Structured Data Extraction from the Web Based on Partial Tree Alignment

Yanhong Zhai and Bing Liu

Abstract—This paper studies the problem of structured data extraction from arbitrary Web pages. The objective of the proposed research is to automatically segment data records in a page, extract data items/fields from these records, and store the extracted data in a database. Existing methods addressing the problem can be classified into three categories. Methods in the first category provide some languages to facilitate the construction of data extraction systems. Methods in the second category use machine learning techniques to learn wrappers (which are data extraction programs) from human labeled examples. Manual labeling is time-consuming and is hard to scale to a large number of sites on the Web. Methods in the third category are based on the idea of automatic pattern discovery. However, multiple pages that conform to a common schema are usually needed as the input. In this paper, we propose a novel and effective technique (called DEPTA) to perform the task of Web data extraction automatically. The method consists of two steps: 1) identifying individual records in a page and 2) aligning and extracting data items from the identified records. For step 1, a method based on *visual information* and *tree matching* is used to segment data records. For step 2, a novel *partial alignment* technique is proposed. This method aligns only those data items in a pair of records that can be aligned with certainty, making no commitment on the rest of the items. Experimental results obtained using a large number of Web pages from diverse domains show that the proposed two-step technique is highly effective.

Index Terms—Web data extraction, wrapper generation, partial tree alignement, Web mining.

1 Introduction

STRUCTURED data in Web pages usually contain important information. Such data are often retrieved from underlying databases and displayed in Web pages using fixed templates. In this paper, we call these structured data objects *data records*. Extracting data records enables one to integrate data from multiple Web sites and pages to provide value-added services, e.g., comparative shopping, meta querying and search. Fig. 1 shows some data records on the Web. Fig. 1a is a Web page segment containing a list of two products (books). The description of each book is a data record. Fig. 1b is a page segment containing a data table, where each table row is a data record.

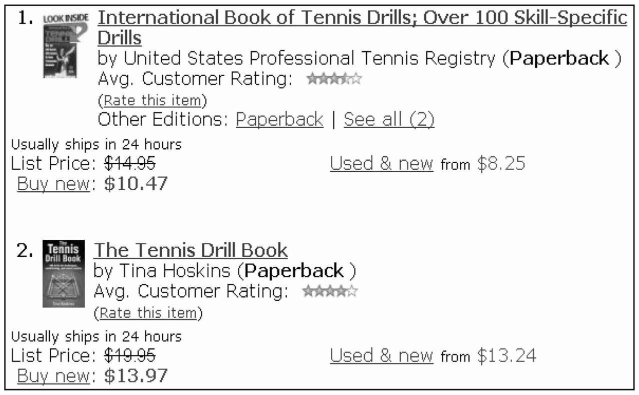
This paper proposes a novel technique to perform automatic data extraction given a single page with lists of data records. Most existing methods require multiple pages. In our view, requiring multiple pages is a limitation. If a page contains a list of data records, a single page is sufficient for finding patterns from the data records in the list to extract data from them. We will discuss this further in the related work section. In addition, our method is able to extract data from noncontiguous data records which cannot be done with existing techniques. By noncontiguous data records, we mean that the HTML codes of these data records intertwine with one another, but when they are displayed on a Web browser, they appear contiguous to the human user.

• The authors are with the Department of Computer Science, University of Illinois at Chicago, 851 South Mogan Street, Chicago, IL 60607. E-mail: {yzhai, liub}@cs.uic.edu.

Manuscript received 1 Aug. 2005; revised 19 Jan. 2006; accepted 5 June 2006; published online 18 Oct. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0288-0805.

Our objective is two fold: 1) to automatically identify data records in a page and 2) to automatically align and extract data items from the records. To perform these tasks, our proposed technique uses two steps:

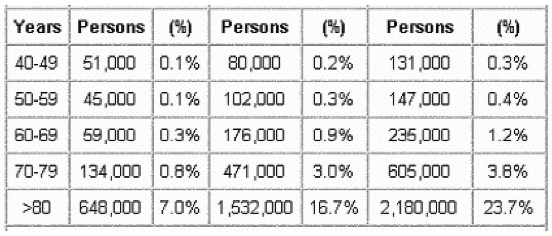
1. Given a page, the method first segments the page using an enhanced tree matching algorithm to identify each data record without extracting its data items. That is, the algorithm identifies data records by analyzing the DOM tree of the page. Visual information is used in this step in three ways. First, it is used to build the DOM tree. A straightforward way to build a DOM tree is to follow the nested tag structure in the HTML code. However, sophisticated analysis has to be incorporated to handle errors in the HTML code (e.g., missing or ill-formatted tags). Whereas the visual information can be obtained after the HTML code is rendered by a Web browser, it also contains information about the hierarchical structure of the tags. In this work, rather than analyzing and correcting the HTML code, visual information is utilized to infer the structural relationship among tags and to construct a DOM tree. This method leads to more robust tree construction due to the high error tolerance of the rendering engines of Web browsers (e.g., Internet Explorer). As long as a page can be rendered correctly by a browser, its DOM tree can be built correctly. Second, visual information helps reduce the tree matching computation as two objects of very different sizes visually are unlikely to match. Finally, visual information helps to improve the accuracy of data record segmentation by allowing the system to identify a valuable piece of information that is not available in the DOM tree, the space gaps between data records. The information can be used to segment data records as any gap within a data



1. International Book of Tennis Drills; Over 100 Skill-Specific Drills
by United States Professional Tennis Registry (Paperback)
Avg. Customer Rating:
(Rate this item)
Other Editions: Paperback | See all (2)
Usually ships in 24 hours
List Price: ~~$14.95~~ Used & new from $8.25
Buy new: $10.47

2. The Tennis Drill Book
by Tina Hoskins (Paperback)
Avg. Customer Rating:
(Rate this item)
Usually ships in 24 hours
List Price: ~~$19.95~~ Used & new from $13.24
Buy new: $13.97

(a)



Years	Persons	(%)	Persons	(%)	Persons	(%)
40-49	51,000	0.1%	80,000	0.2%	131,000	0.3%
50-59	45,000	0.1%	102,000	0.3%	147,000	0.4%
60-69	59,000	0.3%	176,000	0.9%	235,000	1.2%
70-79	134,000	0.8%	471,000	3.0%	605,000	3.8%
>80	648,000	7.0%	1,532,000	16.7%	2,180,000	23.7%

(b)

Fig. 1. Two example page segments with data records. (a) A list of products. (b) A data table.

record is typically smaller than that in between data records.

2. A novel partial tree alignment method is proposed to align corresponding data items from the discovered data records and put the data items in a database table. Using tree alignment is natural because of the nested (or tree structured) organization of the HTML code. Specifically, after all data records have been identified, the subtrees of each data record are rearranged into a single tree as each data record may be contained in more than one subtree in the original DOM tree of the page, and each data record may not be contiguous. The DOM trees of all the data records are then aligned using our partial tree alignment method. By partial tree alignment, we mean that, for each pair of trees (or data records), we only align those nodes (or data items) that can be aligned with certainty, making no commitment on the locations of the unaligned data items. Early uncertain commitments can result in undesirable effects for later alignment involving other data records. This method turns out to be very effective for multiple tree alignment. The resulting alignment enables us to extract items from all data records in the page. It can also serve as a pattern to be used to extract data items from other similar pages.

The proposed approach is called DEPTA (*Data Extraction based on Partial Tree Alignment*), which is very different from all existing methods and does not make those assumptions made by them. As long as a page contains at least two data records, DEPTA will automatically find them. Experimental results show that this approach is highly effective.

This paper is organized as follows: Section 2 discusses the related work. Section 3 presents the system architecture. Section 4 describes the algorithm for identifying data records. Section 5 presents the method for aligning and extracting data items in the data records. Experimental results are given in Section 6. Section 7 concludes the paper.

2 Related Work

Related works on Web data extraction can be classified into three categories: 1) wrapper programming languages, 2) wrapper induction, and 3) automatic extraction. The first approach provides some specialized pattern specification languages to help the user construct extraction programs. Visual platforms are also provided to hide their complexities under simple graphical wizards and interactive processes. Systems that use this approach include WICCAP [40], Wargo [26], Lixto [3], DEByе [19], etc. The second approach is wrapper induction, which uses supervised learning to learn data extraction rules from a set of manually labeled examples. Manual labeling of data is labor intensive and time consuming. Furthermore, for different sites or even pages in the same site, the manual labeling process needs to be repeated because they may follow different templates. Example wrapper induction systems include WIEN [18], Softmealy [16], Stalker [23], WL^2 [8], [25], Thresher [15], IDE [37], etc. Our technique requires no human labeling.

The third approach is automatic extraction. Embley et al. [10] proposes using a set of heuristics and domain ontologies to automatically identify data record boundaries. Buttler et al. [4] proposes additional heuristics for the task without using domain ontologies. However, experimental results in [22] show that the performance of this approach is not satisfactory.

In [6], a method (called IEPAD) is proposed to find patterns from the HTML tag string of a page, and then use the patterns to extract data items. The method uses the Patricia tree [12] and sequence alignment. However, the algorithm generates many spurious patterns. Users have to manually select the correct one for extraction. Our method use tree alignment instead of string alignment, which exploits nested tree structures to perform much more accurate data extraction. Following the work in [6], another system, called DeLa, is reported in [34]. It generates a candidate wrapper based on a single page by finding repeated patterns also in the HTML string, and then multiple similar pages are used to determine a generalized wrapper. This work thus needs multiple pages with a common template as the input.

In [9], a matching algorithm (called RoadRunner) is proposed to infer union-free regular expressions from multiple pages with the same template. The full algorithm has an exponential time complexity. To limit the search, heuristic pruning techniques are employed, which compromise the expressive power of the inferred grammar. This approach is improved in [1], which presents a polynomial time algorithm (called EXALG) by using several heuristics. Both methods need multiple input pages with a common schema/template. In our work, we are concerned with extracting data from a single page with multiple data records, which is also called a *list page*. Another method for data extraction is proposed in [21]. Its main idea is to utilize the redundant information in list pages and detail pages (each of such pages containing additional details about the corresponding data record/object in list pages) to aid information extraction. The method also needs multiple similar pages to infer a template to find tables in list pages, and then finds records from the tables. It is not applicable to pages where data records in list pages have no links to detail pages, e.g., Fig. 1b. Requiring detail pages is a limitation. In [27], tree matching is used to extract news from news pages. It uses clustering to group pages with similar templates and then matches those pages in each cluster. This is different from our work as it works with multiple pages.

It is clear that, for extraction from detail pages, multiple pages are needed to find patterns because each page only focuses on a single object. However, for list pages, a single page is sufficient because it is possible to find patterns from multiple data records in a list to extract such data records. Requiring multiple list pages is a limitation.

Visual information of Web pages has been used in Web mining by several authors. Using visual layout of documents for data extraction is proposed in [28]. Visual information is also used to segment a Web page into different blocks [30]. In [20], it is utilized to label the extracted data. A recent technique also uses the visual information to find data records from returned pages of search engines [39]. However, it does not extract data items from data records.

An assumption that most existing systems make is that the relevant data of a data record is contained in a contiguous segment of the HTML code. However, in some Web pages, the description of one object may intertwine with the descriptions of some other objects. For example, the descriptions of two objects in the HTML source may follow this sequence: part1 of object1, part1 of object2, part2 of object1, and part2 of object2. In this case, the descriptions of the two objects are not contiguous. However, when they are displayed in a browser, they appear contiguous to human viewers. Our method can identify noncontiguous data records and extract their data items.

3 The Architecture of our Prototype System

The general architecture of our system DEPTA is given in Fig. 2. The input to the system is a Web page containing lists of data records (a page may contain multiple regions or areas with regularly structured data records). The system is composed of the following main components:

1. DOM tree builder: It is implemented using MSHTML API, which returns the rendering information of each HTML element. Using the rendering information and the HTML opening tags, this component builds the DOM tree of an input HTML page.
2. Data Regions Identifier: It traverses the DOM tree top-down to identify each area or region in the page that contains a list of similar data records.
3. Data Records Identifier: It detects the boundaries of individual data records in each region. The output is a list of data records (still in HTML) from each region.
4. Data Items Extractor: It aligns and extracts data items by applying the proposed partial tree alignment algorithm. The output of this component is an excel file where the extracted data items are aligned in tables. For a Web page with multiple data regions, data from each region is put in a separate table.

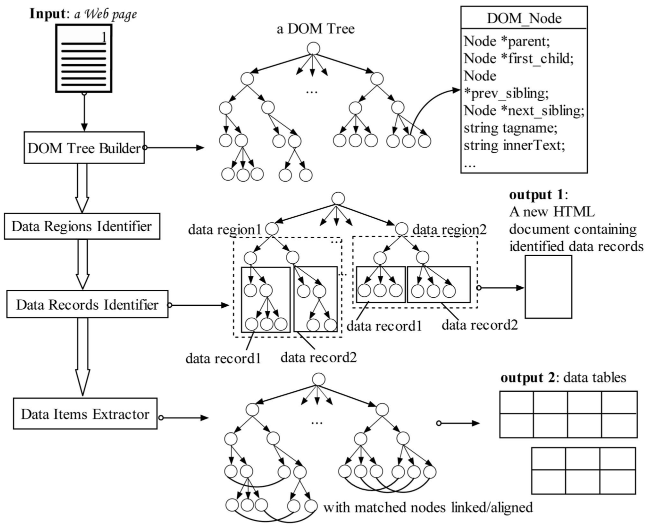


Fig. 2. The general architecture of the DEPTA system.

4 Data Record Identification

This section focuses on the first step, which segments the Web page to identify individual data records. It does not align or extract data items in the data records, which will be the topic of the next section. Since this step is an improvement to our previous technique MDR [22], below we give a brief overview of the MDR algorithm and present the enhancements made to MDR in this work. We also call the enhanced algorithm MDR-2 (version 2 of MDR).

The MDR algorithm is based on two observations about data records in a Web page and a tree matching algorithm [35] for finding data records. The two observations are:

1. A group of data records that contains the descriptions of a set of similar objects are typically rendered in a contiguous region of a page and are formatted using similar HTML tags. Such a region is called a *data record region* (or *data region* in short). For example, in Fig. 1, two books are presented in one contiguous region. They are also formatted using similar HTML tag subtrees. We can use a tree matching approach (introduced in Section 4.2) to compare different subtrees to find similar ones, which may represent similar data records. The problem with this approach is that the computation is prohibitive because a data record can start from any tag and end at any tag. The next observation helps deal with this problem.
2. A set of similar data records are formed by some child subtrees of the same parent node. For example, Fig. 3 shows the (partial) DOM tree of Fig. 1a. In this

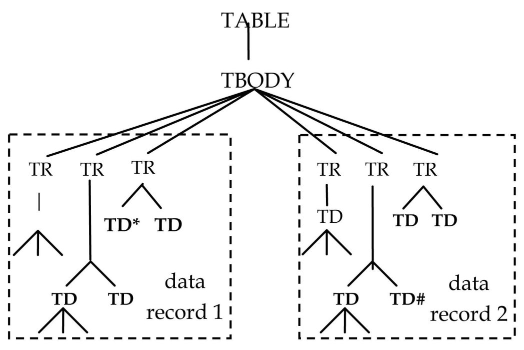


Fig. 3. An example DOM tree of a page segment.

tree, each data record is wrapped in three TR nodes with their subtrees under the same parent TBODY. The two data records are in the two dash-lined boxes. It is unlikely that a data record starts in the middle of a child subtree and ends in the middle of another child subtree. Instead, it starts from the beginning of a child subtree and ends at the end of the same or a later child subtree. For example, it is unlikely that a data record starts from TD* and ends at TD# (Fig. 3). This observation makes it possible to design a very efficient algorithm based on tree mapping to identify data records because it limits the tags at which a data record may start and end.

Experimental results show that these observations work very well. We note that a page can have a few data regions. Each region may have a different format for its records.

Given a Web page, our algorithm works in three steps (our enhancements made to MDR are also discussed):

1. Building a DOM tree of the page. In the new system, visual (rendering) information is used to build the tree.
2. Mining data regions in the page using the DOM tree. A data region is an area in the page that contains a list of similar data records. Instead of mining data records directly, which is hard, the algorithm mines data regions first and then finds data records within them. For example, in Fig. 3, we first find the data region below node TBODY. Both visual information and structure information are used in this step to produce accurate results.
3. Identifying data records from each data region. For example, in Fig. 3, this step finds data record 1 and data record 2 in the data region below node TBODY.

The main enhancement to the MDR algorithm is the use of visual information to help build more robust trees and to find more accurate data regions. We describe them in the next section.

4.1 Building DOM Trees Based on Visual Information

In a Web browser, each HTML element (consisting of a start tag, optional attributes, optional embedded HTML or textual content, and an end tag which may be omitted) is rendered as a rectangle. Each HTML element corresponds to a node in a DOM tree. All the nodes containing textual fragments or images are leaf nodes. We also call these nodes the *terminal nodes*. Before creating a DOM tree, the formatting tags within a textual fragment are removed, e.g., "<TD> Theoretical <B>computer</B> science</TD>" becomes "<TD>Theoretical computer science<TD>." A DOM tree can be constructed based on the nested rectangles. The details are as follows:

1. Find the four boundaries of the rectangle of each HTML element by calling the embedded HTML rendering engine of a Web browser.
2. Check for containment relationships among the rectangles, i.e., whether one rectangle is contained inside another rectangle. A tree can be built based on the checks.

```
Procedure BuildTree(preTag, HTMLSource)  // preTag is the root initially
1. for each open tag currentTag in HTMLSource in sequence do
2.   currentTag.Rect := getBoundingRect(currentTag); // find the bounding
         rectangle of currentTag
3.   while preTag is a valid tree node do
4.     if currentTag.Rect is contained in preTag.Rect then
5.       insert currentTag as the next child node of preTag and break;
6.     else preTag := preTag.parent;
7.   preTag := currentTag.
```

Fig. 4. Tree construction based on the visual information.

Fig. 4 gives the tree construction procedure based on visual information. *x.Rect* is the bounding rectangle of the open tag x (an HTML element). *x.parent* is the parent node of x.

Let us use an example to illustrate the process. Assume we have the HTML code on the left of Fig. 5, which is a table with two rows (trs) and each row with two cells (tds). However, there are three errors in the HTML code. The closing tag </td> for line 3 is put after the opening tag for line 4. Also, the closing tags </tr> in line 4 and </td> in line 7 are missing. However, this HTML segment can be rendered correctly in a browser, with the boundary coordinates (in pixels) for each HTML element shown in the middle of Fig. 5. With this visual information, we can build the tree on the right (Fig. 5) by following the sequence of opening tags and by containment checks of rectangles.

4.2 Tree Distance Measures and Enhanced Simple Tree Matching

To mine data regions in a page, tree pattern matching is employed to find regions that contain lists of similar subtrees. Based on different notions of distances, several distance measures for comparing trees have been proposed in the literature, e.g., tree edit distance [31], alignment

```
1 <table>
2   <tr>
3       <td> data1 <td>
4       </td> data2 </td>
5   <tr>
6       <td> data3 </td>
7       <td> data4
8   </tr>
9 </table>
```

left	right	top	bottom
100	300	200	400
100	300	200	300
100	200	200	300
200	300	200	300
100	300	300	400
100	200	300	400
200	300	300	400

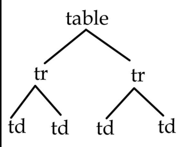


Fig. 5. An HTML code segment with ill-formatted tags (tags with erroneous nested structure and missing closing tags) (left), the boundary coordinates (middle), and the resulting DOM tree (right).

distance [17], isolated-subtree distance [32], top-down distance [31], [35], and bottom-up distance [33]. In this section, we first briefly review some tree distance measures and then present our *enhanced simple tree matching method*. Since DOM trees are rooted, ordered, and labeled, we will only consider rooted, ordered, and labeled trees.

4.2.1 Tree Edit Distance

Similar to string edit distance, tree edit distance [31] between two trees T_1 and T_2 is the cost associated with the minimum set of operations needed to transform T_1 into T_2. In the classic formulation, the set of allowable edit operations includes node removal, node insertion, and node replacement. A cost is assigned to each kind of operation. Solving the tree edit distance problem also finds a minimum-cost mapping between two trees [31]. The concept of mapping [31] is formally defined as follows:

Let T be a tree. n denotes the number of nodes of T. $T[i]$ denotes the $i\text{th}$ node of tree T in a preorder traversal of the tree. A triple (M, T_1, T_2) is a mapping M from T_1 to T_2, where M is any set of pairs of integers (i, j) satisfying:

1. $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$.
2. For any two pairs (i_1, j_1) and (i_2, j_2) in M,
    a. $i_1 = i_2$ iff $j_1 = j_2$,
    b. $i_1 < i_2$ iff $j_1 < j_2$, and
    c. $T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$.

Intuitively, the definition requires that each node can appear no more than once in a mapping and the order between sibling nodes and the hierarchical relation between nodes are both preserved. However, mapping can cross levels. An algorithm is given in [31] to compute the edit distance between two trees in $O(n_1 n_2 h_1 h_2)$ time, where n_i and h_i are the number of nodes and the maximum depth of T_i. The best upper bound for this problem is due to an algorithm presented in [7] with the time complexity $O(n_1 n_2 + l_1^2 + l_1^{2.5} l_2)$, where l_i is the number of leaves in T_i. The tree edit distance problem for unordered trees is NP-complete [38].

4.2.2 Top-Down and Bottom-Up Distances

The top-down distance is introduced in [29], and an algorithm is given in [35] to compute the distance between two trees T_1 and T_2 in $O(n_1 n_2)$ time. A mapping M from tree T_1 to T_2 is top-down if it satisfies the condition: For all i, j such that $T_1[i]$ and $T_2[j]$ are not root nodes, if $(i, j) \in M$, then $(parent(i), parent(j)) \in M$.

The bottom-up distance is introduced in [33], in which for any pair of nodes $T_1[i]$, $T_2[j]$, if $(i, j) \in M$, all child nodes of $T_1[i]$ and $T_2[j]$ should also be in the mapping M. An algorithm for computing this distance is proposed in [33], which has the time complexity of $O(n_1 + n_2)$.

Both top-down distance and bottom-up distance are restricted versions of the tree edit distance. In this paper, we deal with DOM trees that represent Web page structures. Pages generated from a common schema/template differ from each other because different values are retrieved from back-end databases and inserted into the schema. In addition, some data fields are optional. They are shown at the lower level of the DOM trees (or subtrees corresponding to data records). Thus, these DOM trees (or subtrees) usually have more structural dissimilarity at the lower levels (which represent data items) than at the upper levels. Hence, the bottom-up distance is not suitable for our task. Instead, the top-down distance is more appropriate. We thus adopt the top-down distance measure in this work.

```
Algorithm: ESTM($T_1$, $T_2$)
1. if $r_1$ and $r_2$ (the roots of the two trees $T_1$ and $T_2$ ) contain distinct
        symbols OR have visual conflict then
2     return 0;
3 else    $m$ := the number of first-level sub-trees of $T_1$;
4         $n$ := the number of first-level sub-trees of $T_2$;
5         Initialization:    $M[i, 0]$ := 0 for $i$ = 0, …, $m$;
                             $M[0, j]$ := 0 for $j$ = 0, …, $n$;
6         for $i$ = 1 to $m$ do
7            for $j$ = 1 to $n$ do
8               $M[i, j]$ := max($M[i, j-1]$, $M[i-1, j]$, $M[i-1, j-1]+W[i, j]$),
                    where $W[i, j]$ := ESTM ($T_{1[i]}$, $T_{2[j]}$)
9         return ($M[m, n]$+1+ content_similarity($r_1$ , $r_2$))
```

(a)

```
Procedure content_similarity($r_1$, $r_2$)
1 if $r_1$ and $r_2$ are not leaf nodes then
2     return 0
3 else $cs$ := LCS($r_1$.data, $r_2$.data) ;
4     $w$ := the number of words contained in $cs$;
5     $m$ := the maximal number of words contained in $r_1$.data and $r_2$.data;
6 return $w/m$.
```

(b)

Fig. 6. (a) The enhanced simple tree matching (ESTM) algorithm. (b) Calculating the word-level similarity between the textual contents enclosed in two nodes.

4.2.3 Enhanced Simple Tree Matching

We now introduce an algorithm for computing top-down distance, which is used in our work. The original algorithm is proposed in [35] as a method for comparing two computer programs. It is called the *simple tree matching* (STM). STM evaluates the similarity of two trees by producing a maximum matching through preorder traversal of the trees with the complexity of $O(n_1 n_2)$, where n_1 and n_2 are the sizes of the trees T_1 and T_2, respectively. No node replacement or level crossing is allowed. Maximum matching of two trees T_1 and T_2 is a top-down mapping M with the maximum number of pairs of nodes having the same label.

Let $T_1 = \langle r_1, T_{1_{[1]}}, \ldots, T_{1_{[m]}} \rangle$ and $T_2 = \langle r_2, T_{2_{[1]}}, \ldots, T_{2_{[n]}} \rangle$ be two trees, where r_1 and r_2 are the roots of T_1 and T_2, and $T_{1_{[i]}}$ and $T_{2_{[j]}}$ are the $i\text{th}$ and $j\text{th}$ first-level subtrees of T_1 and T_2, respectively. When r_1 and r_2 match, the maximum matching between $T_1$and T_2 is $M_{T_1 T_2} + 1$, where $M_{T_1 T_2}$ is the maximum matching between $\langle T_{1_{[1]}}, \ldots, \langle T_{1_{[m]}} \rangle$ and $\langle T_{2_{[1]}}, \ldots, T_{2_{[n]}} \rangle$. We have enhanced the STM algorithm in two aspects: 1) using visual information to reduce the matching computation and 2) using the text content similarity to resolve conflicts in matching and alignment.

Fig. 6a shows our enhanced simple tree matching (ESTM) algorithm in which the roots of T_1 and T_2 are compared first (line 1). If the roots contain distinct labels or have visual conflicts, the two trees do not match at all. By visual conflict, we mean that the sizes of the bounding rectangles of the two roots are so different (one is twice as large as another or more in either x or y direction) that there

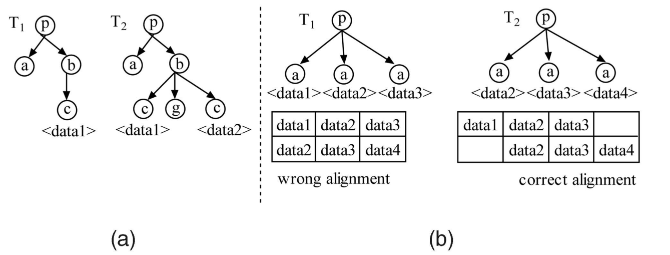


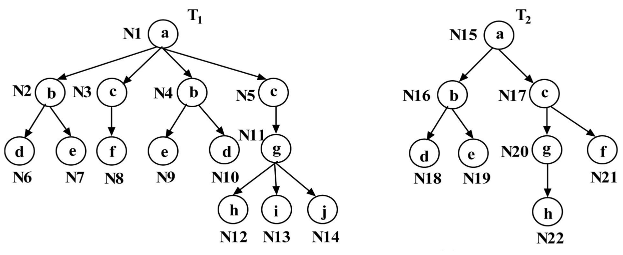
Fig. 7. (a) Two trees with more than one possible matches. (b) Alignment using tags only can produce wrong alignments.

is no possibility that they contain matched data. This reduces the computation significantly. If the roots contain identical labels and are visually consistent, then the algorithm recursively finds the maximum matching between first-level subtrees of T_1 and T_2 and save it in W matrix (line 8). Based on the W matrix, a dynamic programming scheme is applied to find the number of pairs in a maximum matching between two trees T_1 and T_2.

Fig. 6b gives the procedure for calculating the similarity between the content items at the word level. We use a longest common subsequence algorithm (LCS [11]) for the purpose. Lines 1 and 2 say that if r_1 and r_2 are not leaf nodes, return 0 since they have no textual contents. If they are both leaf nodes with textual contents, then compute the longest common subsequence of the two strings, $r_1.data$. and $r_2.data$ (they are the content items enclosed in nodes r_1 and r_2). The similarity value is then calculated as the number of words contained in the LCS sequence, normalized by the maximal number of words in $r_1.data$ and $r_2.data$.

The reason for introducing the content-based function content_similarity (r_1, r_2) is that nodes could be incorrectly aligned in some cases if the content items enclosed by these nodes are ignored. Fig. 7 shows such an example. In Fig. 7, the items in angle brackets represent the content information enclosed in the nodes, which could be words or sentences. The letters (a, b, and c) in circular nodes represent tag names. We notice that in Fig. 7a, node c in tree T_1 can match either the first or the last node c in tree T_2 because both give the maximum matching score if we consider tag names only. In Fig. 7b, using only tag names in matching gives a wrong alignment because data items of different attributes are enclosed by the nodes with the same tag name. Using content information in matching helps alleviate the problem in most cases because data items from the same attribute often share some common words. We call this step *conflict resolution*.

We use an example (Fig. 8) to explain the algorithm. To find the maximum matching between trees T_1 and T_2, their roots, N1 and N15, are compared first. Since N1 and N15 contain identical symbols, $M_{1-15}[4,2] + 1 + \text{content\_similarity}$ (N1, N15) is returned as the maximum matching score between trees T_1 and T_2 (line 11). Note that, in this example, we do not use $\text{content\_similarity}()$. M_{1-15} matrix is computed based on the W_{1-15} matrix, and each entry in W_{1-15}, say $W_{1-15}[i,j]$, is the maximum matching between the ith and jth first-level subtrees of T_1 and T_2, which is computed recursively based on its M matrix. For example, $W_{1-15}[4,2]$ is computed recursively by building the matrices Figs. 8e, 8f, 8g, and 8h.



(a) (b)

0	0	1(N16)	2(N16-17)
0	0	0	0
1(N2)	0	3	3
2(N2-N3)	0	3	5
3(N2-N4)	0	3	5
4(N2-N5)	0	3	6

(c)

	1(N16)	2(N17)
1(N2)	3	0
2(N3)	0	2
3(N4)	2	0
4(N5)	0	3

(d)

	0	1(N20)	2(N20-N21)
0	0	0	0
1(N11)	0	0	2

(e)

	1(N20)	2(N21)
1(N11)	2	0

(f)

	0	1(N22)
0	0	0
1(N12)	0	1
2(N12-N13)	0	1
3(N12-N14)	0	1

(g)

	1(N22)
1(N12)	1
2(N13)	0
3(N14)	0

(h)

Fig. 8. (a) Tree T_1. (b) Tree T_2. (c) M matrix for the first level subtrees of N1 and N15. (d) W matrix for the first level subtrees of N1 and N15. (e)-(h) M matrixes and W matrixes for the lower level subtrees.

All the relevant cells are shaded. The zero columns and rows in M matrices are initializations. Note that we use subscripts for both M and W matrices to indicate the nodes that they are working on. After the matching, we can trace back in the M matrices to find the matching nodes from the two trees and align them suitably.

4.3 Mining Data Regions

Based on the top-down tree distance measure, this step mines every data region in a page that contains similar data records. Instead of mining data records directly, which is hard, we first mine *generalized nodes* (defined below) in a page. A sequence of adjacent generalized nodes forms a data region. From each data region, we will identify the actual data records (discussed later). Below, we define generalized nodes and data regions using DOM trees:

Definition 1. *A generalized node (or a node combination) of length r consists of $r (r \geq 1)$ nodes in a DOM tree with the following two properties:*

1. *the nodes all have the same parent and*
2. *the nodes are adjacent.*

The reason that we introduce the generalized node is to capture two situations:

1. A data record may be contained in a few sibling tag nodes rather than one. For example, in Fig. 1 and Fig. 3, we can see that each book is contained in three table rows (or three TR nodes).

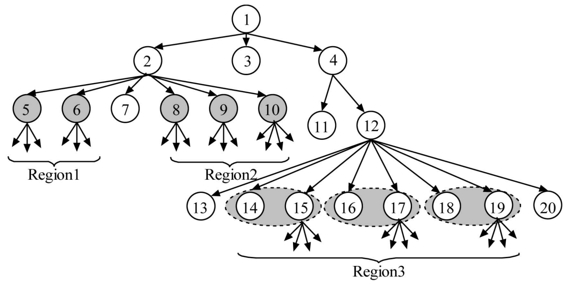


Fig. 9. An illustration of generalized nodes and data regions.

2. An object may not be contained in a contiguous segment of the HTML code.

Note that we call each node in a DOM tree a *tag node* to distinguish it from a *generalized node*.

Definition 2. *A data region is a list of generalized nodes with the following properties:*

1. *the generalized nodes all have the same parent,*
2. *the generalized nodes all have the same length,*
3. *the generalized nodes are all adjacent, and*
4. *the normalized tree distance between adjacent generalized nodes is less than a fixed threshold.*

Given two adjacent generalized nodes of length r, $G_1 = < g_{1[1]}, g_{1[2]}, \dots, g_{1[r]} >$ and $G_2 = < g_{2[1]}, g_{2[2]}, \dots, g_{2[r]} >$, the normalized tree distance between them (computed with the procedure NormTreeDist()) is defined as

$$1 - \frac{\text{MatchedTerminal}(ESTM(T_1, T_2))}{\max(\text{Terminal}(T_1), \text{Terminal}(T_2))},$$

where T_i is the tree with an artificial root node on top of the forest rooted at $g_{i[1]}, g_{i[2]}, \dots$ and $g_{i[r]}, i = 1, 2$. Terminal (T_i) returns the number of nodes enclosing images or texts in tree T_i. Such nodes are called *terminal nodes*. For example, in the tree representing a HTML segment <TABLE><TR> <TD> <B> price: </B> </TD> </TR> </TABLE>, the node labeled with <B> is the only terminal node and, in this case, Terminal (T_i) returns 1. MatchedTerminals(ESTM (T_1, T_2)) returns the number of matched terminal nodes in T_1 and T_2.

It is important to notice that, although the generalized nodes in a data region have the same length, their lower level nodes in their subtrees can be quite different. Thus, they can capture a wide variety of regularly structured objects. To further explain generalized nodes and data regions, we use an artificial tag tree in Fig. 9. The shaded areas are generalized nodes. Nodes 5, 6, 8, 9, and 10 are generalized nodes of length 1 and they form two data regions labeled 1 and 2. Nodes 14 to 19 form three generalized nodes of length 2 and they together define the data region 3.

We end this part with two important notes: 1) In practice, the above definitions are very robust as our experiments show. The key assumption here is that nodes forming a data region are from the same parent, which is realistic. 2) A generalized node may not represent a final data record (see Section 4.4). It will be used to find data records.

4.3.1 Comparing Generalized Nodes

In order to find each data region in a Web page, the mining algorithm needs to find the following: 1) Where does the

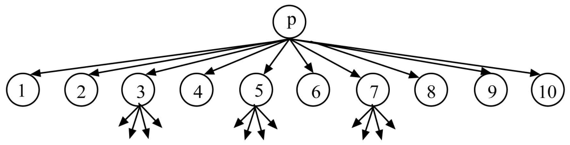


Fig. 10. Combination and comparison.

first generalized node of a data region start? 2) What is the length of a generalized node? Let the maximum number of tag nodes that a generalized node can have be K, which is normally a small number (e.g., in our experiment data, it is only 5). In order to answer question 1, we can try to find a data region starting from each node sequentially. To answer question 2, we can try: one node, two node combination, ..., and K node combination. The comparison results are then used to identify each data region. The number of comparisons is actually not very large because, due to our assumption, we only perform comparisons among the nodes with the same parent node (e.g., in Fig. 9, we do not compare node 8 with node 13) and some comparisons done for earlier nodes are the same as for later nodes. We use Fig. 10 to illustrate the comparison process. Fig. 10 has 10 nodes below a parent node p. We start from each node and calculate normalized tree distance between all possible combinations of component nodes. Let the maximum number of components that a generalized node can have be three in this example.

Start from node 1: We compute the following normalized tree distances:

- $(1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10)$
- $(< 1,2 >, < 3,4 >)$, $(< 3,4 >, < 5,6 >)$, $(< 5,6 >, < 7,8 >)$, $(< 7,8 >, < 9,10 >)$,
- $(< 1,2,3 >, < 4,5,6 >), (< 4,5,6 >, < 7,8,9 >)$.

(1, 2) means that the subtree rooted at node 1 is compared with the subtree rooted at node 2. $(< 1,2 >, < 3,4 >)$ means that the combined subtrees rooted at nodes 1 and 2 are compared with the combined subtrees rooted at nodes 3 and 4.

Start from node 2: We only compute:

- $(< 2,3 >, < 4,5 >)$, $(< 4,5 >, < 6,7 >)$, $(< 6,7 >, < 8,9 >)$ and
- $(< 2,3,4 >, < 5,6,7 >), (< 5,6,7 >, < 8,9,10 >)$.

We do not need to do 1-node comparisons because they have been done when we started from node 1 above.

Start from node 3: We only need to compute:

- $(< 3,4,5 >, < 6,7,8 >)$.

Again, we do not need to do 1-node comparisons. We also do not need to do 2-node comparisons because they have been done when we started from nodes 1 and 2.

We do not need to start from any other nodes after node 3 because all the computations have been done. It is fairly easy to prove that the process is complete.

The algorithm MDR-2 (an enhanced version of our MDR system) for mining data regions is given in Fig. 11a. It traverses the DOM tree from the root downward in a depth-first fashion (lines 3 and 4). At each internal node, procedure CombComp() calculates normalized tree distance between various combinations of children subtrees.

```
Algorithm MDR-2(Node, K)
1 if TreeDepth(Node) => 3 then
2     CombComp(Node.Children, K);
3     for each ChildNode ∈ Node.Children
4         MDR-2(ChildNode, K).

(a)

Procedure CombComp(NodeList, K)
1 for (i := 1; i <= K; i++)          /* start from each node */
2     for (j: = i; j <= K; j++)      /* comparing different combinations
3         if NodeList[i+2*j-1] exists then
4             St := i;
5             for (k := i+j; k < Size(NodeList); k+j)
6                 if NodeList[k+j-1] exists then
7                     NormTreeDist(NodeList[St..(k-1)],NodeList[k..(k+j-1)]);
8                     St := k.

(b)
```

Fig. 11. (a) The algorithm for mining data regions. (b) The structure comparison algorithm.

Line 1 says that the algorithm will not search for data regions if the depth of the subtree from *Node* is 2 or 1 as it is unlikely that a data region is formed with only a single level of tag(s) (data regions are formed by *Node*'s children).

The main idea of CombComp() has been discussed above. In line 1 of Fig. 11b, it starts from each node of *NodeList*. It only needs to try up to the $K\text{th}$ node. In line 2, it compares different combinations of nodes, beginning from $i\text{-component}$ combination to $K\text{-component}$ combination. Line 3 checks if there is at least one pair for comparison. Lines 4-8 calculate the normalized tree distance between various node combinations by calling NormTreeDist().

Assume that the number of elements in *NodeList* is n. Without considering the comparison of trees, the time complexity of CombComp is $O(nK)$. To see this, let us do the following analysis:

Starting at node 1, we need at most the following number of comparisons (running NormTreeDist()) (we assume that n is much larger than K):

$$(n-1)+\left(\frac{n}{2}-1\right)+\left(\frac{n}{3}-1\right)+\ldots+\left(\frac{n}{K}-1\right).$$

Starting from node 2, we have at most:

$$\left(\frac{n}{2}-1\right)+\left(\frac{n}{3}-1\right)+\ldots+\left(\frac{n}{K}-1\right).$$

Starting from node K, we have at most:

$$\left(\frac{n}{K}-1\right).$$

Adding all together, we have $nK-\frac{K(K+1)}{2}$, which gives $O(nK)$. Since K is normally small ($<$ 10), the algorithm can be considered linear in n. Assume that the total number of nodes in the DOM tree is N, the complexity of MDR-2 is O (NK) without considering tree comparison.

4.3.2 Determining Data Regions

After all comparisons have been done, we can identify each data region by finding its generalized nodes. We use Fig. 12 to illustrate the issues. There are eight data records (1-8) in this page. Our algorithm reports each row as a generalized node, and the whole area as a data region.

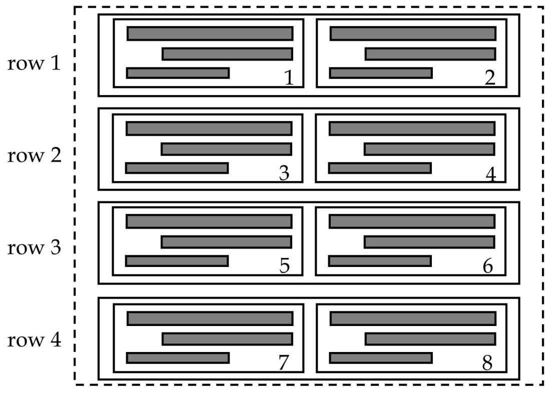


Fig. 12. A possible configuration of data records.

The algorithm basically uses the comparison results at each parent node to find similar child node combinations to obtain candidate generalized nodes and data regions. Three main issues are important for making the final decisions:

1. If a higher level data region covers a lower level data region, we report the higher level region. For example, in Fig. 12, at a low level, cell 1 and cell 2 are candidate generalized nodes and they together form a candidate data region, row 1. However, they are covered by the data region including all the four rows at a higher level. In this case, we only report the higher level one as the data region and each row as a generalized node. The reason for taking this approach is to avoid the situations where many low level nodes (with very small subtrees) are similar but do not represent true data records.
2. A property about similar trees is that if a set of trees $T_1, T_2, \ldots, T_n$ are similar to one another, then a combination of any number of them is also similar to another combination of them of the same number. Thus, we only report generalized nodes of the smallest length that cover a data region, which helps us to find the final data records later. In Fig. 12, we only report each row as a generalized node rather than a combination of two rows (rows 1-2, and rows 3-4).
3. Visual information is used to check the validity of detected data regions and to eliminate false node combinations. The following visual observation about data records is utilized to achieve this goal: *The gap between two data records in a data region should be no smaller than any gap within a data record*. For example, in Fig. 1a, a large gap exists between the two records.

The algorithm for this step is given in Fig. 13a. It finds every data region and its generalized nodes under a given node. τ is the normalized tree distance threshold. K is the maximum number of components of a generalized node. We use $K = 10$ in our experiments, which is sufficient. $Node.DRs$ is the set of data regions under $Node$, and $tempDRs$ is a temporal variable storing the data regions passed up from every $Child$ of $Node$. The basic idea of the algorithm is to traverse the DOM tree depth-first. It performs one function at each node when it goes down

```
procedure FindDRs(Node, K, τ)
1 if TreeDepth(Node) => 3 then
2    Node.DRs := IdenDRs(1, Node, K, τ);
3    tempDRs := ∅;
4    for each Child ∈ Node.Children
5       FindDRs(Child, K, τ);
6       tempDRs := tempDRs∪UnCoveredDRs(Node, Child);
7    Node.DRs := Node.DRs ∪ tempDRs.
                                (a)
Procedure IdentDRs(start, Node, K, τ)
1  maxDR := [0, 0, 0];
2  for (i := 1; i <= K; i++)        /* compute for each i-combination */
3     for (f := start; f <= start+i; f++) /* start from each node */
4       flag := true;
5       for (j := f; j < size(Node.Children); j+i)
6          if Distance(Node, i, j) <= τ then
7             if flag=true then
8                curDR := [i, j, 2*i];
9                flag := false;
10            else   curDR[3] := curDR[3] + i;
11         elseif flag := false then Exit-inner-loop;
12      if (maxDR[3] < curDR[3]) and (maxDR[2] = 0 or
                 (curDR[2]<= maxDR[2]) )and VisuallyValid(curDR) then
13         maxDR := curDR;
14 if ( maxDR[3] != 0 ) then
15    if (maxDR[2]+maxDR[3]-1 != size(Node.Children)) then
16       return {maxDR}∪ IdentDRs(maxDR[2]+maxDR[3], Node, K, τ);
17    else return {maxDR};
18 return ∅;
                                (b)
Procedure UnCoveredDRs(Node, Child)
1  for each data region DR in Child.DRs
2    if Child in range DR[2]…(DR[2]+DR[3]-1) then
3        return null
4  return Child.DRs
                                (c)
```

Fig. 13. (a) Finding data regions under a given node—the control procedure. (b) Data region identification below a node. (c) The UnCoveredDRs() procedure.

(line 2), and performs another when it backs up before going down to another branch of the tree (line 6).

When it goes down, at each node, it identifies all the data regions under the node using procedure IdentDRs(). Note that these are not the final data regions, but only the candidate ones. When it backs up, it checks to see whether the parent level data regions in $Node.DRs$ cover the child level data regions. Those covered child level data regions are discarded. Those that are not in $Child.DRs$ are stored in $tempDRs$ (line 6). After all child nodes of $Node$ are processed, $Node.DRs \cup tempDRs$ gives the current data regions discovered from the subtree starting from $Node$ (line 7).

We now discuss procedure IdentDRs(). Recall that the previous step has computed the distance values of all possible child node combinations. This procedure uses these values and the threshold τ to find data regions under $Node$. That is, it needs to decide which combinations represent generalized nodes, where the beginning is and where the end is for each data region.

Procedure IdentDRs() is given in Fig. 13b, which ensures the smallest generalized nodes are identified. The procedure is recursive (line 16). In each recursion, it extracts the next data region $maxDR$ that covers the maximum number of child nodes. $maxDR$ is described by three members (line 1), 1) the number of nodes in a combination, 2) the location of the start child node of the data region, and 3) the number of nodes covered by the data region. $curDR$ is the current candidate

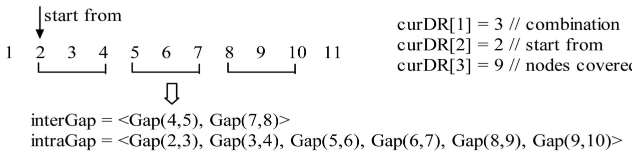

start from
1 2 3 4 5 6 7 8 9 10 11
curDR[1] = 3 // combination
curDR[2] = 2 // start from
curDR[3] = 9 // nodes covered
interGap = <Gap(4,5), Gap(7,8)>
intraGap = <Gap(2,3), Gap(3,4), Gap(5,6), Gap(6,7), Gap(8,9), Gap(9,10)>

Fig. 14. Illustration of procedure VisuallyValid().

data region being considered. Tree comparison results are stored in a data structure attached with each node. The value can be obtained by calling procedure $\mathrm{Distance}(Node,\ i,\ j)$, where i represents i-combination, and j represents the jth child of $Node$. IdentDRs() basically checks each combination (line 2) and each starting point (line 3). For each possibility, it finds the first continuous region with a set of generalized nodes (line 5-line 11). The conditions (line 12) ensure that smaller generalized nodes are used unless the larger ones cover more nodes and start no later than the smaller ones. VisuallyValid() validates the detected data regions based on the visual cues. It uses the heuristic that the gap between two data records in a data region should be no smaller than any gap within a data record. The idea is illustrated in Fig. 14.

Finally, procedure UnCoveredDRs() is given in Fig. 13c. Assume that the total number of nodes in the DOM tree is N, the complexity of FindDRs is $O(NK^2)$. Since K is normally very small. Thus, the computation requirement of the algorithm is low.

4.4 Identifying Data Records

After all data regions are discovered, we identify data records from generalized nodes. The following observation is useful: *If a generalized node contains two or more data records, these data records must be similar in terms of their DOM trees.* This is clear because we assume that a data region contains descriptions of similar data records. Identifying data records from each generalized node is relatively easy because they are nodes (together with their subtrees) at the same level as the generalized node, or nodes at a lower level. This can be done in two steps:

1. Produce one rooted tree for each generalized node: After all components of each generalized node are identified, the components, which are subtrees, are rearranged into a single tree. An artificial root node may be added.
2. Call the FindDRs() procedure again using the tree: Due to the observation above, this step will find the lower level generalized nodes if they exist, which are data records. Otherwise, the original generalized node is a data record. The lower level data records need to satisfy the following conditions:

   - They cover all the data items in the original generalized node.
   - Each data record is not too small, e.g., a single number or a piece of text, which is more likely to be an entry of a data (or spreadsheet) table.

One special case needs additional attention, i.e., non-contiguous data records. We discuss it below.

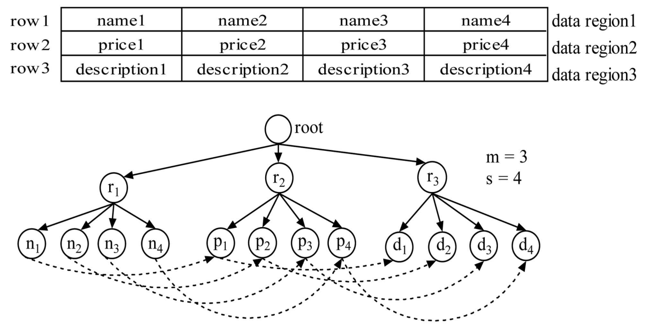

row1 name1 name2 name3 name4 data region1
row2 price1 price2 price3 price4 data region2
row3 description1 description2 description3 description4 data region3
root
m = 3
s = 4
r_1 r_2 r_3
n_1 n_2 n_3 n_4 p_1 p_2 p_3 p_4 d_1 d_2 d_3 d_4

Fig. 15. Three adjacent data regions form four noncontiguous data records.

4.4.1 *Noncontiguous Data Records*

In some Web pages, the description of an object (a data record) is not in a contiguous segment of the HTML code. Fig. 15 shows an example. In the top figure, we can see that row 1 lists the names of four objects, row 2 lists their prices, and row 3 lists their descriptions. This results in the following sequence in the HTML code: name 1, ..., name 4, price 1, ..., price 4, and description 1, ..., description 4. Each row is a data region with four generalized nodes and the three rows are adjacent. Then, the algorithm joins the three data regions together to form four data records. The bottom figure in Fig. 15 shows how this is done, where r represents row, n represents name, p represents price, and d represents description. We note that rows 1, 2, and 3 are not similar to each other; otherwise, the sibling nodes r_1, r_2, and r_3 would be three generalized nodes forming a single data region.

In general, noncontiguous data records are produced as follows: For a list of adjecent sibling parent nodes, $< p_1, p_2, \ldots, p_m >$, each p_i (which also represents a data region) has a list of generalized nodes $< g_{i[1]}, g_{i[2]}, \ldots, g_{i[s]} >$ under it. The numbers of generalized nodes under all the parents are the same, s. Then, for each $j (1 \leq j \leq s)$, the system sequentially combines all $g_{i[j]}$ (for $i = 1, 2, \ldots, m$) to form a noncontiguous data record. This gives us s data records. This process is illustrated by the example in Fig. 15.

5 Data Alignment

We are now ready to align corresponding data items from all data records. We present the partial tree alignment technique for this purpose, which consists of two substeps:

1. Produce one rooted DOM tree for each data record: After all data records are identified, the subtrees of each data record are rearranged into a single tree. As shown above, each data record may be contained in more than one subtree. Thus, this substep is needed to compose a rooted tree for each data record (an artificial root node may need to be added).
2. Partial tree alignment: The DOM trees of all data records in each data region are aligned.

Below, we focus on multiple alignments and present the partial tree alignment method for aligning multiple data records based on their DOM trees. We note here that string edit distance is less suitable as a string does not consider the tree structure, which is very useful in determining the correct alignment of data items. Due to the fact that more than one alignment of two strings may result in the same edit distance, string alignment can result in many errors. The matter is made worse by the fact that most tags used to form data records are trs and tds. After string matching, it is hard to decide which alignment is the correct one. However, tree matching reduces the number of possible alignments significantly because of the tree structure constraint.

5.1 Multiple Alignment

Since each data region in a page contains multiple data records, we need to align multiple DOM trees in order to produce a single database table with all the corresponding data items in the same column of the table. In this data table, each row represents a tree (data record), and each column represents a data item in each data record. Several existing algorithms can perform alignment of multiple sequences/trees. In [5], a multiple alignment method is proposed using multidimensional dynamic programming. The method is optimal but its time complexity is exponential, and thus not suitable for practical use. Many heuristic methods are also proposed [24], [14],[2]. The center string method, which is used in [6], is a particular heuristic method for multiple sequence alignments, which can also be used for trees. In this method, a sequence x_c that minimizes $\sum_{i=0}^{k} D(x_i, x_c)$ is selected as the center $(D(x_i, x_c)$ is the distance of two strings). Then, a pair-wise alignment is performed for each pair (x_i, x_c), where $i \neq c$. Assuming there are k sequences and all sequences have length n, finding the center takes $O(k^2n^2)$ time as each pair-wise comparison takes $O(n^2)$ time. Similarly, we can find a center tree T_c and align all the other trees with T_c. There are two main drawbacks with this technique: First, it runs slowly for pages containing many data records. Second, if the data record represented by the center tree does not have a particular data item, it may not be aligned properly even if other data records contain the data item. Our proposed method deals with both problems nicely. Other popular multiple alignment methods include progressive alignment [14] and iterative alignment [2]. They work like hierarchical clustering, and all requires upfront $O(k^2)$ pair-wise matching. For our task, we can do better because we know that data records follow some predefined template.

5.2 Partial Tree Alignment

Our proposed approach aligns multiple DOM trees by progressively growing a seed tree. The seed tree, denoted by T_s, is initially picked to be the tree with the maximum number of data fields. The reason for choosing this seed tree is clear as it is more likely for this tree to have a good alignment with data fields in other data records. Then, for each $T_i (i \neq s)$, the algorithm tries to find for each node in T_i a matching node in T_s. When a match is found for node $T_i[j]$, a link is created from $T_i[j]$ to $T_s[k]$ to indicate its match in the seed tree. If no match can be found for node $T_i[j]$, then the algorithm attempts to expand the seed tree by inserting $T_i[j]$ into T_s. The expanded seed tree T_s is then used in subsequent matching.

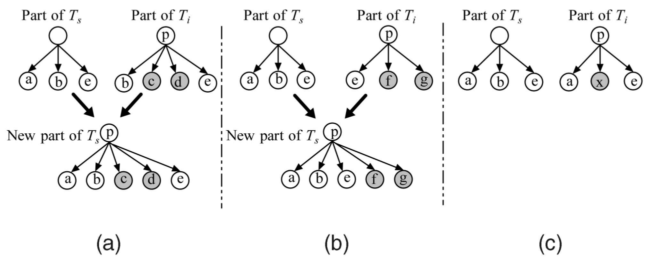


Fig. 16. Expanding the seed tree: (a) and (b) unique expansion and (c) insertion ambiguity.

5.2.1 Partial Alignment of Two Trees

Before presenting the full algorithm for aligning multiple trees, let us first discuss the idea of partial alignment of two trees. As indicated above, after T_s and T_i are matched, some nodes in T_i can be aligned with their corresponding nodes in T_s because they match one another. For those nodes in T_i that are not matched, we want to insert them into T_s as they may contain optional data items. There are two possible situations when inserting a new node $T_i[j]$ into the seed tree T_s, depending on whether a location in T_s can be uniquely determined to insert $T_i[j]$. In fact, instead of considering a single node $T_i[j]$, we can consider a sequence of unmatched consecutive sibling nodes $T_i[j] \dots T_i[m]$ together. Without loss of generality, we assume that the parent node of $T_i[j] \dots T_i[m]$ has a match in T_s and we want to insert $T_i[j] \dots T_i[m]$ into T_s under the matching parent node. We only insert $T_i[j] \dots T_i[m]$ into T_s if the location for the insertion can be uniquely determined in T$_s$. Otherwise, they will not be inserted and left unaligned. The alignment is thus partial. The location for insertion of $T_i[j] \dots T_i[m]$ can be uniquely decided in the following cases:

1. If $T_i[j] \dots T_i[m]$ have two neighboring siblings in T_i, one on the right and one on the left, that are matched with two consecutive siblings in T_s. Fig. 16a shows such a situation, which gives one part of T_s and one part of T_i. We can see that node c and node d (which are consecutive sibling nodes) in T_i can be inserted into T_s between node b and node e in T_s because node b and node e in T_s and T_i match. The new (extended) T_s is also shown in Fig. 16a. It should be noted that nodes a, b, c, d and e may also have their own children. We did not draw them to save space. This applied to all the cases below.
2. If $T_i[j] \dots T_i[m]$ has only one left (right) neighboring sibling x in T_i and x matches the right (left) most node x in T_s, then $T_i[j] \dots T_i[m]$ can be inserted after (before) node x in T_s. Fig. 16b illustrates this case.

Otherwise, we cannot uniquely decide a location for unmatched nodes in T_i to be inserted into T_s. This is illustrated in Fig. 16c. The unmatched node x in T_i could be inserted into T_s either between nodes a and b, or between nodes b and e. In this situation, we will not insert it into T_s.

5.2.2 Partial Alignment of Multiple Trees

Fig. 17 gives the algorithm for multiple tree alignment based on partial alignment of two trees. We use an example in Fig. 18 to explain the algorithm.

```
Algorithm PartialTreeAlignment(S)
1.  Sort trees in S in descending order according to the number of data
        items that are not aligned;
2.  T_s := the first tree (which is the largest) and delete it from S;
3.  flag := false; R = ∅; I = false;
4.  while (S ≠ ∅)
5.      T_i := select and delete next tree from S;
6.      ESTM(T_s, T_i);
7.      L := alignTrees(T_s, T_i);    // based on the result from line 6
8.      if T_i is not completely aligned with T_s then
9.          I = InsertIntoSeed(T_s, T_i);
10.         if not all unaligned items in T_i are inserted into T_s then
11.             Insert T_i into i;
12.     if (L has new alignment) or (I = true) then
13.         flag := true;
14.     if S = ∅ and flag = true then
15.         S := R; R := ∅;
16.         flag := false; I := false;
17. Output data fields from each T_i to the data table based on the align-
        ment results.
```

Fig. 17. The partial tree alignment algorithm.

We have three example trees, all of which have only two levels. Lines 1 and 2 (Fig. 17) basically find the tree with the most data items to be the seed tree. In Fig. 18, the seed tree is the first tree (we omitted many nodes on the left of T_1). Line 3 does some initializations. Line 4 starts the while loop to align each of the rest trees against T_s. Line 5 picks the next unaligned tree, and line 6 does the tree matching. Line 7 finds all the matched pairs by tracing the matrix results of line 6. This procedure is similar to align two strings using edit distance. Note that line 6 and line 7 can be integrated. We present them separately for simplicity. In Fig. 18, T_s and T_2 produce one match, node b, whereas nodes n, c, k, and g have no matching nodes in T_s. Line 9 attempts to insert them into T_s. This is the partial tree alignment discussed above. In Fig. 18, none of the nodes n, c, k, and g in T_2 can be inserted into T_s because no unique location can be found. Line 11 inserts T_2 into R, which is a list of trees that may need to be further processed. In Fig. 18, when matching T_3 with T_s, all unmatched nodes c, h, and k can be inserted into T_s. Thus, T_3 will not be inserted into R. Lines 12-13 set "$flag := \text{true}$" to indicate that some new alignments/matches are found or some unmatched nodes are inserted into T_s. Lines 14-16 check for stopping conditions. "$\text{S} = \emptyset$ and $flag = \text{true}$" means that we have processed all the trees in S, and some new alignments are found or insertions are done. Then, trees in R should be processed again. In Fig. 18, T_2 is the only tree in R, which will be matched to the new T_s in the next round. Now,

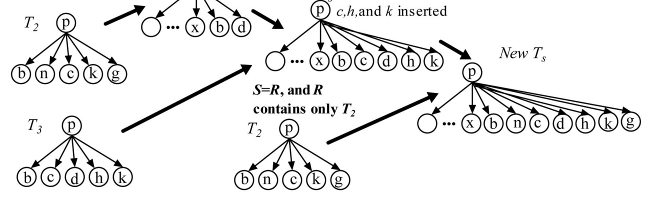


Fig. 18. Iterative tree alignment with two iterations.

TABLE 1
Final Data Table ("1" Indicates a Data Item)

	…	x	b	n	c	d	h	k	g
T_1	…	1	1			1			
T_2			1	1	1			1	1
T_3			1		1	1	1	1	

every node in T_2 can be matched or inserted. The process completes. Line 17 outputs the data items from each tree according to the alignment produced. Note that if there are still unmatched nodes with data after the algorithm completes, each unmatched data will occupy a single column by itself. Table 1 shows the data table for the trees in Fig. 18. We use "1" to indicate a data item.

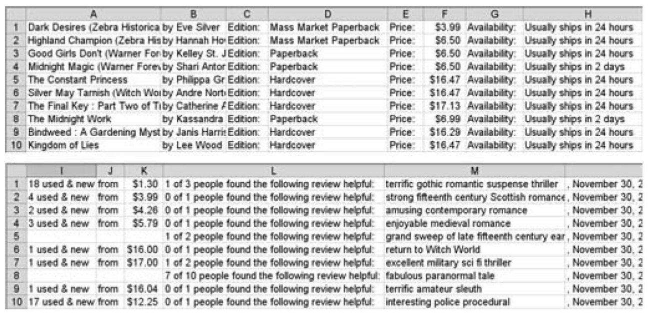
The complexity of the algorithm is $O(k^2)$ without considering tree matching, where k is the number of trees. However, in practice, we almost always need to go though S once (i.e., $R = \emptyset$) only. We note that the resulting alignment T_s can also be used as an extraction pattern for extracting data items from other pages from the same template.

5.2.3 Analysis of Aligned Data Items

After data items are aligned in tables, each column can be classified into one of the following four categories based on the similarities among the items contained in that column (Fig. 19 shows an example of an aligned table):

1. Identical: All the items are identical and exchangeable, e.g., columns C, E, G, and J in Fig. 19. The items in this category are most likely to be labels (attribute names) that are used to indicate the meaning of the items in an adjacent column. Note that the table in Fig. 19 is too wide and is thus split into two.
2. Comparable: The items in such a column contain a common substring (usually a prefix or a suffix substring). Examples include columns B, H, I, and L in Fig. 19. The items in this category present mixed information which indicates that both the data items and their associated labels or attributes are included. For example, all the items in column B contain a common word "by," which implies that the items after "by" are authors.
3. Enumerative: The items in this type of columns can be divided into a few subgroups and the items within each subgroup are identical. Columns D, F, H, I, and L in Fig. 19 are examples of enumerative columns.
4. Distinctive: The items in such columns are different from each other and contain representative information that is unique to each data record. For example, column A in Fig. 19 is one such column.

Further processing of the extracted data such as data labeling, data integration, or ontology construction can be performed based on the alignment result. However, a full investigation of these topics is beyond the scope of this paper. We plan to study them in our future work.



	A	B	C	D	E	F	G	H
1	Dark Desires (Zebra Historica	by Eve Silver	Edition:	Mass Market Paperback	Price:	$3.99	Availability:	Usually ships in 24 hours
2	Highland Champion (Zebra His	by Hannah Ho	Edition:	Mass Market Paperback	Price:	$6.50	Availability:	Usually ships in 24 hours
3	Good Girls Don't (Warner For	by Kelley St. J	Edition:	Paperback	Price:	$6.50	Availability:	Usually ships in 24 hours
4	Midnight Magic (Warner Forev	by Shari Antor	Edition:	Paperback	Price:	$6.50	Availability:	Usually ships in 2 days
5	The Constant Princess	by Philippa Gr	Edition:	Hardcover	Price:	$16.47	Availability:	Usually ships in 24 hours
6	Silver May Tarnish (Witch Wo	by Andre Nort	Edition:	Hardcover	Price:	$16.47	Availability:	Usually ships in 24 hours
7	The Final Key : Part Two of T	by Catherine A	Edition:	Hardcover	Price:	$17.13	Availability:	Usually ships in 24 hours
8	The Midnight Work	by Kassandra	Edition:	Paperback	Price:	$6.99	Availability:	Usually ships in 2 days
9	Bindweed : A Gardening Myst	by Janis Harri	Edition:	Hardcover	Price:	$16.29	Availability:	Usually ships in 24 hours
10	Kingdom of Lies	by Lee Wood	Edition:	Hardcover	Price:	$16.47	Availability:	Usually ships in 24 hours

	I	J	K	L	M	
1	18 used & new	from	$1.30	1 of 3 people found the following review helpful:	terrific gothic romantic suspense thriller	, November 30, 2
2	4 used & new	from	$3.99	0 of 1 people found the following review helpful:	strong fifteenth century Scottish romance	, November 30, 2
3	2 used & new	from	$4.26	0 of 1 people found the following review helpful:	amusing contemporary romance	, November 30, 2
4	3 used & new	from	$5.79	0 of 1 people found the following review helpful:	enjoyable medieval romance	, November 30, 2
5				1 of 2 people found the following review helpful:	grand sweep of late fifteenth century ear	, November 30, 2
6	1 used & new	from	$16.00	0 of 1 people found the following review helpful:	return to Witch World	, November 30, 2
7	1 used & new	from	$17.00	1 of 2 people found the following review helpful:	excellent military sci fi thriller	, November 30, 2
8				7 of 10 people found the following review helpful:	fabulous paranormal tale	, November 30, 2
9	1 used & new	from	$16.04	0 of 1 people found the following review helpful:	terrific amateur sleuth	, November 30, 2
10	17 used & new	from	$12.25	0 of 1 people found the following review helpful:	interesting police procedural	, November 30, 2

Fig. 19. Data table with data items of the same attribute aligned in the same column.

6 Empirical Evaluations

This section evaluates our system, DEPTA (Data Extraction based on Partial Tree Alignment), which implements the proposed techniques. The evaluation consists of two parts:

1. Data record extraction (step 1): Given a single page, it extracts lists of data records (e.g., products and news) in it. We also compare this first step of DEPTA (called MDR-2), with our existing system MDR for identifying data records. We do not compare it with OMINI [4] and IEPAD [6] here as it is shown in [22] that MDR is already able to give better performance.
2. Data items alignment (step 2): This step aligns data items from the lists of data records identified in step 1. We compare this step of DEPTA with RoadRunner [9]. Note that RoadRunner does not perform the first step. It requires multiple input pages. Thus, to accommodate the requirement of RoadRunner, we transfer each data record in a data region of a list page to a separate HTML page. Therefore, each data region becomes a set of data record pages with similar patterns, which is fed to RoadRunner as input.

Experiment pages: The number of sites that we have used in our experiment is 142. The total number of pages used is 200. Seventy-two pages were collected by us randomly from the Web. The selection of pages is based on diversity. We want pages to cover a large number of different cases. The rest of the pages were given to us by a researcher from an Internet company (we could not disclose the name of the company and the name of the researcher for confidential reasons). We use these pages to show how our system works on pages not collected by us.

Tree distance threshold: We obtain our default distance threshold through pilot studies. The training pages were not used in testing our system. A normalized tree distance threshold $\tau = 0.4$ was set as the default value for the system and used in all our experiments.

Execution time: Our system is implemented in Visual C++. All the experiments were conducted on a Pentium 3, 1.0GHz PC with 256 MB RAM. The average execution time for a single page is less than 0.5 second excluding the time used to load the page into a browser, which depends on network traffics and the rendering engine.

TABLE 2
Experimental Results

Cate-gory	No. of Pages	Data Record Extraction							Data Item Alignment			
			MDR			DEPTA (MDR-2)			DEPTA			
		ACT	COR	WRG	MISS	COR	WRG	MISS	ACT	COR	WRG	MISS
1	108	1447	1447	0	0	1447	0	0	9498	9498	0	0
2	37	534	337	4	193	534	0	0	3380	3380	0	0
3	12	210	210	0	0	210	0	0	1416	1362	54	0
4	8	112	68	2	42	112	0	0	1344	1328	16	0
5	35	659	522	142	35	546	142	35	3372	3185	1278	187
Total	**200**	**2962**	**2584**	**148**	**270**	**2849**	**142**	**35**	**19019**	**18753**	**1348**	**187**
Recall			**87.24%**			**96.18%**			**98.60%**			
Precision			**94.55%**			**95.25%**			**93.29%**			

6.1 Experimental Results

Table 2 shows the results of DEPTA on data record extraction and data item alignment. The pages are divided into five categories according to experimental results and are listed in the first column.

For the 108 pages in Category 1, both MDR and DEPTA identiy all the data records correctly. DEPTA also aligns all the data items correctly.

For the 37 pages in Category 2, DEPTA identifies all the data records whereas MDR does not. Data items are aligned correctly by DEPTA.

For the 12 pages in Category 3, both MDR and DEPTA identify all the data records correctly, but a small number of items are incorrectly aligned by DEPTA.

For the eight pages in Category 4, DEPTA identifies all the data records whereas MDR does not. Some data items are aligned incorrectly.

For the 35 pages in Category 5, both MDR and DEPTA miss some data records or identify incorrect data records, and, consequently, their data items are either not extracted or extracted wrongly.

The two columns marked with "ACT" show the number of actual data records and the number of actual data items in each Web page, respectively. For comparison purposes, these numbers are manually counted. Note that we only count the data records which contain valued information. Some areas such as navigation bars in a page also contain data with regular patterns and they are also identified by our proposed technique. DEPTA takes into consideration the visual information of the identified data records to decide whether to output them or not. The criteria used to decide the importance of a list of data records is similar to the ones used in [30].

The three columns marked with "COR" show the numbers of correct data records extracted by MDR and DEPTA and the number of data items correctly aligned and extracted by DEPTA, respectively. The meaning of a data record is clear. The meaning of a data item needs some explanation. In this work, we assume that data items are segmented by HTML elements. Thus, a data item is correctly aligned and extracted if the text fragment is enclosed in an HTML element and aligned correctly with the same type of information in other data records. Note that data items may not be values of attributes as we recognize semantically. For example, for <b>Price: $20</b>, the extraction of "Price: $20" as one data item is considered correct if the alignment is also correct (explained below). Our system does not further split "Price" and "$20" into two items and assign "$20" as the value of the attribute "Price." As another example, for <b>Price:<b> <i>$20</i>, "Price" and "$20" are treated as two separate data items. We have not studied text segmentation and data labeling in this work. We plan to investigate these issues in the future.

A correct alignment also needs some explanation. Basically, we take the best result in the alignment of a set of data items. For example, there are five items representing the same type of information from five different data records, i.e., they should be aligned together. If all the five items are aligned (in one alignemnt), then they are all corrected. However, if they are not aligned together, errors are produced. For instance, two alignments (instead of one) are produced, one with three items and the other with two items, then the three items are aligned correctly, and the two items are not.

The three columns marked by "WRG" show the numbers of wrong data records extracted by MDR and DEPTA and the number of data items incorrectly aligned by DEPTA, respectively. For a data record, a wrong extraction means that only part of the content of the data record is extracted, or information outside of the data record boundary is extracted and enclosed in it. For a data item, incorrect alignment usually means items of the same attribute are placed into different columns, or items of different attributes are placed into the same column.

The three columns marked by "MISS" show the numbers of data records and data items that were not identified or extracted by MDR and DEPTA, respectively.

The last three rows of Table 2 give the total of each column, the *recall* and *precision* of each system. For data record extraction, the recall and precision are computed based on the total number of correct data records found in all pages and the actual number of data records in these pages. For data item alignment, the precision and recall computation has considered all wrongly extracted or missing data records introduced in step 1 of DEPTA.

The conflict resolution method discussed in Section 4.2 helps to improve the alignment of some data items in 30 pages out of 34 pages which have ambiguities in alignments. Also, there are about 7 percent of the sites that contain noncontiguous data records.

The comparison of data item alignment of DEPTA (the second step) and RoadRunner is shown in Table 3. For an accurate comparison, we do not consider the erroneous/missing records introduced by the first step of DEPTA (MDR-2). That is, we only use the data records that are correctly identified by MDR-2 in the comparison.

Out of the 200 pages, there are 174 pages from which all data items are aligned correctly by DEPTA and 26 pages from which 92 percent data items are aligned correctly. In comparison, there are only 110 pages from which all data items are aligned correctly by Road-Runner, 28 pages from which only 58 percent data items are aligned correctly, and 62 pages from which data items cannot be aligned by RoadRunner, which simply returns each data record as an item. There is no error reported by the system. We also ran the Tidy program

TABLE 3
Comparsion of Data Item Alignment of DEPTA and RoadRunner

	ACT	ALG	COR	Precision	Recall
DEPTA	18290	18290	18203	99.52%	99.52%
RoadRunner	18290	13241	10476	79.12%	57.28%

(http://www.w3.org/People/Raggett/tidy/) to clean each page as it was done in RoadRunner experiments.

The data item alignment results of the two systems are summarized in Table 3. "ALG" means aligned and "ACT" and "COR" have the same meanings as above. The results show that DEPTA outperforms RoadRunner significantly on both precision and recall. Note that the precision and recall are the same for DEPTA because all items are aligned (nothing lost).

6.2 Discussion

As it can be seen from Table 2, the DEPTA system is able to extract data records from the test pages and align the data items very accurately. However, there are still some errors that we sumamrize them below and give their causes.

There are 35 pages from which DEPTA failed to identify all data records correctly. The reasons are as follows: 1) When an object is very dissimilar to its neighboring objects, DEPTA misses it. This also causes a few identified data records to contain extra information or to miss part of their orignial data items. 2) The requirement that generalized nodes in each data region must have the same number of components also causes the system to miss data records. In our experiments, there is one page in which some generalized nodes are composed of different numbers of components. 3) DEPTA needs more than one similar object to be present in a data region. For example, given four similar objects in a page, where the first three are listed in a row and the fourth one is listed in a separate row, DEPTA will miss the fourth one because the second row is not sufficiently similar to the first row and the second row contains only a single object.

Incorrect alignment of data items happens in the following two situations: 1) Data items of the same attribute are incorrectly aligned into different columns because they are enclosed by tags with different tag names. 2) Data items of different attributes are incorrectly aligned into the same columns because they are enclosed by tags with the same tag name. Since data records are usually generated from templates, case 1 rarely happens. With the content-based similarity in the matching algorithm in Section 4.4, the problem in case 2 is alleviated to a great extent. However, some alignments need more than syntactic similarities. For example, "Made in USA" and "Imported," which have no common subsequence in terms of words, should be aligned together.

7 Conclusions

In this work, we proposed a new approach to extract structured data from Web pages. Although the problem has been studied by several researchers, existing techniques either are inaccurate or make several assumptions. Our method does not make these assumptions. It only requires that the page contains more than one data record. Our technique consists of two steps: 1) identifying data records without extracting data items in the records and 2) aligning corresponding data items from multiple data records and putting the data items in a database. We proposed an enhanced method based on visual cues for step 1. For step 2, we proposed a novel partial tree alignment technique to align corresponding data fields of multiple data records. Empirical results using a large number of Web pages demonstrated the effectiveness of the proposed technique.

References

[1] A. Arasu and H. Garcia-Molina, "Extracting Structured Data from Web Pages," *Proc. 2003 ACM SIGMOD Int'l Conf. Management of Data,* pp. 337-348, 2003.

[2] G.J. Barton and M.J. Sternberg, "A Strategy for the Rapid Multiple Alignment of Protein Sequences: Confidence Levels from Tertiary Structure Comparisons," *J. Molecular Biology,* vol. 198, no. 2, pp. 327-337, 1987.

[3] R. Baumgartner, S. Flesca, and G. Gottlob, "Visual Web Information Extraction with Lixto," *Proc. 27th Int'l Conf. Very Large Data Bases,* pp. 119-128, 2001.

[4] D. Buttler, L. Liu, and C. Pu, "A Fully Automated Object Extraction System for the World Wide Web," *Proc. 21st Int'l Conf. Distributed Computing Systems,* pp. 361-370, 2001.

[5] H. Carrillo and D. Lipman, "The Multiple Sequence Alignment Problem in Biology," *SIAM J. Applied Math.,* vol. 48, no. 5, pp. 1073-1082, 1988.

[6] C. Chang and S. Lui, "IEPAD: Information Extraction Based on Pattern Discovery," *Proc. 10th Int'l Conf. World Wide Web,* 2001.

[7] W. Chen, "New Algorithm for Ordered Tree-to-Tree Correction Problem," *J. Algorithms,* vol 40, no. 2, pp. 135-158, 2001.

[8] W.W. Cohen, M. Hurst, and L.S. Jensen, "A Flexible Learning System for Wrapping Tables and Lists in HTML Documents," *Proc. 11th Int'l Conf. World Wide Web,* pp. 232-241, 2002.

[9] V. Crescenzi, G. Mecca, and P. Merialdo, "Roadrunner: Towards Automatic Data Extraction from Large Web Sites," *Proc. 27th Int'l Conf. Very Large Data Bases,* pp. 109-118, 2001.

[10] D.W. Embley, Y. Jiang, and Y.K. Ng, "Record-Boundary Discovery in Web Documents," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 467-478, 1999.

[11] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of P-Completeness.* W.H. Freeman, 1979.

[12] G.H. Gonnet and R.B. Yates, *Handbook of Algorithms and Data Structures in Pascal and C.* Addison-Wesley, 1991.

[13] C.M. Hoffmann and M.J. O'Donnell, "Pattern Matching in Trees," *J. ACM,* pp. 68-95, 1982.

[14] P. Hogeweg and B. Hesper, "The Alignment of Sets of Sequences and the Construction of Phylogenetic Trees: An Integrated Method," *J. Molecular Evolution,* vol. 20, pp. 175-186, 1984.

[15] A. Hogue and D. Karger, "Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web," *Proc. 14th Int'l Conf. World Wide Web,* 2005.

[16] C.N. Hsu and M.T. Dung, "Generating Finite-State Transducers for Semistructured Data Extraction from the Web," *Information Systems,* vol. 23, no. 9, pp. 521-538, 1998.

[17] T. Jiang, L. Wang, and K. Zhang, "Alignment of Trees—An Alternative to Tree Edit," *CPM '94: Proc. Fifth Ann. Symp. Combinatorial Pattern Matching,* pp. 75-86, 1994.

[18] N. Kushmerick, "Wrapper Induction: Efficiency and Expressiveness," *Artificial Intelligence,* nos. 1-2, pp. 15-68, 2000.

[19] A.H.F. Laender, B.-R. Neto, and A.S. da Silva, "Debye—Date Extraction by Example," *Data and Knowledge Eng.,* vol. 40, no. 2, pp. 121-154, 2002.

[20] L. Arllota, V. Crescenzi, G. Mecca, and P. Merialdo, "Automatic Annotation of Data Extraction from Large Web Sitess," *Proc. Int'l Workshop Web and Databases,* pp. 7-12, 2003.

[21] K. Lerman, L. Getoor, S. Minton, and C. Knoblock, "Using the Structure of Web Sites for Automatic Segmentation of Tables," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 119-130, 2004.

[22] B. Liu, R. Grossman, and Y. Zhai, "Mining Data Records in Web Pages," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 601-606, 2003.

[23] I. Muslea, S. Minton, and C. Knoblock, "A Hierarchical Approach to Wrapper Induction," *Proc. Third Ann. Conf. Autonomous Agents,* pp. 190-197, 1999.

[24] C. Notredame, "Recent Progresses in Multiple Sequence Alignment: A Survey," technical report, Information Génétique et, 2002.

[25] D. Pinto, A. McCallum, X. Wei, and W.-B. Croft, "Table Extraction Using Conditional Random Fields," *Proc. 26th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* pp. 235-242, 2003.

[26] J. Raposo, A. Pan, M. Alvarez, J. Hidalgo, and A. Vina, "The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes," *Proc. 13th Int'l Workshop Database and Expert Systems Applications,* pp. 313-320, 2002.

[27] D.C. Reis, P.B. Golgher, A.S. Silva, and A.F. Laender, "Automatic Web News Extraction Using Tree Edit Distance," *Proc. 13th Int'l Conf. World Wide Web,* pp. 502-511, 2004.

[28] B. Rosenfeld, R. Feldman, and Y. Aumann, "Structural Extraction from Visual Layout of Documents," *Proc. 11th Int'l Conf. Information and Knowledge Management,* pp. 203-210, 2002.

[29] M.S. Selkow, "The Tree-to-Tree Editing Problem," *Information Processing Letters,* vol. 6, no. 6, pp. 184-186, 1977.

[30] R. Song, H. Liu, J.R. Wen, and W.Y. Ma, "Learning Block Importance Models for Web Pages," *Proc. 13th Int'l Conf. World Wide Web,* pp. 203-211, 2004.

[31] K.C. Tai, "The Tree-to-Tree Correction Problem," *J. ACM,* vol. 26, no. 3, pp. 422-433, 1979.

[32] E. Tanaka and K. Tanaka, "The Tree-to-Tree Editing Problem," *Int'l J. Pattern Recognition and Artificial Intelligence,* pp. 221-240, 1988.

[33] G. Valiente, "An Efficient Bottom-Up Distance between Trees," *Proc. Eighth Int'l Symp. String Processing and Information Retrieval,* pp. 212-219, 2001.

[34] J. Wang and F.H. Lochovsky, "Data Extraction and Label Assignment for Web Databases.," *Proc. 12th Int'l Conf. World Wide Web,* pp 187-196, 2003.

[35] W. Yang, "Identifying Syntactic Differences between Two Programs," *Software—Practice and Experience,* vol. 21, no. 7, pp. 739-755, 1991.

[36] Y. Zhai and B. Liu, "Web Data Extraction Based on Partial Tree Alignment," *Proc. 14th Int'l Conf. World Wide Web,* pp. 76-85, 2005.

[37] Y. Zhai and B. Liu, "Extracting Web Data Using Instance-Based Learning," *Proc. Sixth Int'l Conf. Web Information Systems Eng.,* 2005.

[38] K. Zhang, R. Statman, and D. Shasha, "On the Editing Distance between Unordered Labeled Trees," *Information Processing Letters,* vol. 42, no. 3, pp 133-139, 1992.

[39] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully Automatic Wrapper Generation for Search Engines," *Proc. 14th Int'l Conf. World Wide Web,* pp. 66-75, 2005.

[40] L. Zhao and N.K. Wee, "WICCAP: From Semi-Structured Data to Structured Data," *Proc. 11th IEEE Int'l Conf. and Workshop Eng. of Computer-Based Systems (ECBS '04),* p. 86, 2004.

Yanhong Zhai received the MS degree in computer science from the University of Illinois at Chicago. She is a PhD student in computer science at the University of Illinois at Chicago. Her research interests include information retrieval and machine learning.

Bing Liu received the PhD degree in artificial intelligence from the University of Edinburgh. He is an associate professor of computer science at the University of Illinois at Chicago (UIC). Before joining UIC in 2002, he was with the National University of Singapore. His research interests include data mining, Web mining, and text mining. He has published extensively in these areas. His recent works are mainly in the areas of Web data extraction and opinion mining. His recent professional activities include serving as a program chair of ACM CIKM-2006 conference, and also as a program chair of SDM-2007 (SIAM Data Mining conference). He has served or serves on numerous conference program committees related to data mining, Web mining, and natural language processing, and is (was) on the editorial boards of several journals, including the *IEEE Transactions on Knowledge and Data Engineering*. Apart from research, he also has extensive application experiences. Recently, his group built a major and unique data mining system for Motorola Inc., which has been deployed and is in daily use there for finding interesting and actionable knowledge from Motorola's engineering and service data sets. More information about him can be found at http://www.cs.uic.edu/~liub.