manual efforts, for example, labeling some sample pages, which is labor intensive and time consuming.

2.3 Automatic Approaches

In order to improve the efficiency and reduce manual efforts, most recent researches focus on automatic approaches instead of manual or semi-automatic ones. Some representative automatic approaches are Omini [2], RoadRunner [8], IEPAD [6], MDR [17], DEPTA [29], and the method in [9]. Some of these approaches perform only data record extraction but not data item extraction, such as Omini and the method in [9]. RoadRunner, IEPAD, MDR, DEPTA, Omini and the method in [9] do not generate wrappers, i.e., they identify patterns and perform extraction for each web page directly without using previously derived extraction rules. The techniques of these works have been discussed and compared in [5], and we do not discuss them any further here. Note that all of them mainly depend on analyzing the source code of Web pages. As a result, they cannot avoid the inherent limitations described in the introduction. In addition, there are several works (DeLa [27], DEPTA and the method in [20]) on data item extraction, which is a preparation step for holistic data annotation, i.e., assigning meaningful labels to data items. DeLa utilizes HTML tag information to construct regular expression wrapper and extract data items into a table. Similar to DeLa, DEPTA also operates on HTML tag tree structures to first align data items in a pair of data records that can be matched with certainty. The remaining data items are then incrementally added. However, both data alignment techniques are mainly based on HTML tag tree structures, not visual information. The automatic data alignment method in [20] proposes a clustering approach to perform alignment based on five features of data items, including font of text. However, this approach is primarily text based and tag structure based while our method is primarily visual information based.

The only works that we are aware of that utilize some visual information to extract web data are ViNTS [30], ViPERS [25], HCRF [32], and VENTex [10]. ViNTs uses the visual content features on the query result pages to capture content regularities denoted as Content Lines, and then utilizes the HTML tag structures to combine them. ViPER also incorporates visual information on a web page for data records extraction with the help of a global multiple sequence alignment technique. However, in the two approaches, tag structures are still the primary information utilized while visual information plays a small role. In addition, both of them only focus on data record extraction, without considering data item extraction. HCRF is a probabilistic model for both data record extraction and attribute labeling. Compared to our solution, it also uses VIPS algorithm [4] to represent web pages, but the tag information is still an important feature in HCRF. And furthermore, it is implemented under an ideal assumption that every record corresponds to one block in the Visual Block tree, but this assumption is not always correct according to our observation to the real web pages (about 20% of deep web pages do not meet this assumption). VENTex implements the information extraction from web tables based on a variation of the CSS2 visual box model. So it can be regarded as the only related work using pure visual features. The main difference between our approach and VENTex is their objectives. VENTex aims to extract various forms of tables that are embedded in common pages whereas our approach focuses on extracting regularly arranged data records and data items from deep web pages.

3 VISUAL BLOCK TREE AND VISUAL FEATURES

Before the main techniques of our approach are presented, we describe the basic concepts and visual features that our approach needs.

3.1 Visual Information of Web Pages

The information on web pages consists of both texts and images (static pictures, flash, video, etc.). The visual information of web pages used in this paper includes mostly information related to *web page layout* (location and size) and *font*.

3.1.1 Web page layout

A coordinate system can be built for every web page. The origin locates at the top left corner of the web page. The X-axis is horizontal left-right, and the Y-axis is vertical top-down. Suppose each text/image is contained in a minimum bounding rectangle with sides parallel to the axes. Then, a text/image can have an exact coordinate (x, y) on the web page. Here x refers to the horizontal distance between the origin and the left side of its corresponding rectangle, while y refers to the vertical distance between the origin and the upper side of its corresponding box. The size of a text/image is its height and width.

The coordinates and sizes of texts/images on the web page make up the web page layout.

Table 1.
Font attributes and examples

Font factor	Example	Font factor	Example
Size	A (10pt)	underline	A
face	A(Sans Serif)	italic	*A*
color	A (red)	weight	**A**
strikethrough	~~A~~	frame	$\boxed{A}$

3.1.2 Font

The fonts of the texts on a web page are also very useful visual information, which are determined by many attributes as shown in Table 1. Two fonts are considered to be the same only if they have the same value under each attribute.

3.2 Deep Web Page Representation

The visual information of web pages, which has been introduced above, can be obtained through the programming interface provided by web browsers (i.e. IE). In this paper, we employ the VIPS algorithm [4] to transform a deep web page into a Visual Block tree and to extract the visual information. A Visual Block tree is actually a segmentation of a web page. The root block represents the whole page, and each block in the tree corresponds to a rectangular region on the web page. The leaf blocks are the blocks that cannot be segmented further, and they represent the minimum semantic units, such as continuous texts or images. Fig. 2(a) shows a popular presentation structure of deep web pages and Fig. 2(b) gives its corresponding Visual Block tree. The technical details of building Visual Block trees can be found in [4]. An actual Visual Block tree of a deep web page may contain hundreds even thousands of blocks.

Visual Block tree has three interesting properties. First, block a contains block b if a is an ancestor of b. Second, a and b do not overlap if they do not satisfy property one. Third, the blocks with the same parent are arranged in the tree according to the order of the corresponding nodes appearing on the page. These three properties are illustrated by the example in Fig. 2. The formal representations for internal blocks and leaf blocks in our approach are given below. Each internal block a is represented as a = (CS, P, S, FS, IS), where CS is the set containing its child blocks (note that, the order of blocks is also kept), P is the position of a (its coordinates on the web page), S is its size (height and width), FS is the set of the fonts appearing in a, and IS is the number of images in a. Each leaf block b is represented as b = (P, S, F, L, I, C), where the meanings of P and S are the same as those of an inner block, F is the font it uses, L denotes whether it is a hyperlink text, I denotes whether it is an image, and C is its content if it is a text.

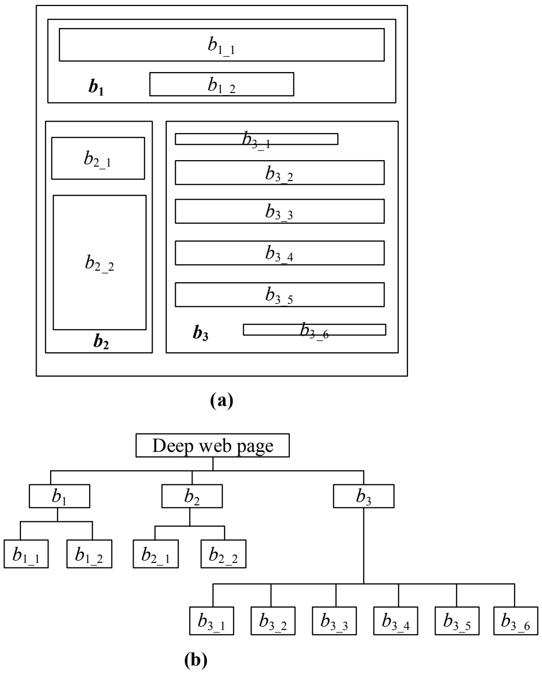


Fig. 2. The presentation structure (a) and its Visual Block tree (b)

3.3 Visual Features of Deep Web Pages

Web pages are used to publish information to users, similar to other kinds of media, such as newspaper and TV. The designers often associate different types of information with distinct visual characteristics (such as font, position, etc.) to make the information on web pages easy to understand. As a result, visual features are important for identifying special information on web pages. Deep web pages are special web pages that contain data records retrieved from web databases, and we hypothesize that there are some distinct visual features for data records and data items. Our observation based on a large number of deep web pages is consistent with this hypothesis. We describe the main visual features in this section, and show the statistics about the accuracy of these features at the end of this Subsection 3.3.

Position Features (*PF*): These features indicate the location of the data region on a deep web page.

PF1: Data regions are always centered horizontally.

PF2: The size of the data region is usually large relative to the area size of the whole page.

Since the data records are the contents in focus on deep web pages, web page designers always have the region containing the data records centrally and conspicuously placed on pages to capture the user's attention. By investigating a large number of deep web pages, we found two interesting facts. First, data regions are always located in the middle section horizontally on deep web pages. Second, the size of a data region is usually large when there are enough data records in the data region. The actual size of a data region may change greatly because it is not only influenced by the number of data records retrieved but also by what information is

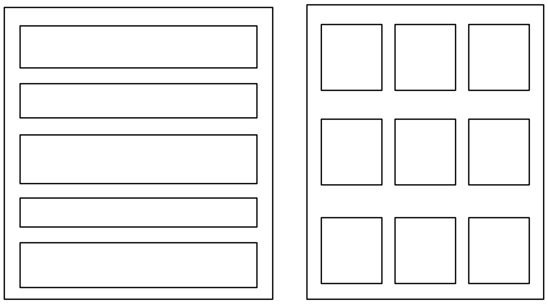


Fig. 3. Layout models of data records on deep web pages

Table 2
Relevant visual information about the top five data records in Fig. 1

	Images (pixel)	plain texts		link texts	
		Total font number	Shared font number	Total font number	Shared font number
record1	115*115	5	5	2	2
record2	115*115	5	5	2	2
record3	115*110	5	5	2	2
record4	115*115	5	5	2	2
record5	115*115	5	5	2	2

included in each data record. Therefore, our approach uses the ratio of the size of the data region to the size of whole deep web page instead of the actual size. In our experiments in Section 7, the threshold of the ratio is set at 0.4, that is, if the ratio of the horizontally centered region is greater than or equal to 0.4, then the region is recognized as the data region.

Layout Features (*LF*): These features indicate how the data records in the data region are typically arranged.

LF1: The data records are usually aligned flush left in the data region.

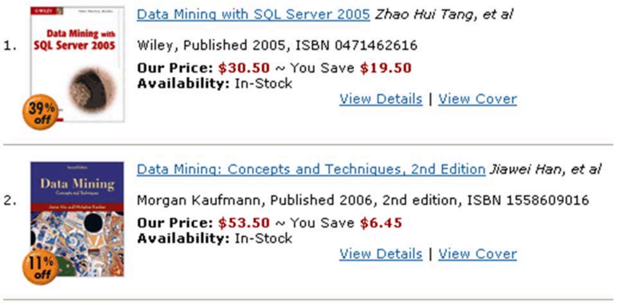
LF2: All data records are adjoining.

LF3: Adjoining data records do not overlap, and the space between any two adjoining records is the same.

Data records are usually presented in one of the two layout models shown in Fig. 3. In Model 1, the data records are arranged in a single column evenly, though they may be different in width and height. LF1 implies that the data records have the same distance to the left boundary of the data region. In Model 2, data records are arranged in multiple columns, and the data records in the same column have the same distance to the left boundary of the data region. Because most deep web pages follow the first model, we only focus on the first model in this paper, and the second model can be addressed with minor implementation expansion to our current approach. In addition, data records do not overlap, which means that the regions of different data records can be separated.

Appearance Features (*AF*): These features capture the visual features within data records.

AF1: Data records are very similar in their appearances, and the similarity includes the sizes of the images they contain and the fonts they use.



1. Data Mining with SQL Server 2005 *Zhao Hui Tang, et al*
Wiley, Published 2005, ISBN 0471462616
Our Price: $30.50 ~ You Save $19.50
Availability: In-Stock
View Details | View Cover

2. Data Mining: Concepts and Techniques, 2nd Edition *Jiawei Han, et al*
Morgan Kaufmann, Published 2006, 2nd edition, ISBN 1558609016
Our Price: $53.50 ~ You Save $6.45
Availability: In-Stock
View Details | View Cover

Fig. 4. Illustrating visual features of deep web pages

AF2: The data items of the same semantic in different data records have similar presentations with respect to position, size (image data item) and font (text data item).

AF3: The neighboring text data items of different semantics often (not always) use distinguishable fonts.

AF1 describes the visual similarity at the data record level. Generally, there are three types of data contents in data records, i.e. images, plain texts (the texts without hyperlinks) and link texts (the texts with hyperlinks). Table 2 shows the information on the three aspects for the data records in Fig. 1. We can see that these five data records are very close on the three aspects. *AF2* and *AF3* describe the visual similarity at the data item level. The text data items of the same semantic always use the same font, and the image data items of the same semantic are often similar in size. The positions of data items in their respective data records can be classified into two kinds: *absolute position* and *relative position*. The former means that the positions of the data items of certain semantic are fixed in the line they belong to, while the latter refers to the position of a data item relative to the data item ahead of it. Furthermore, the items of the same semantic from different data records share the same kind of position. *AF3* indicates that the neighboring text data items of different semantics often use distinguishable fonts. However, *AF3* is not a robust feature because some neighboring data items may use the same font. Neighboring data items with the same font are treated as a *composite data item*. Composite data items have very simple string patterns and the real data items in them can often be separated by a limited number of symbols, such as ",", "/" etc. In addition, the composite data items of the same semantics share the same string pattern. Hence it's easy to break composite data items into real data items using some predefined separating symbols. For example, in Fig. 4, four data items, publisher, publishing date, edition and ISBN, form a composite data item, and they are separated by commas. According to our observation to deep web pages, the granularity of the data items extracted is not larger than what HTML tags can separate, because a composite data item is always included in one leaf node in the tag tree.

Content Feature (*CF*): These features hint the regular-

Table 3
The statistics on the visual features

Feature type		Statistics	Feature type		Statistics
Position Features	*PF1*	99.9%	Appearance Features	*AF1*	99.5%
				AF2	100%
	PF2	99.9%		*AF3*	92.8%
Layout Features	*LF1*	99.3%	Content Features	*CF1*	100%
	LF2	100%		*CF2*	100%
	LF3	100%		*CF3*	6.5%

ity of the contents in data records.

CF1: The first data item in each data record is always of a mandatory type.

CF2: The presentation of data items in data records follows a fixed order.

CF3: There are often some fixed static texts in data records, which are not from the underlying web database.

The data records correspond to the entities in real world, and they consist of data items with different semantics that describe the attribute values of the entities. The data items can be classified into two kinds: *mandatory* and *optional*. Mandatory data items appear in all data records. For example, if every data record must have a title, then titles are mandatory data items. In contrast, optional items may be missing in some data records. For example, ″discounted price″ for products is likely an optional unit. The order of different types of data items from the same web database is always fixed in data records. For example, the order of attributes of data records from Bookpool.com in Fig. 4 is ″title″, ″author″, ″publisher″, ″publish time″, ″edition″, ″ISBN″, ″discount price″, ″save money″, ″availability″, etc. Fixed static texts refer to the texts that appear in every data record. Most of them are meaningful labels that can help users understand the semantics of data items, such as ″Buy new″ in Fig. 4. We call these static texts static items, which are part of the record template.

Our deep web data extraction solution is developed mainly based on the above four types of visual features. *PF* is used to locate the region containing all the data records on a deep web page; *LF* and *AF* are combined together to extract the data records and data items.

Statistics on the Visual Features

To verify the robustness of these visual features we observed, we examined these features on 1,000 deep web pages of different web databases from the General Data Set (GDS) used in our experiments (see Section 7 for more information about GDS). The results are shown in Table 3. For most features (except *AF3* and *CF3*), their corresponding statistics are the percentages of the deep web pages that satisfy them. For example, the statistics of 99.9% for *PF1* means that for 99.9% of the deep web pages, *PF1* feature ″data regions are always centered horizontally″ is true. From the statistics we can conclude that these visual features are very robust and can be reliably applied to general deep web pages. For *AF3*, 92.8% is the percentage of the data items that have different font from their following data items. For *CF3*, 6.5% is the percentage of the static data items over all data items.

Table 4
Non-visual Information Used

Special complementary information	Remarks
Same text	Given two texts, we can determine whether or not they are the same.
Frequent symbol	Given the deep web pages of a web database, if some symbols/words (e.g., ISBN, $) appear in all the data items of an attribute, they are called frequent symbols.
Data type	They are predefined, including image, text, number, date, price, email, etc

We should point out that when a feature is not satisfied by a page, it does not mean that ViDE will fail to process this page. For example, our experiments using the datasets to be described in Section 7 show that among the pages that violate *LF*3, 71.4% can still be processed successfully by ViDE, and among the pages that violate *AF*1, 80% can still be correctly processed.

3.4 Special supplementary information

Several types of simple non-visual information are also used in our approach in this paper. They are *same text*, *frequent symbol*, and *data type*, as explained in Table 4.

Obviously, the above information is very useful to determine whether the data items in different data records from the same web database belong to the same semantic. The above information can be captured easily from the web pages using some simple heuristic rules without the need to analyze the HTML source code or the tag trees of the web pages. Furthermore, they are specific language (i.e. English, French, etc) independent.

4 DATA RECORDS EXTRACTION

Data record extraction aims to discover the boundary of data records and extract them from the deep web pages. An ideal record extractor should achieve the following: (1) all data records in the data region are extracted; (2) for each extracted data record, no data item is missed and no incorrect data item is included.

Instead of extracting data records from the deep web page directly, we first locate the data region, and then extract data records from the data region. *PF1* and *PF2* indicate that the data records are the primary content

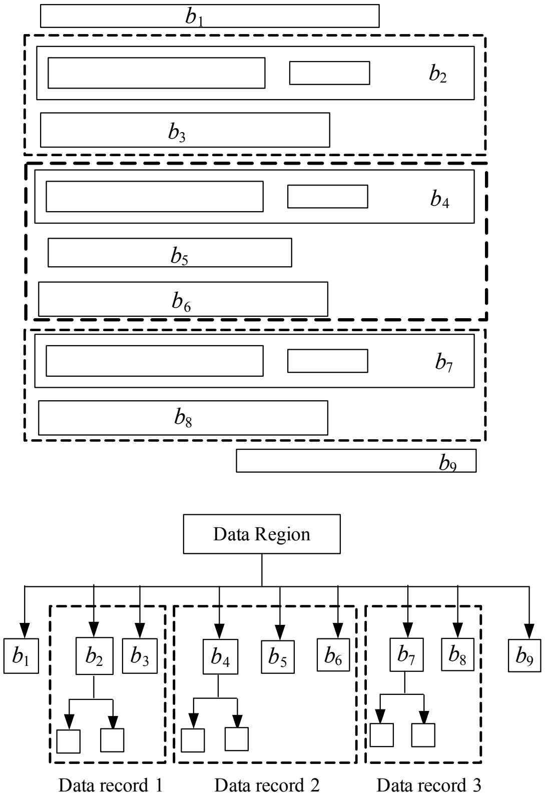


Fig. 5. A general case of data region

on the deep web pages and the data region is centrally located on these pages. The data region corresponds to a block in the Visual Block tree. We locate the data region by finding the block that satisfies the two position features. Each feature can be considered as a rule or a requirement. The first rule can be applied directly, while the second rule can be represented by $(area_b/area_{page}) > T_{region}$, where $area_b$ is the area of block b, $area_{page}$ is the area of the whole deep web page, and T_{region} is a threshold. The threshold is trained from sample deep web pages. If more than one block satisfies both rules, we select the block with the smallest area. Though very simple, this method can find the data region in the Visual Block tree accurately and efficiently.

Each data record corresponds to one or more sub-trees in the Visual Block tree, which are just the child blocks of the data region, as Fig. 5 shows. So we only need to focus on the child blocks of the data region. In order to extract data records from the data region accurately, two facts must be considered. First, there may be blocks that do not belong to any data record, such as the statistical information (e.g., about 2,038 matching results for java) and annotation about data records (e.g., 1 2 3 4 5 [Next]). These blocks are called *noise blocks* in this paper. Noise blocks may appear in the data region because they are often close to the data records. According to *LF2*, noise blocks cannot appear between data records. They always appear at the top or the bottom of the data region.

Table 5
Formulas and remarks

formulas	remarks
$simIMG(b_1,b_2)=\frac{Min\{sa_i(b_1),sa_i(b_2)\}}{Max\{sa_i(b_1),sa_i(b_2)\}}$	*$sa_i(b)$* is the total area of images in block *b*. *$sa_b(b)$* is the total area of block *b*. *$fn_{pt}(b)$* is the total number of fonts of the plain texts in block *b*. *$sa_{pt}(b)$* is the total area of the plain texts in block *b*. *$fn_{lt}(b)$* is the total number of fonts of the link texts in block *b*. *$sa_{lt}(b)$* is the total area of the link texts in block *b*.
$w_i=\frac{sa_i(b_1)+sa_i(b_2)}{sa_b(b_1)+sa_b(b_2)}$	
$simPT(b_1,b_2)=\frac{Min\{fn_{pt}(b_1),fn_{pt}(b_2)\}}{Max\{fn_{pt}(b_1),fn_{pt}(b_2)\}}$	
$w_{pt}=\frac{sa_{pt}(b_1)+sa_{pt}(b_2)}{sa_b(b_1)+sa_b(b_2)}$	
$simLT(b_1,b_2)=\frac{Min\{fn_{lt}(b_1),fn_{lt}(b_2)\}}{Max\{fn_{lti}(b_1),fn_{lt}(b_2)\}}$	
$w_{lt}=\frac{sa_{lt}(b_1)+sa_{lt}(b_2)}{sa_b(b_1)+sa_b(b_2)}$	

Second, one data record may correspond to one or more blocks in the Visual Block tree, and the total number of blocks one data record contains is not fixed. In Fig. 5, block b_1 (statistical information) and b_9 (annotation) are noise blocks; there are three data records (b_2 and b_3 form data record 1; b_4, b_5 and b_6 form data record 2; b_7 and b_8 form data record 3), and the dashed boxes are the boundaries of data records.

Data record extraction is to discover the boundary of data records based on the *LF* and *AF* features. That is, we attempt to determine which blocks belong to the same data record. We achieve this in the following three phases:

Phase 1: Filter out some noise blocks.

Phase 2: Cluster the remaining blocks by computing their appearance similarity.

Phase 3: Discover data record boundary by regrouping blocks.

4.1 Phase 1: Noise Blocks Filtering

Because noise blocks are always at the top or bottom, we check the blocks located at the two positions according to *LF1*. If a block at these positions is not aligned flush left, it will be removed as a noise block. This step does not guarantee the removal of all noise blocks. For example, in Fig. 5, block b_9 can be removed in this step while block b_1 cannot be removed.

4.2 Phase 2: Blocks Clustering

The remaining blocks in the data region are clustered based on their appearance similarity. Since there may be three kinds of information in data records, i.e., images, plain text and link text, the appearance similarity between blocks is computed from the three aspects. For images, we care about the size; for plain text and link text, we care about the shared fonts. Intuitively, if two blocks are more similar on image size and font, they should be more similar in appearance. The formula for computing the appearance similarity between two blocks b_1 and b_2 is given below:

$$sim(b_1, b_2) = w_i * simIMG(b_1, b_2) + w_{pt} * simPT(b_1, b_2) + w_{lt} * simLT(b_1, b_2) \qquad (1)$$

where $simIMG(b_1, b_2)$, $simIMG(b_1, b_2)$, $simLT(b_1, b_2)$ are the similarities based on image size, plain text font and link text font, respectively. And w_i, w_{pt} and w_{lt} are the weights of these similarities, respectively. Table 5 gives the formulas to compute the component similarities and the weights in different cases. The weight of one type of contents is proportional to their total size relative to the total size of the two blocks.

A simple one-pass clustering algorithm is applied. First, starting from an arbitrary order of all the input blocks, take the first block from the list and use it to form a cluster. Next, for each of the remaining blocks, say b, compute its similarity with each existing cluster. Let C be the cluster that has the largest similarity with A. If $sim(b, C) > T_{as}$ for some threshold T_{as}, which is to be trained by sample pages (generally, T_{as} is set to 0.8), then add b to C; otherwise, form a new cluster based on b. Function $sim(b, C)$ is defined to be the average of the similarities between b and all blocks in C computed using Formula 1. As an example, by applying this method to the blocks in Fig. 1, the blocks containing the titles of the data records are clustered together, so are the blocks containing the prices and so on.

4.3 Phase 3: Blocks Regrouping

The clusters obtained in the previous step do not correspond to data records. On the contrary, the blocks in the same cluster all come from different data records. According to *AF2*, the blocks in the same cluster have the same type of contents of the data records.

The blocks need to be regrouped such that the blocks belonging to the same data record form a group. Our basic idea of blocks regrouping is as follows. According to *CF1*, the first data item in each data record is always mandatory. Clearly, the cluster that contains the blocks for the first items has the maximum number of blocks possible; let n be this maximum number. It is easy to see that if a cluster contains n blocks, these blocks contain mandatory data items. Our regrouping method first selects a cluster with n blocks and use these blocks as seeds to form data records. Next, given a block b, we determine which record b belongs to according to *CF2*. For example, suppose we know that title is ahead of author in each record and they belong to different blocks (say an author block and a title block). Each author block should relate to the nearest title block that is ahead of it. In order to determine the order of different semantic blocks, a minimum-bounding rectangle is formed for each cluster on the page. By comparing the positions of these rectangles on the page, we can infer the order of the semantics. For example, if the rectangle enclosing all title blocks is higher than the rectangle enclosing the author blocks, then title must be ahead of its corresponding author. Based on this idea, the algorithm of block regrouping is developed as shown in Fig. 6.

```
Algorithm block regrouping
Input: $C_1,C_2,\ldots,C_m$: a group of clusters generated by blocks clustering from a given sample deep web page $P$
Output: $G_1,G_2,\ldots,G_n$: each of them corresponds to a data record on $P$
Begin
//Step 1. sort the blocks in $C_i$ according to their positions in the page (from top to bottom and then from left to right)
1 for each cluster $C_i$ do
2     for any two blocks $b_{i,j}$ and $b_{i,k}$ in $C_i$    //$1 \le j < k \le |C_i|$
3       if $b_{i,j}$ and $b_{i,k}$ are in different lines on $P$, and $b_{i,k}$ is above $b_{i,j}$
4             $b_{i,j} \leftrightarrow b_{i,k}$;      //exchange their orders in $C_i$;
5       else if $b_{i,j}$ and $b_{i,k}$ are in the same line on $P$, and $b_{i,k}$ is in front of $b_{i,j}$
6             $b_{i,j} \leftrightarrow b_{i,k}$;
7  end until no exchange occurs;
8  form the minimum-bounding rectangle $Rec_i$ for $C_i$;
//Step 2. initialize $n$ groups, and $n$ is the number of data records on $P$
9  $C_{max}=\{C_i \mid |C_i|=\max\{|C_1|,|C_2|,\ldots\ldots,|C_m|\}\}$;   // $n=|C_{max}|$
10 for each block $b_{max,i}$ in $C_{max}$
11     Initialize group $G_i$;
12     put $b_{max,i}$ into $G_i$;
//Step 3. put the blocks into the right groups, and each group corresponds to a data record
13 for each cluster $C_i$
14     if $Rec_i$ overlaps with $Rec_{max}$ on $P$
15         if $Rec_i$ is ahead of (behind) $Rec_{max}$
16             for each block $b_{i,j}$ in $C_i$
17                 find the nearest block $b_{max,k}$ in $C_{max}$ that is behind (ahead
                       of) $b_{i,j}$ on the web page;
18                 place $b_{i,j}$ into group $G_k$;
End
```

Fig. 6. The algorithm of blocks regrouping

This algorithm consists of three steps. Step 1 rearranges the blocks in each cluster based on their appearance order on the web page, i.e., from left to right and from top to bottom (lines 1-7). In addition, a minimum-bounding rectangle is formed for each cluster on the page (line 8). In Step 2, n groups are initialized with a seed block in each group as discussed earlier, where n is the number of blocks in a maximum cluster, denoted as C_{max}. According to *CF1*, we always choose the cluster that contains the first mandatory data item of each record as C_{max}. Let $b_{max,k}$ denote the seed block in each initial group Gk. Step 3 determines to which group each block belongs. If block $b_{i,j}$ (in C_i, C_i is not C_{max}) and block $b_{max,k}$ (in C_{max}) are in the same data record, then $b_{i,j}$ should be put into the same group $b_{max,k}$ belongs to. According to *LF3*, no two adjoining data records overlap. So for $b_{max,k}$ in C_{max}, the blocks that belong to the same data record with $b_{max,k}$ must be below $b_{max,k-1}$ and above $b_{max,k+1}$. For each C_i, if data record R_i is ahead

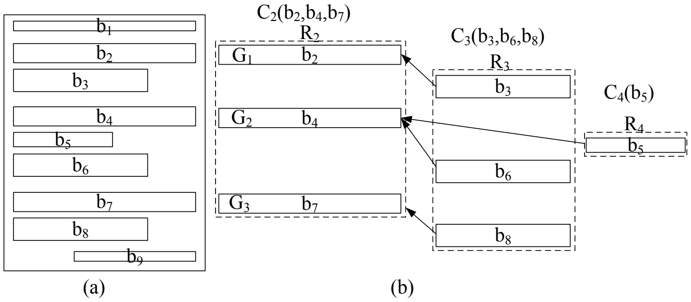


Fig. 7. An illustration of data record extraction

of R_{max}, then the block on top of R_i is ahead of (behind) the block on top of R_{max}. Here "ahead of" means "on the left of" or "above", and "behind" means "on the right of" or "below". According to *CF2*, $b_{i,j}$ is ahead of $b_{max,k}$ if they belong to the same data record. So we can conclude that, if $b_{max,k}$ is the nearest block behind $b_{i,j}$, then $b_{i,j}$ should be put into the group $b_{max,k}$ belongs to. Obviously, the complexity of this algorithm is $O(n^2)$, where n is the number of data records in the sample page.

Example for data record extraction:

Fig. 7 illustrates the case in Fig. 5. First, b_9 is removed according to *LF1*. Then, the blocks on the left in Fig. 7 (b) are clustered using Formula 1. Altogether, four clusters are formed and the blocks in them are also rearranged: $C_1\{b_1\}, C_2\{b_2, b_4, b_7\}, C_3\{b_3, b_6, b_8\}$, and $C_4\{b_5\}$. Next, C_2 is $C_{max}, b_2, b_4, and b_7$ form three initial groups G_1, G_2 and G_3 respectively. Since R_3 and R_4 overlap with R_2 and R_3 is below R_2, we group b_3, b_6 and b_8 with b_2, b_4 and b_7 (the nearest block above it in C_2), respectively. At last, G_1 is $\{b_2, b_3\}$, G_2 is $\{b_4, b_5, b_6\}$, and G_3 is $\{b_7, b_8\}$. Each group forms a complete data record.

5 DATA ITEM EXTRACTION

A data record can be regarded as the description of its corresponding object, which consists of a group of data items and some static template texts. In real applications, these extracted structured data records are stored (often in relational tables) at data item level and the data items of the same semantic must be placed under the same column. When introducing *CF*, we mentioned that there are three types of data items in data records: mandatory data items, optional data items and static data items. We extract all three types of data items. Note that static data items are often annotations to data and are useful for future applications, such as web data annotation. Below we focus on the problems of segmenting the data records into a sequence of data items and aligning the data items of the same semantics together.

Note that data item extraction is different from data record extraction; the former focuses on the leaf nodes of the Visual Block tree while the latter focuses on the child blocks of the data region in the Visual Block tree.

```
Algorithm data item matching
Input: item$_1$, item$_2$: two data items
Output: matched or unmatched: the match result (Boolean)
Begin
1 if (font(item$_1$) ≠ font(item$_2$))
2     Return unmatched;
3 if (position(item$_1$) = position(item$_2$))
4     return matched;
5 if (item$_{p1}$ and item$_{p2}$ are matched)    // item$_{p1}$ and item$_{p2}$ are the data items immediately in front of item$_1$ and item$_2$ respectively
6       return matched;
7 else
      return unmatched;
End
```

Fig. 8. The algorithm of data item matching

5.1 Data Record Segmentation

*AF*3 indicates that composite data items cannot be segmented any more in the Visual Block tree. So, given a data record, we can collect its leaf nodes in the Visual Block tree in left to right order to carry out data record segmentation. Each composite data item also corresponds to a leaf node. We can treat it as a regular data item initially and then segment it into the real data items with the heuristic rules mentioned in *AF3* after the initial data item alignment.

5.2 Data Item Alignment

CF1 indicates that we cannot align data items directly due to the existence of optional data items. It is natural for data records to miss some data items in some domains. For example, some books have discount price while some do not.

Every data record has been turned into a sequence of data items through data record segmentation. Data item alignment focuses on the problem of how to align the data items of the same semantic together and also keep the order of the data items in each data record. In the follow ing we first define visual matching of data items, and then propose an algorithm for data item alignment.

5.2.1 *Visual matching of data items*

AF2 indicates that, if two data items from different data records belong to the same semantic, they must have consistent font and position, including both absolute position and relative position. In Fig. 8, a simple algorithm to match two visually similar data items from different data records is described.

The first four lines of the algorithm say that two data items are matched only if they have the same absolute position in addition to having the same font. Here absolute position is the distance between the left side of the data region and the left side of a data item. When two data items do not have the same absolute position, they can still be matched if they have the same relative position. For match on relative position,

```
Algorithm data item alignment
Input: a set of extracted data records {ri|1≤i≤n}
Output: a set of data records {ri|1≤i≤n} with all the data items
aligned
Begin
1  currentItemSet=ϕ;
2  currentCluster=ϕ;
//put the first unaligned data item of each ri into currentItemSet:
// ItemiU(i) refers to the first unaligned item of the ith data record
3   currentItemSet ↔ ItemiU(i) (1≤i≤n);
4   while currentItemSet≠ϕ
5         use the data item matching algorithm to group the data items
in currentItemSet into k clusters {Ci|1≤i≤k} (k≤n);
6         for each cluster Ci
7            for each rj that does not have a data item in Ci
8              if ItemjU(j)+k is matched with data items in Ci
9                      Log position k;
10             else
11                     Log position 0;
12   Pi = max value of these logged positions for Ci;
     /*Till now, each cluster Ci has a position Pi */
13   if any PL==0
14      currentCluster=CL;
15   else
16      currentCluster=CL whose PL is max {P1, P2,…,PK};
17   for each rj whose ItemjU(j) is in currentCluster CL
18      remove ItemjU(j) from currentItemSet;
19      if ItemjU(j)+1 exists in rj
20            put ItemjU(j)+1 into currentItemSet;
21   for each rj that has no item in currentCluster CL
22      insert a blank item ahead of ItemjU(j)  in rj ;
23    U(j)++;
End
```

Fig. 9. The algorithm of data item alignment

the data items immediately before the two input data items should be matched (from line 5 to line 6). As an example, for the two records in Fig. 4, the titles can be matched based on the absolute positions and the authors on the relative positions.

Because two data items of different semantics can also be visually similar, AF2 cannot really determine whether two data items belong to the same semantic. But the fixed order of the data items in the same data record (*CF2*) can help us remedy this limitation. So we further propose an effective algorithm for data item alignment that utilizes both *CF2* and *AF2*.

5.2.2 Algorithm for data item alignment

CF2 says that the order of data items in data records is fixed. Thus, each data record can be treated as a sequence of data items. We can utilize this feature to align data items. Our goal is to place the data items of the same semantic in the same column. If an optional data item does not appear in a data record, we will fill the vacant position with a predefined blank item. Based on this insight, we propose a multi-alignment algorithm that can process all extracted data records holistically step by step. The basic idea of this algorithm is described as follows. Initially, all the data items are unaligned. We align data items by the order in their corresponding data records. When we encounter optional data items that do not appear in some data records, these vacant positions will be filled with the predefined blank item. This ensures that all data records are aligned and have the same number of data items at the end. Our data item alignment algorithm is shown in Fig. 9.

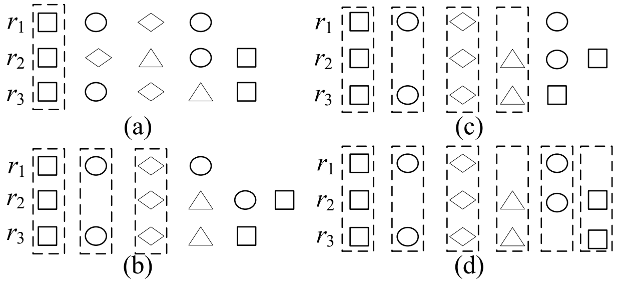


Fig. 10. An example of data item alignment

The input is n data records $\{r_1, r_2, ..., r_n\}$, each data record r_i is denoted as a sequence of data items $\{item_i^1, item_i^2, ..., item_i^m\}$. Any data item has a unique position in its corresponding sequence according to the semantic order. In each iteration, we only process the next unaligned data item of every data record and decide which ones should be ahead of all others. The complexity of this algorithm is $O(n^2 * m)$, where n is the number of data records in the sample page and m is the average number of data items per data record.

Example for data item alignment:

The example shown in Fig. 10 explains the process of data item alignment. Suppose there are three data records $\{r_1, r_2, r_3\}$ and each row is a data record. We use simple geometric shapes (rectangle, circle, triangle, etc.) to denote the data items. The data items represented by the same shape are visually matched data items. We also use $item_i^j$ to denote the *j*th data item of the *i*th data record. Initially (Fig. 10.(a)), all current unaligned data items $\{item_1^1, item_2^1, item_3^1\}$ of the input data records are placed into one cluster, i.e., they are aligned as the first column. Next (Fig. 10.(b)), the current unaligned data items $item_1^2, item_2^2, item_3^2$ are matched into two clusters $C_1 = \{item_1^2, item_3^2\}$ and $C_2 = \{item_2^2\}$ (line 5 in Figure 9). Thus, we need to further decide which cluster should form the next column. The data items in C_1 can match $item_2^4$, and the position value 2 is logged (lines 6-12), which means $item_2^4$ is the third of the unaligned data items of r_2. The data items in C_2 can match $item_1^3$ and $item_3^3$, and the position value 1 is logged (lines 6-12). Because 1 is smaller than 2 (line 16), the data items in C_1 should be ahead of the data items in C_2 and form the next column by inserting the blank item into other records at the current positions (lines 21-22). The remaining data items can be aligned in the same way (Fig. 10.(c) and (d)).

6 VISUAL WRAPPER GENERATION

ViDE has two components, ViDRE (Vison-based Data Record Extracor) and ViDIE (Vison-based Data Item Extracor). There are two problems with them. First, the complex extraction processes are too slow in supporting real-time applications. Second, the extraction processes would fail if there is only one data record on the page. Since all deep web pages from the same web database share the same visual template, once the data records and data items on a deep web page have been extracted, we can use these extracted data records and data items to generate the extraction wrapper for the web database so that new deep web pages from the same web database can be processed using the wrappers quickly without reapplying the entire extraction process.

Our wrappers include data record wrapper and data item wrapper. They are the programs that do data record extraction and data item extraction with a set of parameter obtained from sample pages. For each web database, we use a normal deep web page containing the maximum number of data records to generate the wrappers. The wrappers of previous works mainly depend on the structures or the locations of the data records and data items in the tag tree, such as tag path. In contrast, we mainly use the visual information to generate our wrappers. Note that some other kinds of information are also utilized to enhance the performances of the wrappers, such as the data types of the data items and the frequent symbols appearing in the data items. But they are easy to obtain from the web pages. We describe the basic ideas of our wrappers below.

6.1 Vision-based Data Record Wrapper

Given a deep web page, vision-based data record wrapper first locates the data region in the Visual Block tree, and then extracts the data records from the child blocks of the data region.

Data region location: After the data region *R* on a sample deep web page *P* from site *S* is located by ViDRE, we save five parameters values (*x, y, w, h, l*), where (*x, y*) form the coordinate of *R* on *P*, *w* and *h* are the width and height of *R*, and *l* is the level of *R* in the Visual Block tree.

Given a new deep web page *P** from *S*, we first check the blocks at level *l* in the Visual Block tree for *P**. The data region on *P** should be the block with the largest area overlap with *R* on *P**. The overlap area can be computed using the coordinates and width/height information.

Data record extraction: For each record, our visual data record wrapper aims to find the first block of each record and the last block of the last data record (denoted b_{last}).

To achieve this goal, we save the visual information (the same as the information used in Formula 1) of the first block of each data record extracted from the sample page and the distance (denoted as *d*) between two data records. For the child blocks of the data region in a new page, we find the first block of each data record by the visual similarity with the saved visual information. Next, b_{last} on the new page needs to be located. Based on our observation, in order to help the users differentiate data records easily, the vertical distance between any two neighboring blocks in one data record is always smaller than *d* and the vertical distance between b_{last} and its next block is not smaller than *d*. Therefore, we recognize the first block whose distance with its next block is larger than *d* as b_{last}.

6.2 Vision-based Data Item Wrapper

Table 6
Explanation for (*f*, *l*, *d*)

Parameter	Value	Remarks
f	font	the font used by the data items of this attribute
l	Boolean	*True* denotes that the data items of this attribute are link texts
d	image, text, number, date, email, etc	the data type of this attribute

The data alignment algorithm groups data items from different data records into columns or attributes such that data items under the same column have the same semantic. Table 6 lists useful information about each attribute obtained from the sample page that can help determine which attribute a data item belongs to.

The basic idea of our vision-based data item wrapper is described as follows. Given a sequence of attributes $\{a_1, a_2, ..., a_n\}$ obtained from the sample page and a sequence of data items $\{item_1, item_2, ..., item_m\}$ obtained from a new data record, the wrapper processes the data items in order to decide which attribute the current data item can be matched to. For $item_i$ and a_j, if they are the same on *f*, *l*, and *d*, their match is recognized. The wrapper then judges whether $item_{i+1}$ and a_{j+1} are matched next, and if not, it judges $item_i$ and a_{j+1}. Repeat this process until all data items are matched to their right attributes.

Note that if an attribute on a new page did not appear on the sample page, the data item of the attribute cannot be matched to any attribute. To avoid such a problem, several sample pages may be used to generate the wrapper. This can increase the chance that every attribute appears on at least one of these sample pages.

This process is much faster than the process of wrapper generation. The complexity of data records extraction with the wrapper is $O(n)$, where n is the number of data records in the page. The complexity of data items extraction with the wrapper is $O(n * m)$, where n is the number of data records in the test page and m is the average number of data items per data record.

7 EXPERIMENTS

We have implemented an operational deep web data extraction system for ViDE based on the techniques we proposed. Our experiments are done on a Pentium 4 1.9GH, 512MB PC. In this section we first describe the data sets used in our experiments, and then introduce the performance measures used. At last, we evaluate both ViDRE and ViDIE. We also choose MDR [17] and DEPTA [29] to compare with ViDRE and ViDIE respectively. MDR and DEPTA are the recent works on web data record extraction and data item extraction, and they are both HTML-based approaches.

7.1 Datasets

Most performance studies of previous works used small data sets, which are inadequate in assuring the impartiality of the experimental results. In our work, we use a large dataset to carry out the experiments.

General Dataset (GDS): This dataset is collected from CompletePlanet (www.completeplanet.com), which is currently the largest deep web repository with more than 70,000 entries of web databases. These web databases are classified into 42 categories covering most domains in the real world. GDS contains 1,000 available web databases. For each web database, we submit 5 queries and gather 5 deep web pages with each containing at least three data records.

Special Dataset (SDS): During the process of obtaining GDS, we noticed that the data records from two-thirds of the web databases have less than 5 data items on the average. To test the robustness of our approaches, we select 100 web databases whose data records contain more than 10 data items from GDS as SDS.

Note that, the deep web pages collected in the testbed are the ones that can be correctly displayed by the Web browser we used. An example where a page is not correctly displayed is when some images are displayed as small red crosses. This will cause the positions of the texts on the result page to shift.

7.2 Performance Measures

All previous works use *precision* and *recall* to evaluate their experimental results (some also include F-measure, which is the weighted harmonic mean of *precision* and *recall*). These measures are also used in our evaluation.

Table 7
Performance Measures Used in the Evaluation of ViDE

	precision	*recall*	*revision*
ViDRE	$\frac{DR_c}{DR_e}$	$\frac{DR_c}{DR_r}$	$\frac{WDB_t - WDB_c}{WDB_t}$
ViDIE	$\frac{DI_c}{DI_e}$	$\frac{DI_c}{DI_r}$	

In this paper we propose a new metric, *revision*, to measure the performance of an automated extraction algorithm. It is defined to be the percentage of the web databases whose data records or data items are not perfectly extracted, i.e., either *precision* or *recall* is not 100%. This measure indicates the percentage of web databases the automated solution fails to achieve perfect extraction and manual revision of the solution is needed to fix this. An example is used to illustrate the significance of this measure. Suppose there are three approaches (A1, A2 and A3) which can extract structured data records from deep web pages, and they use the same data set (5 web databases, 10 data records in each web database). A1 extracts 9 records for each site and they are all correct. So the average *precision* and *recall* of A1 are 100% and 90%, respectively. A2 extracts 11 records for each site and 10 are correct. So the average *precision* and *recall* of A2 are 90.9% and 100%, respectively. A3 extracts 10 records for 4 of the 5 databases and they are all correct. For the 5th site, A3 extracts no records. So the average *precision* and *recall* of A3 are both 80%. Based on average *precision* and *recall*, A1 and A2 are better than A3. But in real applications A3 may be the best choice. To make *precision* and *recall* 100%, all wrappers generated by A1 and A2 have to be manually tuned/adjusted, while only one wrapper generated by A3 needs to be manually tuned. In other words, A3 needs the minimum manual intervention.

Because our experiments include data record extraction and data item extraction, we define *precision*, *recall* and *revision* for them separately.

In Table 7, DR_c is the total number of correctly extracted data records, DR_r is the total number of data records, and DR_e is the total number of data records extracted, DI_c is the total number of correctly extracted data items, DI_r is the total number of data items, and DI_e is the total number of data items extracted; WDB_c is the total number of web databases whose *precision* and *recall* are both 100%, and WDB_t is the total number of web databases processed.

7.3 Experimental Results on ViDRE

In this part, we evaluate ViDRE and also compare it with MDR. MDR has a similarity threshold, which is set at the default value (60%) in our test, based on the suggestion of the authors of MDR. Our ViDRE also has a similarity threshold, which is set at 0.8. In this experiment, the input to ViDRE and MDR contains the deep web pages and the output contains data records extracted. For ViDRE, one sample result page containing the most data records is used to generate the data record wrapper for each web database. Table 8 shows the experimental results on both GDS and SDS. Based on our experiment, it takes approximately 1 second to generate the data record wrapper for each page and less than half second to use the wrapper for data record extraction.

From Table 8, we can make the following three observations. First, ViDRE performs significantly better than MDR on both GDS and SDS. Second, ViDRE is far better

Table 8
Comparison Results between ViDRE and MDR

	dataset	*precision*	*recall*	*revision*
ViDRE	GDS	98.7%	97.2%	12.4%
	SDS	98.5%	97.8%	10.9%
MDR	GDS	85.3%	53.2%	55.2%
	SDS	78.7%	47.3%	63.8%

than MDR on revision. ViDRE needs only to revise slightly over 10% of the wrappers, while MDR has to revise almost five times more wrappers than ViDRE. Third, the precision and recall of ViDRE are steady on both SDS and GDS, but for MDR, they drop noticeably for SDS. Our analysis indicates that: for *precision* of ViDRE, most errors are caused by failing to exclude noise blocks that are very similar to the correct ones in appearance; for *recall* of ViDRE, most errors are caused by mistaking some top or bottom data records as the noise blocks; for MDR, its performance is inversely proportional to the complexity of the data records, especially data records with many optional data items.

7.4 Experimental Results on ViDIE

In this part, we evaluate ViDIE and compare it with DEPTA. DEPTA can be considered as the follow-up work for MDR, and its authors also called it MDRII. Only correct data records from ViDRE are used to evaluate ViDIE and DEPTA. For ViDIE, two sample result pages are used to generate the data item wrapper for each web database. Table 9 shows the experimental results of ViDIE and DEPTA on both GDS and SDS. Our experiments indicate that it takes between 0.5 and 1.5 seconds to generate the data item wrapper for each page and less than half second to use the wrapper for data item extraction..

Table 9
Comparison Results between ViDIE and DEPTA

	dataset	*precision*	*recall*	*revision*
ViDIE	GDS	96.3%	97.2%	14.1%
	SDS	95.6%	98.4%	11.6%
DEPTA	GDS	75.3%	71.6%	32.8%
	SDS	66.1%	54.1%	37.6%

From Table 9, we can see that the observations we made in comparing the results of ViDRE and MDR remain basically valid for comparing ViDIE and DEPTA. In addition, we also found that DEPTA often misaligns two data items of different semantics if they are close in the tag tree and have the same tag path, and this leads to the misalignment of all the data items in the same data record that follow the misaligned data items. In contrast, ViDIE can easily distinguish them due to their different fonts or positions.

We also tried to use one sample page and three sample pages to generate the data item wrapper for each web database. When one page is used, the performance is much lower; for example, for SDS, the precision, recall and revision are 91.7%, 95% and 32.3%, respectively. This is caused by the absence of some optional data items from all the data records in the sample page used. When more sample pages are used, the likelihood that this will happen is significantly reduced. When three pages are used, the results are essentially the same as those shown in Table 9, where two sample pages are used. This suggests that using two sample pages to generate the data item wrapper for each web database is sufficient.

We also conducted experiments based on the datasets used in [30] and provided by [13], and the results are similar to those shown in Table 8 and 9. These results are not shown in this paper due to space consideration.

8 CONCLUSIONS AND FUTURE WORKS

With the flourish of the deep web, users have a great opportunity to benefit from such abundant information in it. In general, the desired information is embedded in the deep web pages in the form of data records returned by web databases when they respond to users' queries. Therefore, it is an important task to extract the structured data from the deep web pages for later processing. In this paper, we focused on the structured web data extraction problem, including data record extraction and data item extraction. Firstly, we surveyed previous works on web data extraction and investigated their inherent limitations. Meanwhile, we found that the visual information of web pages can help us implement web data extraction. Based on our observations of a large number of deep web pages, we identified a set of interesting common visual features that are useful for deep web data extraction. Based on these visual features, we proposed a novel vision-based approach to extract structured data from deep web pages, which can avoid the limitations of previous works. The main trait of this vision-based approach is that it primarily utilizes the visual features of deep web pages.

Our approach consists of four primary steps: Visual Block tree building, data record extraction, data item extraction, and visual wrapper generation. Visual Block tree building is to build the Visual Block tree for a given sample deep page using the VIPS algorithm. With the Visual Block tree, data record extraction and data item extraction are carried out based on our proposed visual features. Visual wrapper generation is to generate the wrappers that can improve the efficiency of both data record extraction and data item extraction. Highly accurate experimental results provide strong evidence that rich visual features on deep web pages can be used as the basis to design highly effective data extraction algorithms.

However, there are still some remaining issues and we plan to address them in the future. First, ViDE can only process deep web pages containing one data region while there is significant number of multi-data-region deep web pages. Though [31] has attempted to address this problem, their solution is HTML dependent and its performance has a large room for improvement. We intend to propose a vision-based approach to tackle this problem. Second, the efficiency of ViDE can be improved. In the current ViDE the visual information of web pages is obtained by calling the programming APIs of IE, which is a time-consuming process. To address this problem, we intend to develop a set of new APIs to obtain the visual information directly from the web pages.

ACKNOWLEDGMENTS

This work is supported in part by the NSF of China under grant #60833005, 60573091; China 863 High-Tech Program under grant #2007AA01Z155; China 973 plan under grant #2003CB317000; US NSF grants IIS-0414981 and CNS-0454298. The authors would also like to express their gratitude to the anonymous reviewers for providing some very helpful suggestions.

REFERENCES

[1] G. O. Arocena, A. O. Mendelzon: WebOQL: Restructuring Documents, Databases, and Webs. ICDE 1998: 24-33
[2] D. Buttler, L. Liu, C. Pu: A Fully Automated Object Extraction System for the World Wide Web. ICDCS 2001: 361-370
[3] D. Cai, X. He, J.-R. Wen, W.-Y. Ma: Block-Level Link Analysis. SIGIR 2004: 440-447
[4] D. Cai, S. Yu, J. Wen, W. Ma: Extracting Content Structure for Web Pages Based on Visual Representation. APWeb 2003: 406-417
[5] C.-H. Chang, M. Kayed, M. R. Girgis, K. F. Shaalan: A Survey of Web Information Extraction Systems. IEEE Trans. Knowl. Data Eng. 18(10): 1411-1428 (2006)
[6] C.-H. Chang, C.-N. Hsu, S.-C. Lui: Automatic Information Extraction from Semi-Structured Web pages by Pattern Discovery. Decision Support Systems 35(1): 129-147 (2003)
[7] V. Crescenzi, G. Mecca: Grammars Have Exceptions. Inf. Syst. 23(8): 539-565 (1998)
[8] V. Crescenzi, G. Mecca, P. Merialdo: RoadRunner: Towards Automatic Data Extraction from Large Web Sites. VLDB 2001: 109-118
[9] D. W. Embley, Y. S. Jiang, Y.-K. Ng: Record-Boundary Discovery in Web Documents. SIGMOD Conference 1999: 467-478
[10] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krpl, B. Pollak: Towards Domain-Independent Information Extraction from Web Tables. WWW 2007: 71-80
[11] J. Hammer, J. McHugh, H. Garcia-Molina: Semistructured Data: The TSIMMIS Experience. ADBIS 1997: 1-8
[12] C.-N. Hsu, M.-T. Dung: Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. Inf. Syst. 23(8): 521-538 (1998)
[13] http://daisen.cc.kyushu-u.ac.jp/TBDW/
[14] http://www.w3.org/html/wg/html5/
[15] N. Kushmerick: Wrapper induction: Efficiency and Expressiveness. Artif. Intell. 118(1-2): 15-68 (2000)
[16] A. Laender, B. Ribeiro-Neto, A.da Silva, J.Teixeira: A Brief Survey of Web Data Extraction Tools. SIGMOD Record 31(2): 84-93 (2002)
[17] B. Liu, R. L. Grossman, Y. Zhai: Mining Data Records in Web Pages. KDD 2003: 601-606
[18] W. Liu, X. Meng, W. Meng. Vision-based Web Data Records Extraction. WebDB 2006, pp.20-25, June 2006.
[19] L. Liu, C. Pu, W. Han: XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. ICDE 2000: 611-621
[20] Y. Lu, H. He, H. Zhao, W. Meng, C. T. Yu: Annotating Structured Data of the Deep Web. ICDE 2007: 376-385
[21] J. Madhavan, S. R. Jeffery, S. Cohen, X. L. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale Data Integration: You Can Only Afford to Pay As You Go. CIDR 2007: 342-350
[22] I. Muslea, S. Minton, C. A. Knoblock: Hierarchical Wrapper Induction for Semi-structured Information Sources. Autonomous Agents and Multi-Agent Systems 4(1/2): 93-114 (2001)
[23] Z. Nie, J.-R. Wen, W.-Y. Ma: Object-level Vertical Search. CIDR 2007: 235-246
[24] A. Sahuguet, F. Azavant: Building Intelligent Web Applications Using Lightweight Wrappers. Data Knowl. Eng. 36(3): 283-316 (2001)
[25] K. Simon, G. Lausen: ViPER: Augmenting Automatic Information Extraction with Visual Perceptions. CIKM 2005: 381-388
[26] R. Song, H. Liu, J.-R. Wen, W.-Y. Ma: Learning block importance models for web pages. WWW 2004: 203-211
[27] J. Wang, F. H. Lochovsky: Data Extraction and Label Assignment for Web Databases. WWW 2003: 187-196
[28] X. Xie, G. Miao, R. Song, J.-R. Wen, W.-Y. Ma: Efficient Browsing of Web Search Results on Mobile Devices Based on Block Importance Model. PerCom 2005: 17-26
[29] Y. Zhai, B. Liu: Web Data Extraction Based on Partial Tree Alignment. WWW 2005: 76-85
[30] H. Zhao, W. Meng, Z. Wu, V. Raghavan, C. T. Yu: Fully Automatic Wrapper Generation for Search Engines. WWW 2005: 66-75
[31] H. Zhao, W. Meng, C. T. Yu: Automatic Extraction of Dynamic Record Sections from Search Engine Result Pages. VLDB 2006: 989-1000
[32] J. Zhu, Z. Nie, J. Wen, B. Zhang, W. Ma: Simultaneous Record Detection and Attribute Labeling in Web Data Extraction. KDD 2006: 494-503

Wei Liu received the BS and MS degrees in computer science from Shandong University in 1998 and 2004 respectively, and the PhD degree in computer science from Renmin University of China in 2008. From 2008 to now, he is a post-doctoral fellow in computer science at Institute of Computer Science and Technology of Peking University. His research interests include Web data extraction and Deep Web data integration.

Xiaofeng Meng received the B.S. degree from Hebei University, M.S. degree from Remin University of China , Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, all in computer science. He is currently a professor of School of Information in Renmin University of China. His research interests include web data integration, native XML databases, mobile data management, flash based databases.He is the Secretary General of Database Society of the China Computer Federation (CCF DBS). He has published more than 100 technical papers.

Weiyi Meng received the BS degree in mathematics from Sichuan University, China, in 1982, and the MS and PhD degrees in computer science from the University of Illinois at Chicago in 1988 and 1992, respectively. He is currently a professor in the Department of Computer Science at the State University of New York at Binghamton. His research interests include Web-based information retrieval, metasearch engines, and Web database integration. He is the coauthor of a database book Principles of Database Query Processing for Advanced Applications. He has published more than 100 technical papers.