Abstract—Web databases contain a huge amount of structured data which are easily obtained via their query interfaces only. Query results are presented in dynamically generated web pages, usually in the form of data records, for human use. Decisive for web data integration applications is the problem of automatically extracting data records from query result pages, such as comparison shopping sites, meta-search engines, etc. A number of approaches to query result extraction have been proposed. As the structures of web pages become more critical, these approaches start to fail. Query result pages usually also contain other types of information in addition to query results, e.g., advertisements, navigation bar, etc. Most of the existing approaches do not move out such impertinent contents which may affect the accuracy of data record extraction. We have observed that query results are usually displayed in regular visual patterns and terms used in a query often reappear in query results. The paper proposes a novel approach that makes use of visual features and query terms to identify the data section and extract data records from it. This also uses several content and visual features of visual blocks in a data section to filter out noisy blocks. The results of this experiment tests on a large set of query result pages in different domains show that the proposed approach is highly effective.

I. INTRODUCTION

The bulk of structured data on the Web has been increasing enormously. Such data are usually returned from back-end databases in response to specific user queries, and presented in the form of data records in query result pages. Access to web databases is via their query interfaces (usually HTML query forms) only. In literature, the contents of web databases are usually referred to as the Deep Web. A recent study [20] estimates that the number of web databases that are 'hidden' on the Web is well in the order of $10^5$ and continues expanding rapidly. Many e-commerce sites are supported by web databases.

A Suresh Babu¹, Department of Computer Science and Engineering, JNTUA Pulivendula, India. Email:sureshjntu@gmail.com
Dr. P. Premchand², Professor, Department of Computer Science Engineering, Osmania University, Hyderabad, India.
Email:p.prechand@uceou.edu
Dr. A. Govardhan³, Professor, Department of Computer Science Engineering, Director of evaluationJNTUH, Kukatpalli, Hyderabad, India. Email:govardhan_cse@yahoo.co.in

In general, the majority of query result pages are list pages, each of which contains a number of data records in columns with each row on each column representing a data record. For example, Figure 1 shows a list page from cooking.com, which has a single column containing 10 data records about plates.

Fig. 1. An example query result page

Fig. 2. The visual block layout of the query result page shown in Fig. 1
Extracting data records from query result pages enables integrating data from a magnitude of web databases to generate value-added web applications, such as price comparison sites and meta-search engines, etc. Query result pages are dynamically generated from back-end databases in response to user queries and encoded in HTML using pre-defined templates or script programs. These pages are semi-structured and displayed for human use, rather than for processing by programs. How to automatically extract data records into a structured form that is machine processable is a very challenging problem.

There has been a lot of research on fully-automatic approaches [3-16] for extracting data from query result pages. Those in [3-10] represent the current technical trend of query result extraction. First, they identify a data section, which contains a set of data records. Second, they identify data records from the data section. Finally, they extract data by aligning the corresponding attributes of different records, producing a relational table [4, 5, 8, 10].

However, the existing approaches to query result extraction have some inherent limitations. First, web pages are becoming more complex; their tag structures are ever-growing complex since HTML itself is evolving constantly, and other technologies like JavaScript and CSS are widely deployed to make result pages more dynamic. This may make the layouts of result pages different from their tag tree or token string representations, and thus the existing approaches that rely on such representations may fail. Second, some of the existing approaches employ a similarity measure on page segments to identify data records. However, data records may not be extracted correctly if the sibling tree segments of the same root are not similar to each other. This also makes it impossible to extract a single data record in the data section. Third, most of the existing approaches do not filter out noisy contents. Noisy contents refer to any parts of a query result page that are not part of any data record, e.g. banner advertisements, navigation bar, copyright notice, record statistical information etc. We are most interested in the part of a result page which contains all the data records with few noisy contents which often affect the accuracy of data record extraction. Thus it is very important to remove any noisy contents before data record extraction. In this paper, it is focused on the problem of data record extraction, that is, given a query result page that contains a single column of data records, automatically identify the data section and data records. To overcome the limitations of the existing approaches we propose a novel approach. First, this approach transforms a query result page into a Visual Block tree using the VIPS algorithm [17], which represents a visual partition of the web page.

Such a representation reflects the content structure of the page enforced by visual cues so that content related data items are represented in the same branch of the Visual Block tree. For example, Figure 2 shows a visual partition of the result page shown in Figure 1; Figure 3 shows part of the visual block tree for visual block $b_{1-1}$. We can also get visual features (e.g., positions, width, height etc) of each block on the Visual Block tree. Second, this approach identifies the data section by exploiting the sizes of the visual blocks of the result page, and counting the occurrences of query terms in them. This is based on the following two observations: the data section in a result page, which contains all the data records, usually occupies a significant area of the result page; query terms often reappear in the data records. We use these two observations to identify the data section. For example, block $b_{1-1,-2,-2,3,4}$ in Figure 2 is likely to the data section since it occupies a large portion of the result page in Figure 1 and contains a number of occurrences of query terms (e.g., "Accent Plates"). To get query terms, we make use of the query interface and assume that the result pages are generated in response to the queries made via the interface.

Fig. 3. Part of the Visual Block tree

The identified data section often contains noisy blocks. Data records are obviously more vivid in content than noisy blocks, have one or more links or some images. To filter out noisy blocks, we use a vector of content and visual features to characterize each block within the data section. These features provide statistical information about texts, block area, links and images in the block. The overall importance of a block for a data record should be higher than noisy blocks. We set up a threshold of importance to ensure that any blocks that have less importance than the threshold are identified as noisy blocks and removed. For example, as shown in Figure 1 there are ten data record blocks while the block containing information about the data records ("Items (1 - 15) of 15") is identified as a noisy block and removed.
Third, we observe that each data record contains semantically related data units of a data object, which reside in the leaf nodes of the Visual Block trees, and are visually aligned with and adjacent to each other. This approach identifies data records by purely using the rendering boxes of the leaf nodes in the data section to infer their alignment and proximity. For example, the data units of each data record shown in Figure 1 are aligned with each other, in close proximity and relatively far away from the data units of the other data records. Thus we can group data units based on their positional information with each group representing a data record. This makes the following contributions. First, the paper proposes an approach for identifying data sections based on the visual features of the blocks and recurrences of query terms in them. Based on the content and visual features of visual blocks, this approach for removing noisy blocks can eliminate most of the noisy blocks. Second, it proposes an approach for identifying data records based on an observation that the data units of a data record are visually aligned with and close to each other, and that they are distant from the data units of the other data records. By grouping data units in such a way, this approach does not miss any data record that is not similar to the other data records, and this approach can extract a single data record from a query result page.

II. RELATED WORK

Automatic extraction of web query results has attracted a lot of attention over the recent years. Several automatic extraction systems have been developed. Earlier works mainly focus on finding repetitive patterns and templates in result pages, e.g., IEPAD [13], RoadRunner [12], DeLa [14] and EXALG [15]. Recent techniques have focused on exploiting tag structures and visual features, e.g., MDR [3], DEPTA [4, 5], MSE [7], ViNTs [6], VIPE [8], ViDE [16] and [9].

The works that use visual features include ViPER [8], ViNTs [6], MSE [7] and ViDE [16]. ViDE, is the most related to this approach. It is the first work that is primarily based on visual features. There are several main differences between ViDE and this approach. ViDE first clusters data units of the same semantics based on similarity between their appearances, and then groups appropriate data units from each of the clusters into data records. This approach uses a proximity based technique to directly group data units in the same data records. ViDE may cluster data units with different semantics because sometimes neighboring data units in the same data record may not have distinguishable appearances, resulting in them being clustered together and then grouped into different data records. Second, ViDE uses the positions and sizes of visual blocks to determine if a block is the data section.

If multiple blocks are identified as candidate data sections, it chooses the one with smallest size as the data section. This approach counts the occurrences of query terms in candidate blocks to select the real data section that makes this approach more robust. Third, ViDE identifies noisy blocks by deciding whether the blocks are aligned to the left of a data section but it may not remove all the noisy blocks. This approach evaluates the importance of blocks within the section based on content and visual features which improve the effect of removing noisy blocks.

This approach instead extracts data records from query result pages that have more complex content structures. Though this approach also uses the alignment and adjacency techniques, this alignment definition is much simpler than the one in [1, 2]. This approach uses also query terms in the process of grouping data units.

III. FUNDAMENTALS AND OVERVIEW

In this section, Visual Block trees are introduced and give a formal definition of the rendering box model of web pages is introduced based on the Visual Block tree, which is the basis of this approach. We then define the problem of data record extraction and present an overview of this approach.

A. Visual Representation of Query Result Pages

The content of a query result page is typically organized into different regions to make it easy for human use, e.g., advertisements, menu bar, sponsor links, query results and so on. Each region contains semantically related content. Visual cues (e.g., lines, spaces, font sizes, background colors etc) can be used to distinguish regions from each other. To make use of visual features for data record extraction, we employ the VIPS [17] algorithm to represent a query result page as a Visual Block tree. The root of the tree represents the entire page and each node represents a rendering box (a visual block) on the page. A leaf node represents a block containing a basic semantic unit that cannot be further decomposed, e.g., a text or image. Node a is an ancestor of node b if the block that a represents contains the block that b represents on the page. The blocks represented by nodes at the same level of the tree do not overlap. The order of the child nodes with the same parent follows the order of the blocks they represent on the page, i.e., top-down, left-right. For example, suppose Figure 2 shows the visual block layout produced by the VIPS algorithm for the query result page shown in Figure 1. For example, b1,2 represents the body of the page, b1,1-2,1 represents the block containing the category links on the page, b1,2 contains the website information and b1,1-2,2,2,4 contains all data records denoted as b1,1-2,3,4-1 to b1,1-2,3,4-10. Figure 3 shows part of the Visual Block tree for b1,1-2,3,4-2.
IV. IDENTIFYING DATA SECTIONS

Data section identified as a node in the visual block tree, which represents a rectangular box in the result page that contains all the data record blocks and as few noisy blocks as possible. We observe that the size of a data section is usually large relative to the size of the whole page. For example, as shown in Figure 1, the data section that contains all the plate products occupies a relatively large area. To utilize the observation, we first select those blocks, each of which satisfies a constraint that the ratio between the sizes of the block and the whole page is greater than a threshold $T_{dr}$ ([16]), which can be trained from sample result pages. The method for identifying data sections first takes the root node of the Visual Block tree as input. It returns a set of candidate data section blocks. The blocks at higher levels of the Visual Block tree occupy bigger portions of the result page so that their area ratios are much higher than the threshold and will certainly contain more noisy blocks than the ones at lower levels of the Visual Block tree. The algorithm selects candidate data section blocks in a depth-first fashion. It traverses the Visual Block tree from the root, and identifies those blocks that satisfy the area ratio constraint but none of their child blocks changes it that. These blocks thus contain less noisy blocks. For example, after applying the area constraint, we can identify $b_{1,1}$, $b_{1,1-2,3}$, $b_{1,1-2,2,2,2}$, and $b_{1,1-2,2,2,3}$ as candidate data sections.

Candidate data sections are further considered to determine the real data section. To do this we make use of query terms that are used in queries over query interfaces. A query interface exposes the attributes of the web database schema to the user and usually consists of a set of input elements, e.g., text boxes, radio buttons, check boxes and selection lists. Each input element is associated with an attribute ([18]). For example "Dinnerware" "Plates" "Royal Doulton" and "$25 to $50" are query terms used for input elements associated with attributes "Category" "Brand" and "Price" of the query interface, as shown in Figure 4. We observe that query terms often re-appear in the data records. For example, the data records shown in Figure 1 are in response to the query shown in Figure 4. We can see that the text nodes of each data record contain the occurrences of query terms "Plates" and "Royal Doulton". The frequency of each query term in a candidate block reflects the importance of the candidate block. The more query terms occur in a block, the more likely the block is the data section. Given a set of query terms $q_i$ for $i = 1, 2, ..., n$, and a candidate block, the importance of the block is measured as

$$R = \sum_{i=1}^{n} f_i$$  \hspace{1cm} (1)

where $f_i$ represents the frequency of query term $i$ in the candidate block. The block that has the maximum number of occurrences of query terms among all the candidate blocks is identified as the data section. For example, after applying the second constraint to the candidate data sections, $b_{1,1-2,2,3}$ as shown in Figure 3, is identified as the data section.

V. REMOVING NOISY BLOCKS

The identified data section usually contains noisy blocks on the top and bottom of the section, and data records in the middle of the section with no noisy blocks on either the left or right of the records. Noisy blocks are the ones that are in a data section but are not part of any data record [16], such as data record numbers (e.g., "Items (1-15) of 15") in Figure 1. We observe that a data record typically contains images, description of data, links, and occupies a significant area on the page. For example, each of the data records shown in Figure 1 contains the image, name, and model etc. of the product, one or more links for detailed information about a specific model and the rectangle of each data record is very noticeable. Specifically, we evaluate the importance of each first-level child block within the data section by using the five features about the content of the block: $ImgNum$ (the number of images in the block), $LinkNum$ (the number of links in the block), $TextLen$ (the anchor text length of the block), $ExtLen$ (the text length of the block), and $Area$ (the rendering area of the block). These content features are provided by the Visual Block tree and are normalized across the whole data section. The importance of a child block is defined as

$$ImBlk = w_1 \times ImgN um + w_2 \times LinkN um + w_3 \times LinkT extLen + w_4 \times T extLen + w_5 \times Area$$ \hspace{1cm} (2)

where $w_1$, $w_2$, $w_3$, $w_4$ and $w_5$ are real numbers so that $w_1 + w_2 + w_3 + w_4 + w_5 = 1$, and $0 \leq ImBlks \leq 1$.

$LinkTextLen$ and $TextLen$ are considered as the most important features for differentiating data record blocks from noisy blocks. When the $ImBlk$ of a block is greater than the given threshold $\theta$, it is very likely that the block is a data record. Otherwise the block is taken as a noisy block. The threshold can be trained using sample pages.

VI. GROUPING DATA UNITS OF DATA RECORDS

A data record represents a data object retrieved from a web database and consists of multiple data units that are semantically related. Data units are represented as leaf nodes on the Visual Block tree, and they are visually aligned with & adjacent to each other on result pages.
For example, as shown in Figure 1, the data units of each record are the leaf nodes in the Visual Block tree, and they are visually aligned with and adjacent to each other on the web page. To identify data records, this approach first identifies leaf nodes that are part of a data record and can be used as starting points for grouping other data units of the record. Given a starting point, it first group data unit that are horizontally aligned with it to form a data unit group based on the positions of the visual blocks of the corresponding leaf nodes. It then groups data units that are horizontally aligned with each other to form leaf node groups. Finally, this approach progressively expands each data unit group with other data unit groups and leaf node groups that are vertically adjacent to it until there is no vertically adjacent group. Each data unit group thus corresponds to a data record.

**Definition 1.** (Block and group positions) - We use the coordinate of the top-left corner, height and width of the visual block of a data unit to determine its left, right, top and bottom positions. Furthermore, we use the left position of the leftmost node of a node group as the left position of the group, the top position of the topmost node as the top position of the group, the right position of the rightmost node of a node group as the right position of the group, and the bottom position of the bottom node as the bottom of the group.

**Definition 2.** (Horizontal alignment) - We say that two leaf nodes, a and b, are horizontally aligned with each other, if they have similar top positions. Furthermore, we say that two node groups, a and b, are horizontally aligned with each other, if they have similar top positions.

**Definition 3.** (Vertical adjacency) - We say that two leaf nodes, a and b, are vertically adjacent, if the distance between the bottom position of a and the top position of b, or the distance between the top position of a and the bottom position of b (vertical distance) is less than a given number of pixels (in close proximity). Furthermore, we say that two node groups, a and b, are vertically adjacent, if the shortest vertical distance between the nodes in a and b is less than a given number of pixels (in close proximity), and the nodes in the two groups are on the same sub tree.

**Algorithm 1.** Grouping Data Units

**Input:** a set of query terms \( T \), a data section block \( B \)

**Output:** a set of data records \( R \)

1: Set \( R \), a set of leaf nodes \( N_l \), a set of starting leaf nodes \( N_s \), a set of data unit groups \( G \), a set of leaf node groups \( G_l \), and a set of horizontally expanded data

unit groups \( G \) all to \{\}

2: Add every text node in \( B \) to \( N_l \)

3: for every leaf node \( n_l \in N_l \) do

4: if \( n_l \) contains a query term \( t \in T \) then

5: Add \( n_l \) to \( N_s \)

6: Remove \( n_l \) from \( N_l \)

7: for every starting leaf node \( n_s \in N_s \) do

8: Set a data unit group \( g \) to \{\( n_s \)\}

9: for every leaf node \( n_l \in N_l \) do

10: if \( n_l \) is horizontally aligned with \( n_s \) then

11: Add \( n_l \) to \( g \)

12: Remove \( n_l \) from \( N_l \)

13: Add \( g \) to \( G \)

14: repeat

15: Remove a leaf node \( n_l \) from \( N_l \)

16: Set a leaf node group \( g_l \) = \{\( n_l \)\}

17: for each leaf node \( n_l \in N_l \) do

18: if \( n_l \) is horizontally aligned with \( n_s \) then

19: Add \( n_l \) to \( G_l \)

20: Remove \( n_l \) from \( N_l \)

21: Add \( g_l \) to \( G_l \)

22: until \( N_l = \{\} \)

23: repeat

24: Remove a data unit group \( g \) from \( G \)

25: for each data unit group \( g \in G \) do

26: if \( g \) is horizontally aligned with \( g \) then

27: Set \( g \) to \( g \cup g \)

28: Remove \( g \) from \( G \)

29: Add \( g \) to \( G \)

30: until \( G = \{\} \)

31: repeat

32: Remove a horizontally expanded data unit group \( g \in G \)

33: for each horizontally expanded data unit group \( g \in G \) do

34: if \( g \) is vertically adjacent to \( g \) then

35: Set \( g \) to \( g \cup g \)

36: Remove \( g \) from \( G \)

37: for each leaf node group \( g_l \in G_l \) do

38: if \( g_l \) is vertically adjacent to \( g \) then

39: Set \( g \) to \( g \cup g_l \)

40: Remove \( g_l \) from \( G_l \)

41: Add \( g \) to \( R \)

42: until \( G = \{\} \)

43: Return \( R \)

The algorithm, as shown in Algorithm 1, takes as input a set of query terms (denoted as \( T \)) and a data section block (denoted as \( B \)), and identifies as output a set of data records (denoted as \( R \)). The algorithm first identifies starting leaf nodes (lines 2-6) by matching each query term with each text node in the Visual Block tree of the data section. Second, the algorithm forms a data unit group with each starting leaf node and tries to expand it with leaf nodes that are horizontally aligned with it (lines 7-13).
Third, the algorithm groups leaf nodes that are not included in data unit groups, and horizontally aligned with each other, into leaf node groups $G_l$ (lines 14-22). Fourth, the algorithm expands each data unit group with the other data unit group that is horizontally aligned with it (lines 23-30). Fifth, the algorithm expands each data unit group with other data unit group and leaf node groups that are vertically adjacent to it (lines 31-42). The algorithm ends when there is no more leaf node group or data unit group that is vertically adjacent to the expanding data unit group.

To illustrate how the algorithm works, we take the first two data records shown in Figure 1 as an example. "Plates" has been used as a query term. Two leaf nodes representing text "Dinner Plates by" are identified as starting leaf nodes. These two starting leaf nodes are used to initiate two data unit groups. Those leaf nodes representing the second rows of these two data records form two leaf node groups. Each data unit group is expanded with a leaf node group which is vertically adjacent to it. Each extracted data unit group represents a data record.

VII. EXPERIMENTAL RESULTS

This experiment tests on a data set of 200 query result pages that are returned from 20 web databases in the UIUC Web Integration Repository [19]. These web databases are from 5 domains - Books, Jobs, Movies, Music and Hotels. 15 of these pages contain a single data record. For each web database, 10 result pages are collected after manually submitting 10 different queries via its query interface. We use two common measures, recall and precision, to evaluate the performance of this approach. Recall is the percentage of the number of data records that have been correctly extracted over the total number of data records on a result page. Precision is the percentage of the number of data records that have been correctly extracted over the total number of data records that have been extracted.

We compare this approach with ViDE [16], which is a well known data record extraction system. Table 1 shows the experimental results of both this approach and ViDE. As we can see from Table 1, this approach has much better experimental results than ViDE, and in almost every domain this approach significantly outperforms ViDE. The precision and recall of this approach are both high across all domains, approaching 100%. This approach can also extract query result pages with single data records.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Our approach</th>
<th>ViDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>97.86%</td>
<td>96.76%</td>
</tr>
<tr>
<td>Hotel</td>
<td>99.20%</td>
<td>98.30%</td>
</tr>
<tr>
<td>Jobs</td>
<td>99.48%</td>
<td>98.37%</td>
</tr>
<tr>
<td>Movies &amp; Music</td>
<td>100%</td>
<td>98.54%</td>
</tr>
<tr>
<td>Single record</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Total</td>
<td>99.26%</td>
<td>98.11%</td>
</tr>
</tbody>
</table>

VIII. CONCLUSION

This paper presents a new approach an automatic for extracting data records from query result pages. This approach first uses the sizes of visual blocks and the occurrences of query terms in visual blocks to identify the data section. It then groups data units in the data section, which are in close proximity, into data records. It also uses content and visual features of visual blocks to evaluate their importance and to filter out noisy blocks. This work can be part of a web data integration system which interacts with multiple web databases, e.g. e-commerce web sites. This experimental results show that the proposed approach is highly effective. In future work, it will help to develop algorithms for aligning data units in the extracted data records so that data units of the same attribute can be aligned into the same column of the query result table.

REFERENCES

AUTHORS PROFILE

Dr. A. Govardhan did his B.E in Computer Science and Engineering from Osmania University College of Engineering, Hyderabad, M.Tech from Jawaharlal Nehru University, Delhi and Ph.D from Jawaharlal Nehru Technological University, Hyderabad. He is presently working as professor in computer science and engg and Direcotr of evaluation at JNTUH, Hyderabad. More than 10 members has submitted Ph.D under his guidance. He has guided more than 100 M.Tech projects and number of MCA and B.Tech projects. He has research publications at International/National Journals and Conferences. He has been a program committee member for various International and National conferences. He is also a reviewer of research papers of various conferences. He has delivered number of Keynote addresses and invited lectures. He is also a member in various professional bodies. His areas of interest include Databases, Data Warehousing & Mining, Information Retrieval, Computer Networks, Image Processing and Object Oriented Technologies.

Dr. P. Premchand has graduated in Electrical Engineering from National Institute of Technology, Jamshedpur. He has obtained his M.E and Ph.D degrees in the branch of computer science and engineering from Andhra University, Visakapatnam. He has joined as lecturer in the department of CSE of Andhra University, Visakapatnam. Later he has shifted to Osmania University, Hyderabad into department of CSE as Associate professor.

He has also served in various positions such as director at AICTE New Delhi and as an additional controller of Exams at Osmania University, Hyderabad. Later he is elevated as Professor in his parent Department at Osmania University and served as Head of the Department CSE and chair man Board of Studies and presently he is serving as Dean Faculty of Informatics at Osmania University Hyderabad.He is also an active member of AICTE –NBA, selection committee member at J.N.T.U, A.U, A.N.U, K.U, ISRO, NRSA and ADRIN. He is actively involved in research, supervising 5 research students for the award of their Ph.D and many more students are pursuing their Ph.D at O.U, JNTU and A.N.U. He has presented several papers in national and international conferences and journals.

A. Suresh Babu Received the Bachelor's Degree in Computer Science and Engineering from Sri Krishnadevaraya University Annapatapur and Master's Degree in Computer Science and Engineering from J.N.T.University college of engg Hyderabad. Now pursuing Ph.D in Computer Science and Engineering from J.N.T.University Annapatapur. He is presently working as Assistant Professor in CSE Department at JNTUACE Pulivendula. His current research interests in Data Mining.