Extracting Web Data Based On Partial Tree Alignment Using Fivatech

I. INTRODUCTION

The World Wide Web is a vast and rapidly growing source of information. Most of this information is in the form of unstructured text, making the information hard to query. There are, however, many websites that have large collections of pages containing structured data, i.e., data having a structure or a schema.

These pages are typically generated dynamically from an underlying structured source like a relational database. Extracting structured data from the web pages is clearly very useful, since it enables us to pose complex queries over the data. Extracting structured data has also been recognized as an important sub-problem in information integration systems, which integrate the data present in different web-sites. Therefore, there has been a lot of recent research in the database and AI communities on the problem of extracting data from web pages (sometimes called information extraction (IE) problem).

In other words, these pages are generated with a predefined template by plugging data values. In practice, template pages can also occur in surface Web (with static hyperlinks). In addition, templates can also be used to render a list of records to show objects of the same kind. Thus, information extraction from template pages can be applied in many situations. In this, we focus on page-level extraction tasks and propose a new approach, called FiVaTech, to automatically detect the schema of a Website. The proposed technique presents a new structure, called fixed/variant pattern tree, a tree that carries all of the required information needed to identify the template and detect the data schema.

We combine several techniques: alignment, pattern mining, as well as the idea of tree templates to solve the much difficult problem of page-level template construction.

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 [7] for this purpose. Specifically, the new method also uses visual cues to find data records.

2. A novel partial tree alignment method is proposed to align and to extract 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 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. 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 table.

Keywords— Data Record Extraction, Partial Tree Alignment, Wrapper, Web Data Extraction

Abstract— In this paper studies the problem of extracting structured data from Web pages. The objective of the proposed research is to automatically extract data items/fields from records, and store the extracted data in a database. We formally define a template, and propose a model that describes how values are encoded into pages using a template. For this purpose we perform the techniques of alignment, pattern mining, as well as the idea of tree templates to solve the much difficult problem of page-level template construction. Based on above two steps an unsupervised, page level data extraction approach is used to deduce schema and Template for each individual Deep Web site.
2. PROBLEM FORMULATION:

In this section, we formulate the model for page creation, which describes how data are embedded using a template. As we know, a Webpage is created by embedding a data instance $x$ (taken from a database) into a predefined template.

**Definition 2.1 (Structured data).** A data schema can be of the following types:

- A tuple, denoted by <$k$>, if the number of instances is $k$.
- A set, denoted by $\{k\}$, if the cardinality is greater than $k$ for some instantiation.
- A disjunction, denoted by $\left(\bigcup_{i=1}^{k} T_i \right)$, if all $T_i (i=1,...,k)$ are options and the cardinality sum of the $k$ options $\bigcup_{i=1}^{k} T_i$ equals $1$ for every instantiation of $T$.

**Definition 2.2.** Let $T_1$ and $T_2$ be two trees, we define the operation $T_1 \& T_2$ as a new tree by appending tree $T_2$ to the rightmost path of $T_1$ at the $i$th node (position) from the leaf node.

For example, given the templates trees $C;E$ and data contents $P;S$ (for content data “Product 1” and “Save 5 percent,” respectively). We also show two sub trees $N$ (for content data “Now $3:79$”) and $E \& S$ inserted as sibling under template $D$ at insertion point 0. We denote this operation by $D \& (N,E \& S)$.

**Definition 2.3 (Level-aware encoding).** We define the template for a type constructor $T$ as well as the encoding of its instance $x$ (in terms of encoding of subvalues of $x$) as described below.

1. If $T$ is of a basic type, $\beta$, then the encoding $\lambda(T,x)$ is defined to be a node containing the string $x$ itself.
2. If $T$ is a type constructor of order $k$, then the template is denoted by: $T(\Theta, \{C_1,\ldots,C_k\},(i_1,\ldots,i_k))$ where $P,C_1,\ldots,$ and $C_{k+1}$ are template trees.
   a. For single instance $x$ of the form $(x_1,\ldots,x_k), \lambda(T,x)$ is a tree produced by concatenating the $k+1$ ordered subtrees, $C_1 \Theta_1 \lambda(T,x_1), C_2 \Theta_2 \lambda(T,x_2)\ldots C_k \Theta_k \lambda(T,x_k)$ and $C_{k+1}$ at the leaf on the rightmost path of template $P$.
   b. For multiple instances $e_1, e_2,\ldots,e_m$ where each $e_i$ is an instance of type $T$, the encoding $\lambda(T, (e_1, e_2,\ldots,e_m))$ is the tree by inserting the $m$ subtrees $\lambda(T, e_1), \lambda(T, e_2)\ldots \lambda(T, e_m)$ as siblings at the leaf node on the rightmost path of the parent template $P$. Each subtree $\lambda(T, e_i)$ is produced by encoding $e_i$ with template $[\Theta, \{C_1,\ldots,C_{k+1}\}, (i_1,\ldots,i_k)]$ using the procedure for single instance as above $\Theta$ is the null template (or a virtual node).
   c. For disjunction, no template is required since the encoding of an instance $x$ will use the template of some $T_i (1 \leq i \leq k)$, where $x$ is an instance of $T_i$.

**Definition 2.4 (Wrapper induction).** Given a set of $n$ DOM trees, $D_{OM} = \lambda(T,x_i) (1 \leq i \leq n)$, created from some unknown template $T$ and values $x_1,\ldots,x_n$, deduce the template, schema, and values from the set of DOM trees alone. We call this problem a page-level information extraction.

If only one single page ($n = 1$) that contains tuple constructors is given, the problem is to deduce the template for the schema inside the tuple constructors. We call this problem a record-level information extraction task.

3. FIVATECH TREE MERGING:
The proposed approach FiVaTech contains two modules: tree merging and schema detection (see Fig. 4). The first module merges all input DOM trees at the same time into a structure called fixed/variant pattern tree, which can then be used to detect the template and the schema of the Website in the second module. In this section, we will introduce how input DOM trees can be recognized and merged into the pattern tree for schema detection.

There are at least two advantages in this design. First, as the number of child nodes under a parent node is much smaller than the number of nodes in the whole DOM tree or the number of HTML tags in a Webpage, thus, the effort for multiple string alignment here is less than that of two complete page alignments in RoadRunner [8]. Second, nodes with the same tag name (but with different functions) can be better differentiated by the sub trees they represent, which is an important feature not used in EXALG [2].

Then, we conduct the four steps: peer node recognition, matrix alignment, pattern mining, and optional node detection in turn.

In the peer node recognition step, two nodes with the same tag name are compared to check if they are peer subtrees. All peer subtrees will be denoted by the same symbol.

The alignment is thus partial. The algorithm tries to find for each node in $T_i$ a matching node in $T_s$, which can be uniquely decided: 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.

### 4.1. 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 $n_i$ 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 $n_j...n_m$ from $T_i$ together. Without loss of generality, we assume that the parent node of $n_j...n_m$ has a match in $T_s$ and we want to insert $n_j...n_m$ into $T_s$ under the same parent node.

We only insert $n_j...n_m$ into $T_s$ if a position for inserting $n_j...n_m$ 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 $n_j...n_m$ can be uniquely decided:

1. if $n_j...n_m$ have two neighboring siblings in $T_s$, one on the right and one on the left, that are matched with two consecutive siblings in $T_i$, Figure 3(A) shows such a situation, which gives one part of $T_i$ and one part of $T_s$.

We can see that node $c$ and node $d$ (which are consecutive sibling nodes) in $T_s$ can be inserted into $T_i$ between node $b$ and node $e$ in $T_i$ because node $b$ and node $e$ in $T_i$ and $T_s$ match. The new (extended) $T_i$ is also shown in Figure 3(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 \( y \) in \( T_s \), then \( n_j \ldots n_m \) can be inserted after node \( x \) in \( T_s \). Figure 3(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 3(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 \( T_s \).

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: Final data table (“1” indicates a data item)

<table>
<thead>
<tr>
<th></th>
<th>...</th>
<th>x</th>
<th>b</th>
<th>n</th>
<th>c</th>
<th>d</th>
<th>h</th>
<th>k</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>...</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Expanding the seed tree: (A) and (B) Unique expansion; (C) Insertion ambiguity.

Table 1 shows the data table for the trees in Figure 4. We use “1” to indicate a data item. In Figure 4, \( T_1 \) 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 attached or inserted. The process completes. Note that if there are still un-matched nodes with data, each un-matched data will occupy a single column by itself. Table 1 shows the data table for the trees in Figure 4. We use “1” to indicate a data item.

5. CONCLUSION:

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. 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.

In this paper, we proposed a new Web data extraction approach, called FiVaTech to the problem of page-level data extraction. In this paper we are using a partial tree alignment as a DOM tree in FiVatech framework. The advantage is that nodes with the same tag name can be better differentiated by the sub tree they contain.

6. REFERENCES:


