RECORD EXTRACTION USING RECORD SEGMENTATION TREE

A Suresh Babu¹, P. Premchand² and A. Govardhan³

¹Department of Computer Science and Engineering, JNTUACE Pulivendula, India

asureshjntu@gmail.com

²Professor, Department of Computer Science Engineering, Osmania University, Hyderabad, India

p.prechand@uceou.edu

³Professor, Department of Computer Science Engineering, JNTUH, Kukatpalli, Hyderabad, India.

govardhan_cse@yahoo.co.in

Abstract

In spite of extensive study of information extraction from web pages, the existing methods fail to extract all the data from the web pages. Also, the existing methods divide the data extraction into two phases, namely, record region detection and record segmentation. In this paper, we proposed a unified method for data extraction from a structured web page. We propose a new search structure Record Segmentation Tree (RST), and few search pruning techniques on RST to make the extraction faster and efficient. This method can handle more complicated web pages as we have used token based edit distance instead of string or tree edit distances. And, the partial tree alignment method is used to align the extracted data into a more understandable form. Experiments have been conducted on data sets used in different existing methods and our method gives more efficient result than those existing methods.

1. Introduction

In the earliest days of the Web, there were relatively few documents and sites. It was a manageable task to “post” all documents as “static” pages; so they could be easily crawled by conventional search engines. But today, World Wide Web is a large source of information and being developed extensively. A great amount of effort is often required for a user to manually locate and extract useful data from the Web sites. Researchers have built tools that can generate wrappers automatically under wrapper induction methods [1][2] such as MDR [3], DeLa [4][5], Viper [6], DEPTA [7] and IEPAD [8] were designed to tackle the task of record level extraction from a single web page. The methods that address the record extraction task can be categorized into 5 types. The repetitive pattern based methods, IEPAD [8] and DeLa, mine some repetitive patterns as clues for locating records in the page as similar templates are used in formatting the records. But these methods fail at handling optional data and tags inserted into records. The similarity based methods, MDR and DEPTA handle this problem utilizing string and tree edit distance to assess whether two adjacent subtree groups are a repetition of the same data type. Another work, ViPER in which resemblance of each pair of single subtrees is calculated to detect record region, then involves some visual perception to segment the detected regions into records. In contrast, ViNTs [9] utilizes the visual information first to identify content regularity, and then to generate wrappers, combines it with tag structure regularity. ViNTs cannot separate horizontally arranged records, e.g., nested records in a table, and identify multiple regions. Pure visual feature based methods include VENTex [10] and ViDE [11], and they are effective to extract records from pages with well organized visual features. The limitation of all the above methods is that they require two steps record region detection and record segmentation. The proposed frame work unifies the above two steps into one using the newly proposed search structure, called as Record Segmentation Tree (RST) and uses partial tree alignment for alignment of extracted data.
2. Proposed system

There are two basic observations made by considering the characteristics of data records by the previous works. They are:

1. A group of data records describing a set of similar objects are usually presented in a particular region of a page and are formatted using similar HTML tags.

2. A group of similar data records being placed in a specific region is reflected in the tag tree by the fact that they are under one parent node, although we do not know which parent. It is very unlikely that a data record starts from an inner node of a child subtree and ends at an inner node of another child subtree of the parent node.

Based on these two observations, the information extraction task has been divided into 2 steps namely, record region detection and record segmentation by the previous works. But our work unifies the two tasks by performing these two tasks simultaneously. The company information is organized in a “table” is shown in the Web page.
fragment given in Figure 1(a), each data record corresponds to a company. The DOM tree of the table is given in Figure 1(b). We can see that the records share some common fields such as company description, investor, and established year. And some records do not contain certain fields. For instance, the URL information is not given in the third record. Furthermore, similar HTML templates are used to format the records, and several rows form a record in the table. We use T to denote the DOM tree given in Figure 1(b), and T’s subtree sequence is denoted by S, and an Sj is referred to as an element in S. In the DOM tree in Figure1(b), T is “table”, and S includes S1, S2, etc. T and Si are also used to refer to the root nodes of the corresponding DOM trees. S1, S2, etc. Sj denotes a fragment of the sequence S, where 1 ≤ i ≤ j ≤ |S|. The aim of data record extraction is to identify the sequence Si..j and Set of separating indexes b in which each bh∈b s.t. i < bi ≤ j. The indexes in b are used to separate the data records in the identified sequence S1..j, a record region.

For the current example, the record region is S2..|S|, and the boundary of records in this region are indicated by separating index set {4, 7, 9...}. Thus, the records in the above example are S2..4, S5..7, S8..9. It is not necessary for a record region to start from the first subtree of T, and the length (number of subtrees) of different records need not be the same. In addition, some DOM trees may contain 0 or more than one regions. If there is no record region, no subtree sequence should be identified. If there are several regions, several subsequences of S should be identified. Once the records are extracted using the proposed system they are aligned into data tables using the partial tree alignment method.

3. Record Segmentation Tree

Record Segmentation Tree (RST), is exploited to detect possible records for the given subtree sequence S on which the searching and identifying the data records is carried out. If some data records are identified, they naturally compose record regions. Thus, performing region detection and record segmentation can be done simultaneously. It has the following properties:

1. The root node represents an empty region and each other node represents a possible record region.
2. Each R covers a prefix sequence S1..a of S, referred to as Sr, where 0 ≤ n ≤ |S|, and has a separating indexes set b which segments S1..a into records. Each record of R is denoted by Rr. The root has an empty prefix subsequence, i.e. n = 0, and an empty separating indexes set.
3. Each R with S1..a and b has at most K children.

Each child of R covers S1..m where n + 1 ≤ m ≤|S| + K, |S|, and has a separating indexes set b ∪ {m}. Where K is the at most subtrees in S.

In this example 2, record set is used to label each node in the segmentation tree. A node containing two records, namely, R1 = S1..3, and R2 = S3..6 is denoted by R = f S1..3, S3..6/f, S1..6, is the covered prefix and f, 6/ is the separating indexes set. |R| is equal to the number of subtrees in R. For instance, |R1| = 3, |R2| = 3. We use |R| to denote the number of records in R.

The average length of the records in R is denoted by LR and is calculated using the formula (∑|R|/|R|) or |SR|/|R|.

From the observation that the records in the same region are formatted using similar tags, the node that achieves higher average pair wise record similarity would be the correct segmentation. If we cannot find a node with pair wise record similarity greater than a pre-defined threshold, we may conclude that no record region exists starting from S1. Precisely, given a DOM tree T and its subtree sequence S, record extraction with the RST structure of S aims at finding a node R* such that:

\[ R^* = \text{argmax} \frac{|R|}{|Q(R)| \theta} \]

Where \( \theta \) is a pre-defined threshold. Q(·) is the quality function of an RST node R, which is defined as the average pair wise record similarity of records in R:

\[ Q(R) = \sum_{R_i \in R \setminus \{R_f\}} \frac{sim(R_i, R_f)}{|R_i| \cdot |R_f| - 1} / 2 \]

Where sim is a similarity function between two records.

3.1 Pruning Strategies

prune the RST significantly. Can be done by threshold-based top k search which can reduce the complexity to O(|S|2) without considering pair wise similarity computation of subtrees. Furthermore, instead of calculating the similarity of all record pairs in Equation 2, we may only check the similarity of a record and its nearest previous neighbours. The following Prune Strategies shows threshold hold top k search.

3.1.1 Initialization:

The RST has to be expanded to first 2 layers to attain a better starting as shown in Figure 2. If there are K numbers of children to each of the non leaf node, K is the number of initial candidate nodes denoted by R, for future expansion and search. As K value is
very small full expansion of tree in layer 2 is accepted. Using the Equation 2, the quality for each of the candidate nodes in “R” is calculated and they are pruned before proceeding to the next layer of RST. For a particular node R in R’, it will be pruned if \( Q(R) < \theta' \) where \( \theta' \) is a value less than that of \( \theta \). The top \( k \) nodes with best quality will be retained. For example in Figure 2, the node \( S_i, S_{j,f} \) is very likely to be pruned. If R= \( \theta \), then no region starting from \( S_i \) exists in R.

### 3.1.2 Pruning Search:

Let R be a node in R and the last record in R is \( S_{m,n} \). Some children of R, \( RU(S_{m1,m}) \), \( RU(S_{m1,m2}) \ldots \ldots ,RU(S_{m1,ms}) \) are constructed. The similarity between newly generated records and the existing ones in R are calculated. Precisely, for each new record \( S_{n1,m} \) \((n+1 \leq m \leq n+K)\) and each \( R\epsilon R, \text{sim}(S_{n1,m},R) \) is calculated. Then, the quality of \( S_{n1,m} \) is defined as:

\[
Q(S_{n1,m}) = \sum \text{ReR sim}(S_{n1,m},R) / |R| \quad (3)
\]

Each R in R is expanded, and the quality of each new record is calculated in the way shown above. After the RST structure is expanded one more layer, a new pruning strategy different from that for “R” is adopted. For a new RST node \( RU(S_{m1,m}) \) is retained, if \( Q(S_{m1,m}) \geq \theta \) and \( Q(RU(S_{m1,m})) \geq \theta \), pruned otherwise. The reason is that some record, say \( S_{m1,m} \), may differ with its previous records largely. But our assumption is that the difference is not large, thus reducing the number of searching nodes. In this way, the search procedure can overcome the problems caused by some outlier records. Meanwhile, the quality of the outliers is also bounded by \( \theta' \). The rationale behind using \( \theta' \) instead of \( \theta \) in the pruning of the initial candidate nodes R’ is that each element of R’ contains two records. For a particular R, if no \( RU(S_{m1,m}) \) is retained after pruning, R is put into R*. If the number of retained nodes is more than \( k \) again, we keep the top \( k \) nodes with the best quality, and construct a new R. This procedure is applied on the new R until it is empty or the end of S is reached. R* is also obtained by the time building an RST structure for S starting from \( S_1 \) is finished. Note that the RST structure needs not reach \( S_{K,0} \), which indicates that some subtrees in the end of S should not be included in the record region.

### 3.1.3 Retrospect with Short-term Memory:

All the previous records are considered while examining the quality of a new record, as shown in Equation 3 for expanding and pruning. But this thorough retrospect is unnecessary and further it is time consuming. To overcome this problem and reduce the computation workload, we adopt a short-term memory retrospect strategy. When a node is expanded one more layer, we only calculate the similarity between a new record and the nearest \( r \) previous records. Thus, the number of pair wise similarity calculations is reduced from \(|R|\) to \( r \) in Equation 3. And the cost of evaluating one node is reduced from \(|R|(|R| - 1)/2 \) to \( r|R| \), in Equation 2.

### 3.2 Record Similarity Measure

#### Token Based Edit Distance:

The HTML cleaning package by name HtmlCleaner is employed in fixing the ill-formatted Web pages to avoid time consuming rendering the web pages into DOM trees. In the DOM tree, we have two kinds of nodes, namely, tag node and text node. Each tag node has a name such as “tr”, “div”, etc. A piece of visible text between a pair of “<” and “>” is regarded as a basic text unit, and normalized to a text node. Types of text nodes defined in this framework is shown in the Table 1. The priority indicates the order in which different types are attempted when normalizing a piece of text. Each subtree in the DOM is encoded with the sequence of node names in it, which is obtained by traversing the tree in pre-order. Each node name in the sequence is called one token, and it is an inseparable unit in similarity calculation.
Figure 2 An example of basic record segmentation tree with $K = 4$.

Figure 3 Expanding the seed tree: (A) and (B) Unique expansion; (C) Insertion ambiguity
Table 1. Types of text nodes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Meaning</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMAIL</td>
<td>An email address</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>URL</td>
<td>A URL string</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>PRICE</td>
<td>Digital number with a</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>TIME</td>
<td>Time in predefined</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>DATE</td>
<td>Date in predefined</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>TEXT</td>
<td>Other kinds of text</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>YEAR</td>
<td>Four digits, arrange</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>from 1900 to 2050</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tandem Repeat Detection and Distance based Measure:
Token-based edit distance [12] scenario utilizes suffix tree structure to identify the tandem repeats which are not longer than z in a sequence of length n in O(n+z) time. In detection of tandem repeats instead of character comparisons token comparisons are used as the basic unit used in this approach as a token but not a character. Furthermore, if the token sequence is originated from several subtrees, it is constrained that tandem repeats are only detected in the token sequence of each single subtree.

4. Alignment of data using partial trees
This section emphasizes the concept of partial tree alignment of two trees. Some nodes in TI can be aligned with their corresponding nodes of Tj once Ti and Tj are matched, as they match one another. For those nodes in Tj that are not matched, they should be inserted into Ti, as they may contain optional data items. There are two possible situations when inserting a new node nj from Tj into the seed tree T0, depending on whether a location in T0 can be uniquely determined to insert nj. A set of unmatched consecutive sibling nodes nj…nm from Tj together can be considered instead of considering a single node nj. Without loss of generality, we assume that the parent node of nj…nm does not have a match in T0 and we want to insert nj…nm into T0 under the same parent node. The sequence of nodes nj…nm can be inserted into T0 if a position for inserting nj…nm can be uniquely determined in T0. Otherwise, they will be left unaligned and will not be inserted into T0. Thus, the alignment is partial. The location for insertion of nj…nm can be uniquely decided:
1. If nj…nm have two neighboring siblings in T0, one on the right and one on the left, that are matched with two consecutive siblings in T0. Figure 3(A) shows such a situation, which gives one part of T0 and one part of Tj. We can see that node c and node d (which are consecutive sibling nodes) in T0 can be inserted into Tj between node b and node e in Tj, because node b and node e in T0 and Tj match. The new (extended) Tj 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 nj…nm has only one left neighbouring sibling x in Tj and x matches the right most node x in Tj, then nj…nm can be inserted after node x in Tj. Figure 3 (B) illustrates this case.
   3. if nj…nm has only one right neighbouring sibling x in Tj and it matches the left most node x in Tj, then nj…nm can be inserted before node x in Tj. This case is similar to above.
   Otherwise, we cannot uniquely decide a location for unmatched nodes in Tj to be inserted into T0. This is illustrated in Figure 3 (C). In this case, the unmatched node x in Tj could be inserted into T0 in two positions, between nodes a and b, or between node b and e in Tj. In this situation, we will not insert it into T0.

Full algorithm
Following is the full algorithm for multiple tree alignment based on Partial tree alignment of two tag trees.

Algorithm: Partial Tree Alignment(S)
1. Sort trees in S in descending order according to the number of data items that are not aligned;
2. T0 = the first tree (which is the largest) and delete it from S;
3. flag = false; R = ∅; I = false;
4. while (S ≠ ∅)
5. Tj = select and delete next tree from S;
6. Simple_Tree_Matching (T0, Tj);
7. L = align Trees (Ts, Tj); // based on the result from line 6
8. if Ti is not completely aligned with Ts then
9. I = InsertIntoSeed (Ts, Ti);
10. if not all unaligned items in Ti are inserted into Ts then
11. Insert Ti 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 = ∅ and flag = true then
18. S = R; R = ∅;
19. flag = false; I = false
20. endif;
21. end while;
22. Output data fields from each Ti to the data table based on the alignment results.

5. Experimental Results
For evaluating the performance of the proposed method and conducting comparison with existing method MDR, existing data sets and available implementation of existing method is used. Web pages from different websites of different domain are used for testing the performance of current framework on flat data record extraction. As no existing data set for nested or intertwine records is available, only limited numbers of web pages are collected, and the results are compared with MDR only. With respect to evaluation metrics, the commonly used precision and recall values are employed.
For a given web page, the algorithm adopts a top down manner to scan its DOM tree for detecting records. Several record regions may be detected for some pages. Pages contains more than one record regions, it only reports the largest one, and lacks others. The regions whose record size is smaller than 3 are filtered out, where the record size is defined as that number of leaf DOM nodes it has. Finally threshold value \( \theta = 0.75 \), \( \theta' \) is to be 0.65, \( K \) is set to 10, both \( k \) and \( r \) are set to 5.

Experiments on the proposed system conducted are on the web pages from different web sites are shown in Table 5.3. The measure of proposed method is based on three factors, the number of actual data records to be extracted, the number of extracted data records from the test pages, and the number of correct data records.

### Table 2: Experimental results of Record Segmentation Tree

<table>
<thead>
<tr>
<th>URL</th>
<th>Actual</th>
<th>Correct</th>
<th>Extract</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RST</td>
<td>MDR</td>
<td>RST</td>
<td>MDR</td>
<td>RST</td>
</tr>
<tr>
<td>crafts.listings.ebay.co/</td>
<td>400</td>
<td>400</td>
<td>388</td>
<td>385</td>
<td>388</td>
</tr>
<tr>
<td>ebay.com</td>
<td>550</td>
<td>550</td>
<td>536</td>
<td>525</td>
<td>420</td>
</tr>
<tr>
<td>reviews.cnet.com</td>
<td>520</td>
<td>520</td>
<td>476</td>
<td>470</td>
<td>460</td>
</tr>
<tr>
<td>sensualexpression.co</td>
<td>420</td>
<td>420</td>
<td>386</td>
<td>380</td>
<td>382</td>
</tr>
<tr>
<td>shopping.yahoo.com</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>240</td>
<td>250</td>
</tr>
<tr>
<td><a href="http://www.albris.com/">www.albris.com/</a></td>
<td>370</td>
<td>370</td>
<td>340</td>
<td>330</td>
<td>320</td>
</tr>
<tr>
<td><a href="http://www.ashford.com/">www.ashford.com/</a></td>
<td>482</td>
<td>482</td>
<td>440</td>
<td>450</td>
<td>440</td>
</tr>
<tr>
<td>bargainoutfitters.co</td>
<td>650</td>
<td>650</td>
<td>620</td>
<td>600</td>
<td>610</td>
</tr>
<tr>
<td><a href="http://www.bestbuy.com/">www.bestbuy.com/</a></td>
<td>310</td>
<td>310</td>
<td>300</td>
<td>280</td>
<td>300</td>
</tr>
<tr>
<td>cameraworld.co</td>
<td>450</td>
<td>430</td>
<td>425</td>
<td>415</td>
<td>424</td>
</tr>
<tr>
<td><a href="http://www.circuitmicro.co">www.circuitmicro.co</a></td>
<td>220</td>
<td>220</td>
<td>195</td>
<td>190</td>
<td>193</td>
</tr>
</tbody>
</table>
extracted from the test pages. Based on these three values, precision and recall are calculated.

The precision and recall values of RST when compared with MDR are much higher and MDR failed in extracting the complex records like nested and intertwined records. Finally the performance the RST in identifying the repeated objects is much better when compared with MDR.

6. CONCLUSION
In this paper, we present extracting data records from Web page and alignment of data records. The proposed RST structure can be utilized to address several key issues in the record extraction task. A manner for both record region detection and record segmentation, are handled simultaneously by RST structure. Different from the existing similarity-based methods, our method examines the similarity between the dynamically generated subtree groups taking into account the characteristics of the current record region. Owing to the pruning strategies, our method has a comparative complexity compared with the existing methods. Further Multiple subtree alignment is used for alignment of data records.

REFERENCES: