Web Data Extraction Based on Partial Tree Alignment

Yanhong Zhai
Department of Computer Science
University of Illinois at Chicago
851 South Morgan Street,
Chicago, IL 60607-7053
yzhai@cs.uic.edu

Bing Liu
Department of Computer Science
University of Illinois at Chicago
851 South Morgan Street,
Chicago, IL 60607-7053
liub@cs.uic.edu

ABSTRACT

This paper studies the problem of extracting data from a Web page that contains several structured data records. The objective is to segment these data records, extract data items/fields from them and put the data in a database table. This problem has been studied by several researchers. However, existing methods still have some serious limitations. The first class of methods is based on machine learning, which requires human labeling of many examples from each Web site that one is interested in extracting data from. The process is time consuming due to the large number of sites and pages on the Web. The second class of algorithms is based on automatic pattern discovery. These methods are either inaccurate or make many assumptions. This paper proposes a new method to perform the task automatically. It consists of two steps, (1) identifying individual data records in a page, and (2) aligning and extracting data items from the identified data records. For step 1, we propose a method based on visual information to segment data records, which is more accurate than existing methods. For step 2, we propose a novel partial alignment technique based on tree matching. Partial alignment means that we align only those data fields in a pair of data records that can be aligned (or matched) with certainty, and make no commitment on the rest of the data fields. This approach enables very accurate alignment of multiple data records. Experimental results using a large number of Web pages from diverse domains show that the proposed two-step technique is able to segment data records, align and extract data from them very accurately.

Categories and Subject Descriptors

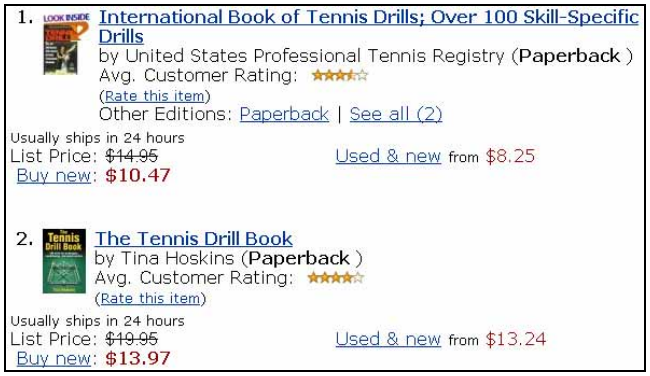
H.3.m [Information Storage and Retrieval]: Miscellaneous – Data Extraction, Wrapper Generation, Web

General Terms: Algorithms, Experimentation.

Keywords: Data extraction, wrapper, data record extraction.

1. INTRODUCTION

Structured data objects are a very important type of information on the Web. Such data objects are often records from underlying databases and displayed in Web pages with some fixed templates. In this paper, we also call them *data records*. Mining data records in Web pages is useful because they typically present their host pages' essential information, such as lists of products and services. Extracting these structured data objects enables one to integrate data/information from multiple Web pages to provide value-added services, e.g., comparative shopping, meta-querying and search. Figure 1 gives some example data records on the Web. Figure 1(A) shows a Web page segment containing a list of two products (books). The description of each book is a data record. Figure 1(B) shows a page segment containing a data table, where each data record is a table row. Our objective is twofold: (1) automatically identify such data records in a page, and (2) automatically align and extract data items from the data records.



(A) A list of products

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) A data table

Figure 1: Two example page segments with data records

Several approaches have been reported in the literature for mining data records from Web pages. The first approach is the manual approach. By observing a Web page and its source code, the programmer finds some patterns from the page and then writes a program to identify and extract all the data items/fields. This approach is not scalable to a large number of pages. Other approaches all have some degree of automation. There are two main types of algorithms, wrapper induction and automatic extraction. In wrapper induction [11, 19, 23, 25, 33], a set of extraction rules are learnt from a set of manually labeled pages or data records. These rules are then used to extract data items from similar pages. This method still requires substantial manual efforts. In automatic methods, [12][1] find patterns or grammars from multiple pages containing similar data records. Requiring an initial set of pages containing similar data records is, however, a major limitation of this type of approaches because such pages

WWW 2005, May 10-14, 2005, Chiba, Japan.
ACM 1-59593-046-9/05/0005.

have to be found manually or by another system. [20] proposes a method that tries to explore the detailed information page behind the current page to extract data records. The need for detailed information pages behind is also a serious limitation because many data records do not have such pages behind (e.g., Figure 1(B)). Furthermore, the method assumes that the detail pages are given, which is not realistic in practice. Due to a large number of links in a typical Web page, automatically identifying links that point to detailed information pages is a non-trivial task. [8] proposes a string matching method. However, its results are weak as shown in [21]. Another assumption that most current systems make is that the relevant information 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, part2 of object2. Thus, the descriptions of both object1 and object2 are not contiguous. However, when they are displayed on a browser, they appear contiguous to human viewers. In Section 2, we discuss these methods in detail and compare with our proposed approach.

This paper proposes a two-step strategy to solve the problem.

1. Given a page, the method first segments the page to identify each data record without extracting its data items. We have improved our previous technique MDR [21] for this purpose. Specifically, the new method also uses visual cues to find data records. Visual information helps the system in two ways:

   (i) It enables the system to identify gaps that separate data records, which helps to segment data records correctly because the gap within a data record (if any) is typically smaller than that in between data records.

   (ii) The proposed system identifies data records by analyzing HTML tag trees or DOM trees [7]. A straightforward way to build a tag 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 or display 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 the HTML code, visual information (i.e., the locations on the screen at which tags are rendered) is utilized to infer the structural relationship among tags and to construct a tag 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 the browser is able to render a page correctly, its tag tree can be built correctly.

2. A novel partial tree alignment method is proposed to align and to extract 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 HTML code. This new method is very accurate as our experiments show.

   Specifically, after all data records have been identified, the sub-trees of each data record are re-arranged into a single tree as each data record may be contained in more than one sub-tree in the original tag tree of the page, and each data record may not be contiguous. The tag trees of all the data records are then aligned using our partial alignment method. By partial alignment, we mean that for each pair of trees (or data records), we only align those data fields that can be aligned with certainty and ignore those parts that cannot, i.e., 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 data items from all data records in the page. It can also serve as an extraction pattern to be used to extract data items from other pages with data records generated using the same template.

Our two-step approach called DEPTA (Data Extraction based Partial Tree Alignment), which is very different from all existing methods, does not make those assumptions made by existing methods. As long as a page contains at least two data records, our system will automatically find them (see Section 3.5 for more discussion). Our experimental results using a large number of pages show that the proposed technique is highly effective.

2. RELATED WORK

Related works to ours are in the area of wrapper generation. A wrapper is a program that extracts data from a Web site or page and put them in a database [1, 11, 12, 16, 18, 19, 22, 23, 25]. There are two main approaches to wrapper generation.

The first approach is wrapper induction, which uses supervised learning to learn data extraction rules from a set of manually labeled positive and negative examples. Manual labeling of data is, however, labor intensive and time consuming. Additionally, for different sites or even pages in the same site, the manual labeling process needs to be repeated because they follow different templates/patterns. Example wrapper induction systems include WIEN [19], Softmealy [18], Stalker [23], WL^2 [11], [25], etc. Our technique requires no human labeling. It mines data records in a page and extracts data from the records automatically.

The second approach is automatic extraction. In [14], a study is made to automatically identify data record boundaries. The method is based on a set of heuristic rules, e.g., highest-count tags, repeating-tags and ontology-matching. [5] proposes a few more heuristics to perform the task without using domain ontology. However, [21] shows that these methods produce poor results. In addition, these methods do not extract data from data records.

[8] proposes a method 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 and sequence alignment to find inexact matches. However, [21] shows that its performance is also weak. Our new method does not use tag strings for alignment but trees, which exploits nested tree structures to perform much more accurate data extraction. [13] also gives a set of heuristics to find individual product information, e.g., price and others.

In [1, 12, 34], two more techniques are proposed. However, they need to use multiple pages (which are assumed to be given) that contain similar data records from the same site to find patterns or grammars from the pages to extract data records. Assuming the availability of multiple pages containing similar data records is a serious limitation. Our method works on each single page.

[20] proposes another method for data extraction. Its main idea is to utilize the detailed data in the page behind the current page to identify data records. It is common that a page with multiple data records does not contain the complete information of each data

record. Instead, a link is normally used to point to the page with complete details. For example, a product record normally has a link pointing to the page that contains the detailed description of the product. The technique is thus applicable to the example in Figure 1(A), but not to the one in Figure 1(B) because each data record in Figure 1(B) has no link to a detail page. Furthermore, the method in [20] assumes that detail pages are given (in their experiments such pages are manually identified), which is not realistic. Due to a large number of links in a typical Web page, automatically identifying the correct links that point to detail pages is not a trivial task. Our technique is applicable to both types of pages in Figure 1 as it does not require any detail page.

Another problem with most existing approaches is that they assume that the relevant information of a data record is contained in a contiguous segment of the HTML code. This is not always true. This issue has been discussed in the Introduction section. The proposed method is able to handle this situation because our record segmentation method is able to identify such data records.

In [21], we propose the MDR algorithm, which only identifies data records but does not align or extract data items from the data records. Thus, it only performs the first step of our task. Even for the first step, it has two main shortcomings. (1) The algorithm makes use of the HTML tag tree of the Web page to extract data records from the page. However, erroneous tags in the HTML source of some pages make it hard to build correct trees, which make it impossible to find correct data records in these pages. Using visual (rendering) information to build trees in our new system solves this problem. (2) A single data record may be composed of multiple sub-trees. Due to noisy information, MDR may find wrong combinations of sub-trees. In our new system, visual gaps between data records help to deal with this problem. Note that visual cues have been used in other Web tasks, e.g., finding different semantics blocks [29, 28].

Finally, tree matching has been used for finding the main contents in news pages in [27]. However, their task is different from ours.

3. DATA RECORD EXTRACTION

We now start to present our proposed technique. This section focuses on the first step: segmenting 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 [21], 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).

3.1 The Basic Idea of MDR

The MDR algorithm is based on two observations about data records in a Web page and an edit distance string matching algorithm [2] to find data records. The two observations are:

1. A group of data records that contains descriptions of a set of similar objects are typically presented 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 Figure 1(A) two books are presented in one contiguous region. They are also formatted using almost the same sequence of HTML tags. If we regard the HTML tags of a page as a long string, we can use string matching (e.g., edit distance [2]) to compare different sub-strings to find those 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. A set of data records typically does not have the same length in terms of its tag strings because it may not contain exactly the same pieces of information (see Figure 1(A)). The next observation helps to deal with this problem.

2. The nested structure of HTML tags in a Web page naturally forms a *tag tree*. For example, Figure 2 shows an example tag tree. In this tree, each data record is wrapped in 3 TR nodes with their sub-trees under the same parent TBODY. The two data records are in the two dash-lined boxes. Our second observation is that a set of similar data records are formed by some child sub-trees of the same parent node.

   It is unlikely that a data record starts in the middle of a child sub-tree and ends in the middle of another child sub-tree. Instead, it starts from the beginning of a child sub-tree and ends at the end of the same or a later child sub-tree. For example, it is unlikely that a data record starts from TD* and ends at TD# (Figure 2). This observation makes it possible to design a very efficient algorithm based on edit distance string comparison to identify data records because it limits the tags from which a data record may start and end in a tag tree. [1]

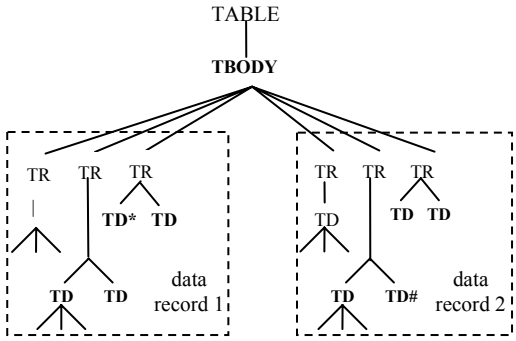


Figure 2: An example tag tree of a page segment

Experiments show that these observations work very well. By no means do we assume that a Web page has only one data region that contains data records. In fact, a Web page may contain a few data regions. Different regions may have different data records.

Given a Web page, the algorithm works in three steps (we also discuss the enhancements made to MDR in our current work):

Step 1: Building a HTML tag tree of the page. In the new system, visual (rendering) information is used to build the tag tree.

Step 2: Mining data regions in the page using the tag 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, MDR mines data regions first and then finds data records within them. For example, in Figure 2, we first find the single data region below node TBODY. In our new system, again visual information is used in this step to produce better results.

Step 3: Identifying data records from each data region. For example, in Figure 2, 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

[1] We may also use tree edit distance. However, since string edit distance already works very well, we did not use tree edit distance for this step.

information to help building more robust trees and also to find more accurate data regions. We describe them below.

3.2 Building a HTML Tag Tree

In a Web browser, each HTML element (consisting of a start *tag*, optional attributes, optional embedded HTML content, and an end tag that may be omitted) is rendered as a rectangle. A tag tree can be constructed based on the nested rectangles (resulted from nested tags). The details are as follows:

1. Find the 4 boundaries of the rectangle of each HTML element by calling the embedded parsing and rendering engine of a browser, e.g., Internet explorer.
2. Detect the containment relationship among the rectangles, i.e., whether one rectangle is contained inside another rectangle. A tree can be built based on the containment check.

Let us use an example to illustrate the process. Assume we have the HTML code on the left of Figure 3, which is a table with two rows (tr's) and each row with two cells (td's). The rendering engine of the browser produces the boundary coordinates (in pixels) for each HTML element shown on the right of Figure 3.

		left	right	top	bottom
1	<table>	100	300	200	400
2	<tr>	100	300	200	300
3	<td> … </td>	100	200	200	300
4	<td> … </td>	200	300	200	300
5	</tr>				
6	<tr>	100	300	300	400
7	<td> … </td>	100	200	300	400
8	<td> … </td>	200	300	300	400
9	</tr>				
10	</table>				

Figure 3: A HTML code segment and boundary coordinates

With the visual information, we can build the tree in Figure 4 by following the sequence of opening tags and by containment checks. The tree construction algorithm is fairly straightforward. We will not discuss it further here.

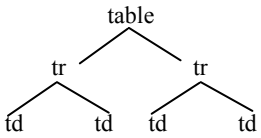


Figure 4: Tag tree for the HTML code in Figure 3

3.3 Mining Data Regions

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 data regions. By comparing tag strings of individual nodes (including their descendents) and combination of multiple adjacent nodes, we can find each data region.

We use an artificial tag tree in Figure 5 to explain. We find that nodes 5 and 6 are similar (based on edit distance) and form the data region labeled 1, nodes 8, 9 and 10 are similar and form the data region labeled 2, and the pairs of nodes (14, 15), (16, 17) and (18, 19) are similar and form the data region labeled 3. To avoid using both individual nodes and node combinations, we use the concept of the *generalized node* to denote each similar individual (tag) node and each (tag) node combination. Thus, a sequence of adjacent generalized nodes forms a data region. Each shaded individual node or node combination in Figure 5 is a generalized node. The concept of generalized node captures the situations that a data record may be contained in a few sibling tag nodes rather than one and that data records may not be contiguous in the tag tree, but generalized nodes are contiguous (see below).

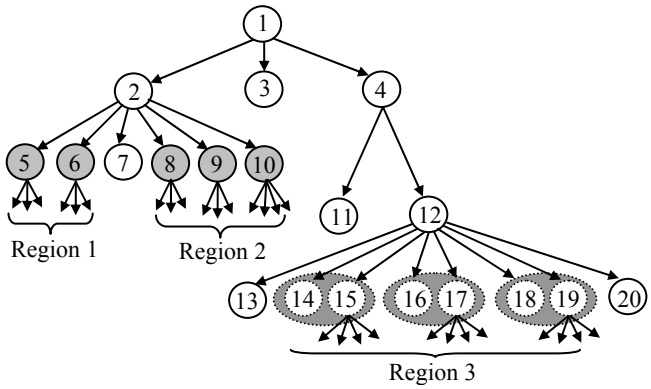


Figure 5: An illustration of generalized nodes & data regions

Due to the observation in Section 3.1, the number of string comparisons to find generalized nodes for identifying data regions is not very large. We only need to perform comparisons among the children nodes of a parent node. The process of identifying data regions is involved; see [21] for more details.

In our new system, gaps between data records are used to eliminate false node combinations. We utilize the following visual observation about data records:

- The gap between two data records in a data region should be no smaller than any gap within a data record. For example, in Figure 1(A), a large gap exists between the two data records.

3.4 Identifying Data Records

After all data regions are identified, we identify data records from generalized nodes. We note that each generalized node (a single or a combination of tag nodes in the tag tree) may not represent a single data record. The situations can be quite complex. Below, we only highlight two interesting cases in which a data record is not contained in a contiguous segment of the HTML code in order to show some advanced capabilities of our system (see [21] for more details and other simpler cases).

3.4.1 Non-contiguous Data Records: Case 1

In some Web pages, the description of an object (a data record) is not in a contiguous segment of the HTML code. There are two main cases. Figure 6 shows an example of the first case.

In this example, the data region contains two generalized nodes, and each generalized node contains two tag nodes (two rows), which indicates that these two tag nodes (rows) are not similar to each other. But each tag node has the same number of children nodes and the children nodes are similar to each other. One row lists the names of the two objects in two cells, and the next row lists the other pieces of information of the objects also in two cells. This results in the HTML code: name 1, name 2, description 1, description 2, name 3, name 4, description 3, description 4.

For this kind of situation, the corresponding children nodes of every tag node in a generalized node form a non-contiguous data record. This is illustrated by the tag tree at the bottom of Figure 6,

where *r* represents *row*, *n* represents *name* and *d* represents *description*. G1 and G2 are generalized nodes. (n1, d1), (n2, d2), (n3, d3), and (n4, d4) form four data records.

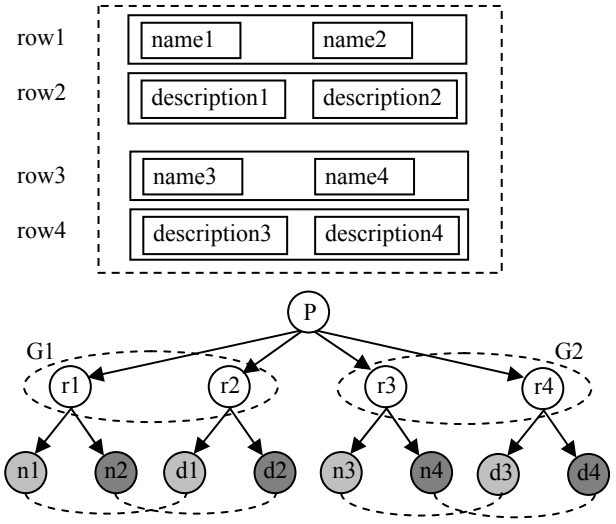


Figure 6: A multiple-record data region: each generalized node contains more than one non-contiguous data record

3.4.2 Non-contiguous Data Records: Case 2

Figure 7 shows an example of the second case, where two or more data regions form multiple data records. In this example, row 1 and row 2 are not similar to each other, but row 1 forms a data region and row 2 forms another data region. Each data region contains two (small) generalized nodes.

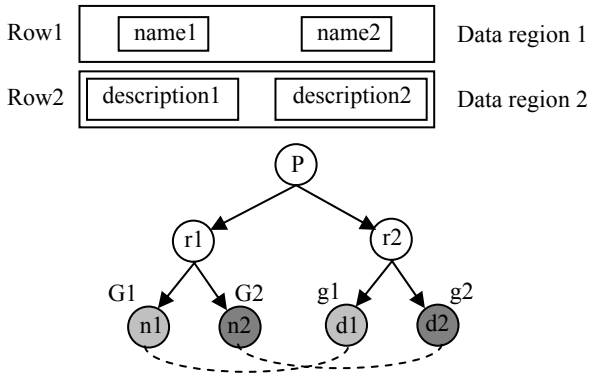


Figure 7: Adjacent data regions form more than one non-contiguous data records

From the tag tree in Figure 7, we see that this case has the same structure as the one in Figure 6. Therefore a similar strategy can be applied here, i.e., the corresponding generalized nodes of each data region are joined together to form non-contiguous data records. This process is illustrated by the tag tree in Figure 7 (G1, G2, g1 and g2 are generalized nodes).

3.5 An Important Note on Data Records

Finally, it is important to note that MDR or MDR-2 does not know what regular data records are useful to a user. It simply finds all of them. However, in a particular application, the user is usually interested in only a specific type of data records, e.g., a list of products, or data tables. Simple heuristics can be designed to output only the required type of data records. For example, in MDR (or MDR-2), as an option it can output only product data records based on some indicators, e.g., image, price and others.

4. DATA EXTRACTION

We now present the partial tree alignment technique for data extraction. The key task is how to match corresponding data items or fields from all data records. There are two sub-steps:

1. Produce one rooted tag tree for each data record: After all data records are identified, the sub-trees of each data record are re-arranged into a single tree. As shown above, each data record may be contained in more than one sub-tree of the original tag tree of the page, and each data record may not be contiguous. Thus, this sub-step is needed to compose a single tree for each data record (an artificial root node may also need to be added). We will not discuss this further as it is fairly simple.
2. Partial tree alignment: The tag trees of all data records in each data region are aligned using our partial alignment method which is based on tree matching. It should be noted that in the matching process, we only use tags. No data item is involved.

Below, we first give a brief introduction to tree edit distance or tree matching and then present a restricted tree matching method that we use in this work. After that we will discuss multiple alignments and present the partial tree alignment method for aligning multiple data records based on their tag trees.

We note here that string edit distance is not suitable for this step 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 tr's and td's. After string matching, it is hard to decide which alignment is the correct one as there are many possible alignments. However, tree matching significantly reduces the number of possible alignments because of the tree structure constraint. In our algorithm, we only use one simple rule to resolve conflicts when there is more than one possible tree alignment. We simply choose the possible sub-tree alignment that appears the earliest in the tree. This method works quite well in our experiments. Thus, we did not design more sophisticated conflict resolution strategies.

4.1 Tree Edit Distance

Similar to string edit distance, tree edit distance [31, 30] between two trees *A* and *B* (we are only interested in *labeled ordered rooted trees*) is the cost associated with the minimum set of operations needed to transform *A* into *B*. In the classic formulation, the set of operations used to define tree edit distance includes three operations: node removal, node insertion, and node replacement. A cost is usually assigned to each of the operations. Solving the tree edit distance problem is often assisted by finding a minimum-cost *mapping* between two trees [30]. The concept of mapping [30] is formally defined as:

Let X be a tree and let $X[i]$ be the ith node of tree X in a preorder walk of the tree. A *mapping* M between a tree A of size n_1 and a tree B of size n_2 is a set of ordered pairs (i, j), one from each tree, satisfying the following conditions for all (i_1, j_1), $(i_2, j_2) \in M$:

(1) $i_1 = i_2$ *iff* $j_1 = j_2$;
(2) $A[i_1]$ is on the left of $A[i_2]$ *iff* $B[j_1]$ is on the left $B[j_2]$;
(3) $A[i_1]$ is an ancestor of $A[i_2]$ *iff* $B[j_1]$ is an ancestor of $B[j_2]$.

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. Figure 8 shows a mapping example.

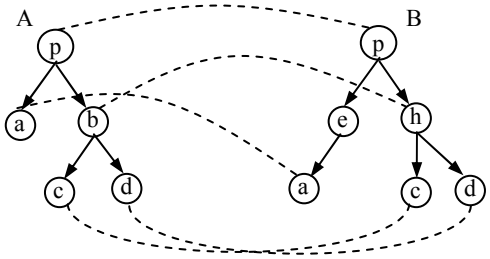


Figure 8: A general tree mapping example

Several algorithms have been proposed to address the problem of finding the minimum set of operations (i.e., the one with the minimum cost) to transform one tree into another. All the formulations have complexities above quadratic [10]. It has also been shown that if the trees are not ordered, the problem is NP-complete [36]. In [30], a solution based on dynamic programming is presented. The algorithm has a complexity of $O(n_1 n_2 h_1 h_2)$, where n_1 and n_2 are the sizes of the trees and h_1 and h_2 are heights of the trees. In [32][10], two other algorithms are also presented with similar complexities.

4.2 Simple Tree Matching

In the above general setting, mapping can cross levels, e.g., node *a* in tree *A* and node *a* in tree *B*. There is also replacement, e.g., node *b* in *A* and node *h* in *B*. In this work, we use a restricted matching algorithm [35], which was first proposed to compare two computer programs in software engineering. It was called *simple tree matching* (STM). STM evaluates the similarity of two trees by producing the maximum matching through dynamic programming with complexity $O(n_1 n_2)$, where n_1 and n_2 are the sizes of trees *A* and *B* respectively. No node replacement and no level crossing are allowed.

Let *A* and *B* be two trees and $i \in A$, $j \in B$ are two nodes in *A* and *B* respectively. A *matching* between two trees is defined to be a mapping *M* such that for every pair $(i, j) \in M$ where *i* and *j* are non-root nodes, (parent(*i*), parent(*j*)) $\in M$. A *maximum matching* is a matching with the maximum number of pairs.

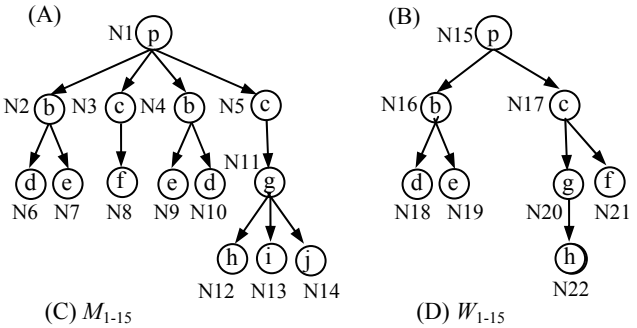
Let $A = <R_A, A_1, A_2, \ldots, A_m>$ and $B=<R_B, B_1, B_2, \ldots, B_n>$ be two trees, where R_A and R_B are the roots of *A* and *B*, and A_i, B_j are the *i*th and *j*th first-level sub-trees of *A* and *B* respectively. When R_A and R_B contain identical symbols, the maximum matching between *A* and *B* is $M_{A,B}+1$, where $M_{A,B}$ is the maximum matching between $<A_1, A_2, \ldots, A_m>$ and $<B_1, B_2, \ldots, B_n>$. $M_{A,B}$ can be obtained by the following dynamic programming scheme:

1. If the maximum matching between A_m and B_n, is larger than any maximum matching between A_m and B_i ($1 \le i < n$), then $M_{A,B}$ is the maximum matching between $<A_1, A_2, \ldots, A_{m-1}>$ and $<B_1, B_2, \ldots, B_{n-1}>$ plus the maximum matching between A_m and B_n.
2. Otherwise, $M_{A,B}$ is the same as the maximum matching between $<A_1, A_2, \ldots, A_m>$ and $<B_1, B_2, \ldots, B_{n-1}>$, or between $<A_1, A_2, \ldots, A_{m-1}>$ and $<B_1, B_2, \ldots, B_n>$.

In the Simple_Tree_Matching algorithm in Figure 9, the roots of *A* and *B* are compared first (line 1). If the roots contain distinct symbols, then the two trees do not match at all. If the roots contain identical symbols, then the algorithm recursively finds the maximum matching between first-level sub-trees of *A* and *B* 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 *A* and *B*.

Algorithm: Simple_Tree_Matching(*A*, *B*)
1. **if** the roots of the two trees *A* and *B* contain distinct symbols
2. **then return** (0);
3. **else** *m*:= the number of first-level sub-trees of *A*;
4. *n*:= the number of first-level sub-trees of *B*;
5. Initialization: $M[i, 0] := 0$ for $i = 0, \ldots, m$;
   $M[0, j] := 0$ for $j = 0, \ldots, n$;
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 $W[i,j]$ = Simple_Tree_Matching(A_i, B_j)
9. **endfor;**
10. **endfor;**
11. **return** ($M[m, n]+1$)
12. **endif**

Figure 9. The simple tree matching algorithm



(C) $M_{1\text{-}15}$

	0	1 (N16)	2 (N16-N17)
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

(D) $W_{1\text{-}15}$

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

(E) $M_{5\text{-}17}$

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

(F) $W_{5\text{-}17}$

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

(G) $M_{11\text{-}20}$

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

(H) $W_{11\text{-}20}$

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

Figure 10. (A) Tree *A*; (B) Tree *B*; (C) *M* matrix for the first level sub-trees of N1 and N15; (D) *W* matrix for the first level sub-trees of N1 and N15; (E)-(H) *M* matrixes and *W* matrixes for the lower level sub-trees.

We use an example from [35] to explain the algorithm (Figure 10). To find the maximum matching between trees *A* and *B*, their roots, N1 and N15, are compared first. Since N1 and N15 contain

identical symbols, $M_{1\text{-}15}[4,2]+1$ is returned as the maximum matching value between trees A and B (line 11). $M_{1\text{-}15}$ matrix is computed based on the $W_{1\text{-}15}$ matrix, and each entry in $W_{1\text{-}15}$, say $W_{1\text{-}15}[i, j]$, is the maximum matching between the ith and jth first-level sub-trees of A and B, which is computed recursively based on its M matrix. For example, $W_{1\text{-}15}[4, 2]$ is computed recursively by building the matrices (E)-(H). All the relevant cells are shaded. The zero column and row 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.

During the matching (or after the matching), we can trace back in the M matrices to find the matched/aligned nodes from the two trees. When there is more than one match for a node that gives the maximum result, we choose the one that appear the earliest in the tree. For example, in Figure 11, node c in tree A can match either the first or the last node c in tree B. We choose the first node c in B. This heuristics is used because for visual effectiveness in a Web page, if an earlier node x in tree A is to match a later node y in tree B, there is usually some indication (tags) before x. This heuristic works well according to our experiments.

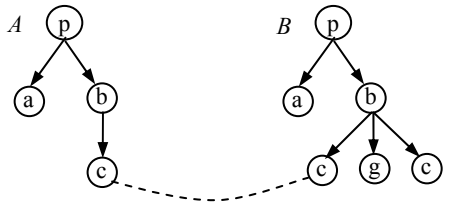


Figure 11. Two trees with more than one possible match

4.3 Multiple Alignment

Since each data region in a page contains multiple data records, we need to align multiple tag trees in order to produce a single database table with all the corresponding data items/fields in the same column of the table. In this data table, each row represents a tree (data record), and each column represents a data field in each data record. Several existing algorithms can perform assignment of multiple sequences/trees. In [6], 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, 17, 3]. Center string method, which is used in [8], 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 ($D(x_i, x_c)$ is the distance of two strings)

$$\sum_{i=0}^{k} D(x_i, x_c)$$

is selected as the center. 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 and each step of the iterative pair-wise alignment takes $O(n^2)$ time. Therefore the overall time cost is $O(k^2n^2)$. 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: Firstly, although the algorithm has a polynomial time complexity, it runs slowly for pages containing many data records or data records containing many attributes. Secondly, if the center tree does not have a particular data item, other data records that contain the same data item will not be aligned. We implemented the method and the results were poor. Other popular multiple alignment methods include progressive alignment [17] and iterative alignment [3]. They work like hierarchical clustering, and all require upfront $O(k^2)$ pair-wise matching. For our task, we can do better because we know that data records follow some predefined template.

4.4 Partial Tree Alignment

Our proposed approach aligns multiple tag trees by progressively growing a seed (tag) tree. The seed tree, denoted by T_s, is initially picked to be the tree with the maximum number of data fields. Note that the seed tree is similar to the center tree but without the $O(k^2)$ pair-wise tree matching to choose it. 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 n_i, a link is created from n_i to n_s to indicate its match in the seed tree. If no match can be found for node n_i, then the algorithm attempts to expand the seed tree by inserting n_i into T_s. The expanded seed tree T_s is then used in subsequent matching. Note that data items in the tag tree nodes are not used during matching or alignment.

4.4.1 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 of 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 n_i from T_i into the seed tree T_s, depending on whether a location in T_s can be uniquely determined to insert n_i. In fact, instead of considering a single node n_i, we can consider each set of unmatched consecutive sibling nodes $n_j \ldots n_m$ from T_i together. Without loss of generality, we assume that the parent node of $n_j \ldots n_m$ has a match in T_s and we want to insert $n_j \ldots n_m$ into T_s under the same parent node. We only insert $n_j \ldots n_m$ into T_s if a position for inserting $n_j \ldots n_m$ can be uniquely determined in T_s. Otherwise, they will not be inserted into T_s and left unaligned. The alignment is thus partial. The location for insertion of $n_j \ldots n_m$ can be uniquely decided:

1. if $n_j \ldots n_m$ 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. Figure 12(A) 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 Figure 12(A). 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 $n_j \ldots n_m$ has only one left neighboring sibling x in T_i and x matches the right most node x in T_s, then $n_j \ldots n_m$ can be inserted after node x in T_s. Figure 12(B) illustrates this case.
3. if $n_j \ldots n_m$ has only one right neighboring sibling x in T_i and it matches the left most node x in T_s, then $n_j \ldots n_m$ can be inserted before node x in T_s. This case is similar to above.

Otherwise, we cannot uniquely decide a location for unmatched nodes in T_i to be inserted into T_s. This is illustrated in Figure 12(C). In this case, the unmatched node x in T_i could be inserted into T_s in two positions, between nodes a and b, or between node b and e in T_s. In this situation, we will not insert it into T_s.

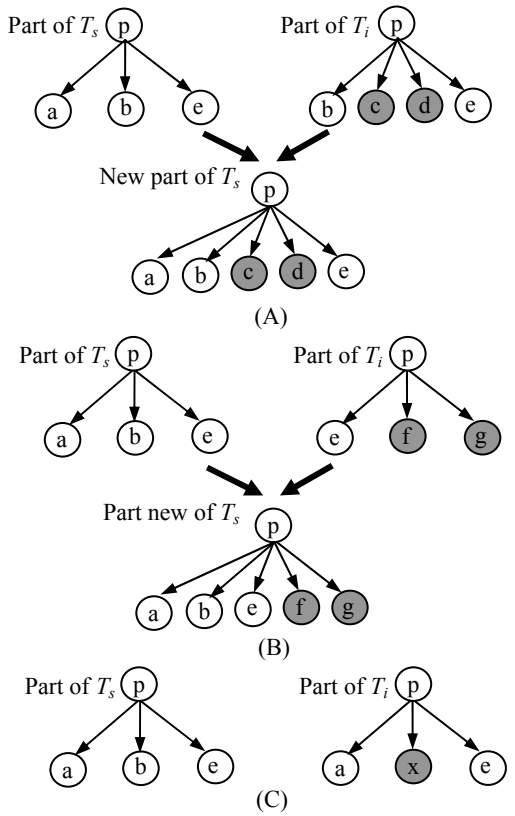


Figure 12. Expanding the seed tree: (A) and (B) Unique expansion; (C) Insertion ambiguity.

4.4.2 Full algorithm

Figure 13 gives the full algorithm for multiple tree alignment based on partial alignment of two tag trees.

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; $R = \varnothing$; *I* = false;
4. **while** ($S \neq \varnothing$)
5. T_i = select and delete next tree from *S*;
6. Simple_Tree_Matching(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 *R*;
12. **endif**;
13. **endif**;
14. **if** (*L* has new alignment) or (*I* is true) **then**
15. *flag* = true
16. **endif**;
17. **if** $S = \varnothing$ and *flag* = true **then**
18. $S = R$; $R = \varnothing$;
19. *flag* = false; *I* = false
20. **endif**;
21. **endwhile;**
22. Output data fields from each T_i to the data table based on the alignment results.

Figure 13. The partial tree alignment Algorithm.

We use a simple example in Figure 14 to explain the algorithm. We have three example trees, all of which have only two levels.

Lines 1 and 2 (Figure 13) basically find the tree with the most data items. This is the seed tree. In Figure 14, 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. We will not discuss this further. Note that line 5 and line 6 can be integrated. We present them separately for simplicity. In Figure 14, T_s and T_2 produce one match, node *b*. Nodes *n*, *c*, *k* and *g* are not matched to T_s. Line 8 checks that. Line 9 attempts to insert them into T_s. This is the partial tree alignment discussed above. In Figure 14, 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 14 inserts T_2 into *R*, which is a list of trees that may need to be further processed. In Figure 14, 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 14-16 set "*flag* = true" to indicate that some new alignments/matches are found or some unmatched nodes are inserted into T_s.

Lines 17-21 check for stopping conditions. "$S = \varnothing$ and *flag* = 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 Figure 14, T_2 is the only tree in *R*, which will be matched to the new T_s in the next round. Now every node in T_2 can be matched or inserted. The process completes. Line 23 outputs the data items from each tree according to the alignment produced. Note that if there are still un-matched nodes with data after the algorithm completes, each un-matched data will occupy a single column by itself. Table 1 shows the data table for the trees in Figure 14. We use "1" to indicate a data item.

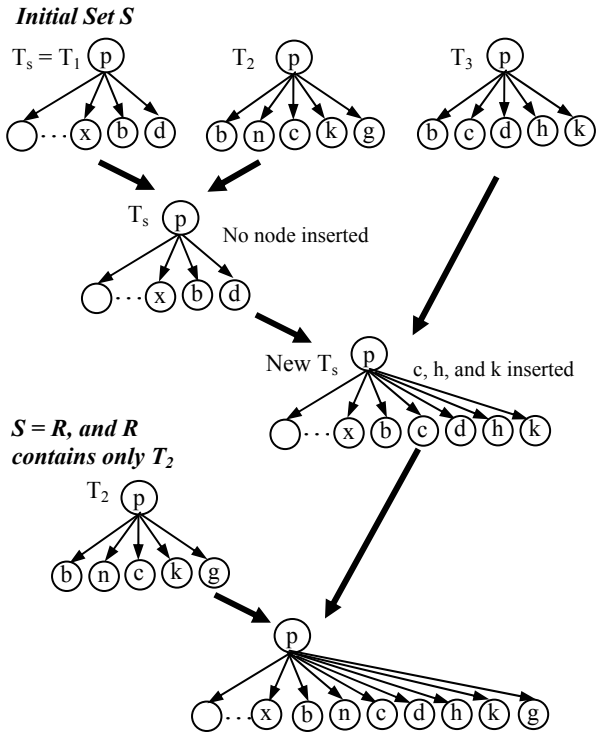


Figure 14. Iterative tree alignment with two iterations.

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	

The complexity of the algorithm is $O(k^2)$ without considering tree matching, where k is the number of trees. However, in practice, almost always we only need to go though S once (i.e., $R = \varnothing$).

It should be noted that the resulting alignment T_s can also be used as an extraction pattern for extracting data items from other pages generated using the same template.

5. EMPIRICAL EVALUTIONS

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): We compare the first step of DEPTA (also called MDR-2), with our existing system MDR for identifying data records. We do not compare it with the method in [5] and the method in [8] here as it is shown in [21] that MDR is already more effective than them.
2. Data items/fields alignment and extraction (step 2): This is the second step of DEPTA. [8] is able to perform the same task. However, as shown in [21], it performs poorly in finding right data records, and thus could not extract data items well. We do not compare with the systems in [1][12] as they require multiple pages and all of them contain similar data records to find patterns from the pages to extract data items. The technique in [20] requires the detail page behind the page (to be extracted), and in their experiments, such detail pages are manually identified and downloaded, which is unrealistic in practice. DEPTA is more general. Given a single page, it is able to extract data records and data items from it.

Our experimental results are given in Table 2.

Column 1: It lists the URL of each site. In some sites more than one page are tried (which have different data record formats). The number of sites that we have used in our experiments is 49. The total number of pages used is 72. All our experimental Web pages are collected randomly. Due to the long URLs of most pages, we could not list them here. We will post all the URLs of the test pages on our Web site.

Columns 2 and 4: They give the numbers of correct (Cor.) records extracted by MDR and MDR-2 (step 1 of DEPTA) from the pages of each site respectively. These data records are those obvious ones of the pages (e.g., product lists). They do not include navigation areas, which may also have regular patterns.

Note that although the MDR-2 framework is able to handle nested data records (records in records) due to its nested similarity comparison, we did not explicitly handle such data records in this work as they are relatively rare in record lists. We will add this in the future. The proposed partial tree alignment method is able to align data items in nested records.

Columns 3 and 5: They give the numbers of data records extracted wrongly (Wr.) by MDR and MDR-2 (step 1 of DEPTA) from the pages of each site respectively. *x/y* means that *x* is the number of extracted results that are incorrect, and *y* is the number of results that are not extracted.

Table 2: Experimental results

URL	MDR		DEPTA			
			MDR-2: Step 1		Step 2	
	Cor.	Wr.	Cor.	Wr.	Cor.	Wr.
accessories.gateway.co	6	0/0	6	0/0	12	0/0
advanced.search.shopp	15	0/0	15	0/0	195	0/0
crafts.listings.ebay.co	50	0/0	50	0/0	350	0/0
froogle.google.com/...	0	0/10	10	0/0	79	1/0
google1-cnet.com/...	32	0/6	32	0/6	224	0/42
google-zdnet.com/...	23	0/0	23	0/0	207	0/0
list.auctions.shopping.	25	0/0	25	0/0	225	0/0
photography.listings.e	27	0/23	50	0/0	300	0/0
reviews.cnet.com/...	7	0/0	7	0/0	42	0/0
search.ebay.com/...	100	0/0	100	0/0	576	0/0
sensualexpression.com	6	0/0	6	0/0	12	0/0
shopping.yahoo.com/...	10	0/0	10	0/0	30	0/0
store.babycenter.com/..	10	0/0	9	0/1	43	0/4
store.yahoo.com/...	7	0/0	7	0/0	35	0/0
video.shopping.yahoo.	15	0/0	15	0/0	150	0/0
www.abtelectronics.co	38	0/4	39	0/3	134	0/12
www.acehardware.co	9	0/0	9	0/0	54	0/0
www.adesso.us/...	10	7/8	16	0/2	62	0/8
www.alibris.com/...	14	1/4	18	1/0	163	15/0
www.amazon.com/...	4	2/9	12	0/1	132	0/11
www.ashford.com/...	7	0/12	19	0/0	57	0/0
www.bargainoutfitters.	22	0/0	22	0/0	132	0/0
www.bestbuy.com/...	18	0/3	21	0/0	273	0/0
www.bobsdiscountmar	16	0/0	16	0/0	64	0/0
www.buy.com/...	10	0/5	15	0/0	148	2/0
www.cameraworld.co	28	1/4	31	0/1	131	1/7
kwww.circuitmicro.co	15	0/0	15	0/0	63	0/0
www.compusa.com/...	8	0/0	8	0/0	80	0/0
www.cooking.com/...	22	15/15	35	0/2	173	0/28
www.dealtime.com/...	5	0/15	19	0/1	114	0/6
www.drugstore.com/...	11	3/3	11	0/3	67	0/16
www.essentialapparel.	8	0/0	8	0/0	32	0/0
www.magazinesofame	6	0/0	6	0/0	42	0/0
www.nextag.com/...	15	0/0	15	0/0	178	0/0
www.nothingbutsoftw	65	1/0	65	1/0	260	4/0
www.npg.org/states/...	104	0/0	104	0/0	416	0/0
www.officedepot.com/	19	0/0	19	0/0	228	0/0
www.overstock.com/...	39	0/0	39	0/0	264	0/0
www.pricegrabber.co	62	0/0	62	0/0	459	0/0
www.radioshack.com/.	3	0/0	3	0/0	24	0/0
www.randomhouse.co	25	0/0	25	0/0	258	0/0
www.refurbdepot.com/	12	0/6	18	0/0	158	0/0
www.rochesterclothing	20	0/0	20	0/0	40	0/0
www.shoebuy.com/...	10	0/0	10	0/0	60	0/0
www.shopping.com/...	25	0/0	25	0/0	205	0/0
www.sitistore.com/...	0	0/20	20	0/0	206	0/0
www.smartbargains.co	8	0/0	8	0/0	16	0/0
www.target.com/...	0	0/8	8	0/0	40	0/0
www.tigerdirect.com/..	14	0/0	14	0/0	43	0/0
Total	1005	30/155	1140	2/20	7256	23/134
Recall	**86.64%**		**98.27%**		**98.18%**	
Precision	**97.10%**		**99.82%**		**99.68%**	

Columns 6 and 7: They give the numbers of correct data items and wrong data items extracted by step 2 of DEPTA from the data records of each site respectively. An extracted data item with a wrong alignment is also considered an error.

The last three rows of Table 2 give the total of each column, the recall and precision of each system. For MDR and MDR-2 (step 1 of DEPTA), 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 extraction of DEPTA, the precision and recall computation has considered the 20 lost data records because of step 1 of DEPTA. We can see that the data extraction is highly effective. Almost all the errors are due to data record extraction. We also observe that MDR-2 performs significantly better than MDR.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a new approach to extract structured data from Web pages. Although the problem has been studied by several researchers, existing techniques are either inaccurate or make many strong assumptions. Our method does not make these assumptions. It only requires that the page contains more than one data record, which is almost always true for pages with data records. Our technique consists of two steps: (1) identifying data records without extracting each data field in the data records, and (2) aligning corresponding data fields from multiple data records to extract data from them to put in a database table. We proposed an enhanced method based on visual information for step (1), which significantly improves the accuracy of our previous algorithm. 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 show that the new two-step technique can segment data records and extract data from them very accurately.

7. ACKNOWLEDGMENTS

This work was supported by NSF (IIS-0307239).

8. REFERENCES

[1]. Arasu, A. and Garcia-Molina, H. Extracting Structured Data from Web Pages. *SIGMOD-03*, 2003.

[2]. Baeza-Yates, R. Algorithms for string matching: A survey. *ACM SIGIR Forum*, 23(3-4):34-58, 1989.

[3]. Barton, G., Sternberg, M. A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons. *J. Mol. Biol.* 1987, 327-337.

[4]. Bar-Yossef, Z. and Rajagopalan, S. *Template Detection via Data Mining and its Applications*, *WWW 2002*, 2002.

[5]. Buttler, D., Liu, L., Pu, C. A fully automated extraction system for the World Wide Web. *IEEE ICDCS-21*, 2001.

[6]. Carrillo, H., Lipman, D. The multiple sequence alignment problem in biology. *SIAM J. Applied Math.*, 1988;48(5).

[7]. Chakrabarti, S. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann Publishers, 2002.

[8]. Chang, C. and Lui, S-L. IEPAD: Information extraction based on pattern discovery. *WWW-10*, 2001.

[9]. Chen, H.-H., Tsai, S.-C., and Tsai, J.-H. Mining tables from large scale html texts. *COLING-00*, 2000.

[10]. Chen, W. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40:135–158, 2001.

[11]. Cohen, W., Hurst, M., and Jensen, L. A flexible learning system for wrapping tables and lists in HTML documents. *WWW-2002*, 2002.

[12]. Crescenzi, V., Mecca, G. and Merialdo, P. Roadrunner: Towards automatic data extraction from large web sites. *VLDB-01*, 2001.

[13]. Doorenbos, R., Etzioni, O., Weld, D. A scalable comparison shopping agent for the World Wide Web. *Agents-97*, 1997.

[14]. Embley, D., Jiang, Y and Ng, Y. "Record-boundary discovery in Web documents." *SIGMOD-99*, 1999.

[15]. Gusfield, D. *Algorithms on strings, tree, and sequence*, Cambridge. 1997.

[16]. Hammer, J., Garcia-Molina, H., Cho, J., Aranha, R., and Crespo, A. Extracting semi-structured information from the Web. *Workshop on Manag. of Semi-structured Data*, 1997.

[17]. Hogeweg, P., Hesper, B. The alignment of sets of sequences and the construction of phylogenetic trees: An integrated method. *J. Mol. Evol*., 20, 175-186 (1984).

[18]. Hsu, C.-N. and Dung, M.-T. Generating finite-state transducers for semi-structured data extraction from the Web. *Information Systems*. 23(8): 521-538, 1998.

[19]. Kushmerick, N. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence*, 118:15-68, 2000.

[20]. Lerman, K., Getoor L., Minton, S. and Knoblock, C. "Using the Structure of Web Sites for Automatic Segmentation of Tables." *SIGMOD-04*, 2004.

[21]. Liu, B., Grossman, R. and Zhai, Y. "Mining data records from Web pages." *KDD-03*, 2003.

[22]. Meng, X., Lu, H., Wang, H., and Gu, M. Schema-Guided Wrapper Generator. *ICDE-02*, 2002.

[23]. Muslea, I., Minton, S. and Knoblock, C. "A hierarchical approach to wrapper induction." *Agents-99*, 1999.

[24]. Notredame, C. Recent progresses in multiple sequence alignment: a survey. *Technical Report*. 2002.

[25]. Pinto, D., McCallum, A., Wei, X. and Bruce, W. Table Extraction Using Conditional Random Fields. *SIGIR-03*.

[26]. Ramaswamy, L., Ivengar, A., Liu, L., and Douglis, F. Automatic detection of fragments in dynamically generated Web pages. *WWW-04*, 2004.

[27]. Reis, D. Golgher, P., Silva, A., Laender, A. Automatic Web news extraction using tree edit distance, *WWW-04*, 2004.

[28]. Rosenfeld, B., Feldman, R., Aumann, Y. Structural extraction from visual layout of documents. *CIKM-02*, 2002.

[29]. Song, R., Liu, H., Wen, J.-R., Ma, W.-Y. Learning block importance models for Web pages. *WWW-04*, 2004.

[30]. Tai, K. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979

[31]. Valiente, G. Tree edit distance and common subtrees. Research Report LSI-02-20-R, Universitat Politecnica de Catalunya, Barcelona, Spain, 2002.

[32]. Wang, J., Shapiro, J., Shasha, D., Zhang, K., Currey, K. An algorithm for finding the largest approximately common substructures of two trees. *IEEE PAMI*, 20(8), 1998.

[33]. Wang, Y., Hu**,** J. A machine learning based approach for table detection on the Web. *WWW-2002*.

[34]. Wang, J.-Y., and Lochovsky, F. Data extraction and label assignment for Web databases. *WWW-03*, 2003.

[35]. Yang, W. Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, 21(7):739–755, 1991.

[36]. Zhang, K., Statman, R., Shasha, D. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.