Tree Wrap-data Extraction Using Tree Matching Algorithm

Jer Lang Hong[1], Fariza Fauzi[2]
1- School of IT, Monash University, Malaysia
Email: david.hong @infotech.monash.edu.my
2- School of IT, Monash University, Malaysia
Email: wan.fariza@infotech.monash.edu.my

Received: December 2009 Revised: February 2010 Accepted: April 2010

ABSTRACT:
In this paper, we develop a non-visual automatic wrapper to extract data records from search engine results pages which contain important information for computer users. Our wrapper consists of a series of data filter to detect and remove irrelevant data from the web page. In the filtering stages, we incorporate two main algorithms which are able to check the similarity of data records and to detect and extract the correct data region based on their component sizes. To evaluate the performance of our algorithm, we carry out experimental and deletion tests. Experimental tests show that our wrapper outperforms the existing state of the art wrappers such as ViNT and DEPTA. Deletion studies by replacing our novel techniques with state of the art conventional techniques show that our wrapper design is efficient and could robustly extract data records from search engine results pages. With the speed advantages, our wrapper could be beneficial in processing large amount of web sites data, which could be helpful in meta search engine development.

KEYWORDS: Information Extraction, Automatic Wrapper, Search Engine, Tree Matching Algorithm.

1. INTRODUCTION

The extraction of relevant data from a target source is called Information Extraction. The target source can be a natural language source or structured records (data records) which usually contain important information. Therefore, there is a need to develop wrappers to extract these structured records. Wrappers developed recently are mostly fully automated and they could have significant speed advantages when processing large volumes of web site data, therefore they could be helpful in meta search engine development [1], [2], [3] and in comparing and evaluating shopping lists [4].

Non visual wrappers use tree matching algorithm to check the similarity of data records by comparing the position and identity of each node in the trees (data structure to represent the data records' structure in a tree form) to remove irrelevant data records with dissimilar structure [5], [20], [4]. However, the implementation and coding of the algorithm are complicated [4]. This algorithm also runs in a time complexity of O(n1n2) where n1 is the number of nodes in the first tree and n2 is the number of nodes in the second tree. In general, most web pages consist of complex trees with a large number of nodes. Therefore, these complexities slow down the current tree matching algorithm.

In this paper, we focus on developing an automated non-visual wrapper for the extraction of data records, particularly the search engine result pages. Our aim is to improve on current non-visual based wrapper performance and demonstrate that our wrapper, Tree Matching Wrapper (Tree Wrap) performs equally as well, and in many cases, better than the current state of the art automatic visual wrappers. A preliminary version of this paper has appeared in [18], [19].

We incorporate a series of data filters to remove irrelevant data records from the HTML page. These filters are designed based on heuristic techniques, each of them works based on the observations made by authors of [3-5, 7, 13, 15-17, 20-25, 23-24, 29, 31-33, 36]. The idea is to reduce the "noise" or irrelevant data records in each filtering stage so that the wrapper can be more efficient in extracting the correct data region containing data records.

We also propose a Dummy Tree Matching algorithm based on the frequency measures of a tree structure as part of the filtering stages to check the similarity of data records. This algorithm does not actually match two tree structures and find their similarity by checking the identity of each node, but uses the number of nodes in a tree to determine the similarity of two trees. As our method does not require the comparison of all the nodes in a tree structure, it will reduce significantly the computational overhead. Our algorithm works in a time complexity of O(n) (n is the number of nodes in the tree), and is faster than the

current tree matching algorithms. This increase in speed is useful when our wrapper is used in large scale web comparisons.

This paper is divided into several sections. Section 2 describes the work relevant to our research. In Section 3 we discuss our proposed methodology in detail. Section 4 discusses the result of our experimental tests while Section 5 summarizes our work. It is worth noting that data labeling is outside the scope of this paper.

2. RELATED WORK

The key component of a wrapper is the algorithm that checks the similarity of data records. Data records are retained and considered valid if they are similar and discarded if they are dissimilar. Current wrappers such as Data Extraction based on Partial Tree Alignment (DEPTA) [4] and Mining Data Region (MDR) [35] use edit distance techniques to check the similarity of the structure of data records. Common edit distance techniques in such area are string edit distance and tree edit distance [26], [8]. For more information on edit distance techniques the readers are encouraged to refer to the surveys of Baeza-Yates [26], Gusfield [8] and Navarro [14].

The string edit distance algorithm involves matching two strings and the determination of how the first string is to be transformed into the second string. String edit distance algorithms are generally fast in operation and run in a time complexity of O(m) (m is the number of tags in a data record). However, these algorithms are unable to compare two trees having nearly similar tree structures, with iterative and disjunctive data. This is because these algorithms match flat level data, which occur in single level (strings) rather than tree structures. String edit distance algorithms are also unable to distinguish HTML Tag as a single entity (they tend to compare strings by examining the characters in these strings), therefore this may result in inaccurate matching. For example, when two HTML tags <P> and <NOBR> are matched, we assume that this mismatch is counted as one (one mismatch of two HTML tags), but string edit distance algorithms consider this mismatch as 4 (4 characters in the second string do not match with the 1 character in the first string). There are several variants of string edit distance algorithms, some common ones are Levenshtein distance [30], Hamming distance [9], Episode distance [12], and Longest common subsequence distance [2], [27].

The tree edit distance algorithm uses two tree structures and matches them by comparing the node identity and position. Tree matching algorithms developed are the tree edit distance [22], alignment distance [28], isolated-sub tree distance [10], top down distance [22], [34], and bottom up distance [11]. Tree edit distance algorithm is quite similar to string edit distance, except that it includes tree nodes matching. Tree edit distance algorithm for unordered tree is *NP Complete*. The top down algorithm was proposed in [34]. For this algorithm, two trees are matched in $O(n_1n_2)$ time (n_1 is the number of nodes in the first tree and n_2 is the number of nodes in the second tree). The bottom up approach was introduced by [11] and the time complexity for it is $O(n_1+n_2)$ (n_1 is the number of nodes in the first tree and n_2 is the number of nodes in the second tree). The top down and bottom up approaches are restricted versions of tree matching algorithm.

DEPTA [4] [36] uses a bottom up tree matching algorithm to match tree structures of data records. A tree matching algorithm matches two tree structures and determines how the first tree can be transformed into the second tree. DEPTA's tree matching algorithm determines the maximum matches between two trees by comparing the location and identity of the nodes in the tree structures. Although this algorithm solves the problem emerged in data matching successfully, the algorithm requires a complex data structure for its implementation. Therefore, an algorithm that could simplify the implementation process will be helpful.

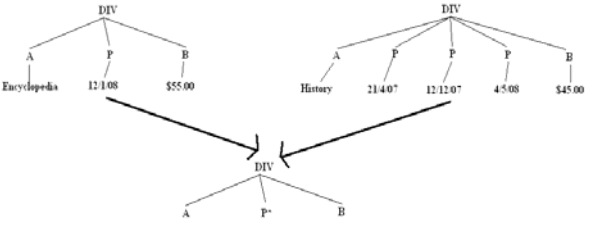

Fig. 1. Trees with different numbers of iterative data but with the same template (bottom tree)

DEPTA checks the similarity of two trees using the percentage similarity of the trees. In this context, DEPTA may not be able to match two trees having particular elements (HTML Tags) which occur iteratively in the two trees. This is due to the fact that the number of occurrence of the particular element (HTML Tags) in a tree might not be the same as it occurs in the other tree. Figure 1 shows such a case. The upper left and right trees are of different sizes, DEPTA will assume that the upper left tree is 3/((5+3)/2)=3/4=75% similar to that of the upper right tree. Basically, the two trees have a similar template (the bottom tree of Figure 1), DEPTA treats the two trees as dissimilar because they have different numbers of iterative data. A reasonable way to check the similarity of two trees is to calculate the difference in the number of nodes of the two trees. As an example, given two trees with 4 and 5 nodes each, and assume that they have 3 similar nodes; DEPTA will assume the

first tree is ¾=75% similar to the second tree. However, for large trees say with 50 and 51 nodes each, assuming they have 49 similar nodes, then DEPTA will assume the first tree is 49/50=98% similar to the right tree. We consider the trees of the two examples as nearly similar as in each case; the difference between the total number of nodes and the total number of similar nodes is only 1. DEPTA's tree matching algorithm works in a time complexity of $O(n_1n_2)$ time (n_1 is the number of nodes in the first tree and n_2 is the number of nodes in the second tree).

A wrapper is also designed to locate and extract the correct data region. Visual based wrappers such as ViNT [17], VSDR [23], and ViPER [21] use visual cue to locate and extract correct data region. These wrappers calculate the boundary and location of a data region, and take data region which is large and centrally located as the correct data region. For example, VSDR uses the Vision-based Page Segmentation Algorithm (VIPS) which segments HTML page content into several regions while ViPER uses the boundary of a HTML tag to determine data region which is centrally located.

3. THE IMPLEMENTATION OF TREEWRAP

3.1. Overview of Tree Wrap

In this section we discuss the requirements and the assumptions made for Tree Wrap. For Tree Wrap to work successfully the sample pages used for data extraction should be obtained from a search engine query and each of these sample pages must contain at least three data records. Tree Wrap however, does not require the HTML page to be converted to XHTML format as the parser can recognize the HTML format. The first component involves parsing the HTML page and organizing it into the DOM tree representation. In the second component, Tree Wrap extracts data records using dummy tree matching algorithm and scoring function. Component 1 is described in Section 3.2.1. Detailed description of Component 2 is presented in Section 3.2.2 which includes set of filtering rules.

3.2. Components of Tree Wrap

3.2.1. Component 1: Parser for Tree Wrap

A search engine result page is required as input for a parser to parse this web page. We experimented with several open source HTML to DOM tree parsers, and settled on the parser called *"HTML Parser"* (http://htmlparser. sourceforge.net/). This parser will read the sample pages and divide them into tokens. There are two types of tokens, the HTML command tag (known in short as tag) and text tokens. Tag token could be defined as any text starting with '<' and ending with '>'. Others are assumed to be text token. Once the sample web pages are parsed, Tree Wrap stored and arranged the contents in a DOM tree, which will be used for further processing in the subsequent component.

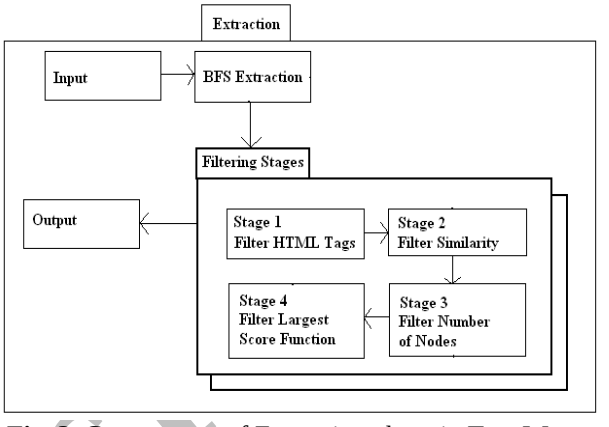

Fig. 2. Components of Extraction phase in Tree Wrap

3.2.2. Component 2: Data Extraction at Record Level in Tree Wrap

3.2.2.1. Breadth First Search (BFS) Extraction Technique

Once a web page is parsed and represented in a DOM Tree structure, our wrapper needs to traverse through the DOM Tree and identify the various data regions in the web page. To achieve this, we use Breadth First Search (BFS) technique to detect and label the different data regions. Our BFS technique developed is based on the improved and modified version used in MDR [4]. A data region can be defined as a set of data records. Data records in turn can be defined as any records that have similar parent HTML tag, contains repetitive sequence of HTML tags and are located in the same level of the DOM tree.

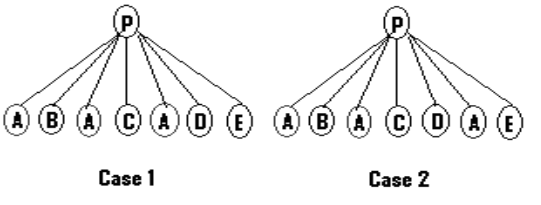

Fig. 3. Potential groups of data records (Case 1: Nodes A separated by same distance, Case 2: Nodes A separated by different distance)

The nodes in the same level are checked to determine their similarities. In the case where none of the nodes can satisfy this criterion, then the search will go one level lower and perform the search again on all the lower level nodes. Tree Wrap takes all the nodes in the same level having similar HTML tag as a potential group of data records regardless of the distance between them (Figure 3). As long as there is a repetitive sequence of HTML tags, TreeWrap treats and

labels these similar tag nodes as one group (Figure 3).Figure 3shows two cases, where the first case has three A Nodes which are separated by the same distance of 2 while the second case has two A Nodes separated by distance of 2 and a third A Node separated by a distance of 3.

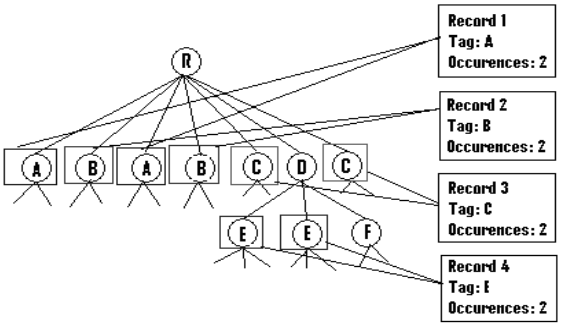

Fig. 4. Potential data records, where a node occurs more than 2 times in a level of a tree

In Tree Wrap, potential data records are treated as containing two or more nodes in one group. Figure 3 a depicts 4 data records, as shown by the rectangles. These data records appear at least twice in the same level of the tree, and have similar HTML tag identity.

3.2.2.2. Filtering Stages

3.2.2.2.1. Overview

To locate and extract the relevant data region from a pool of available data regions, Tree Wrap uses four heuristic techniques for data extraction, each of them is related to the definition of a data record. The authors of the papers [3-5, 7, 13, 15-17, 20-24, 29, 31-33, 36] on Information Extraction in Web Pages have pointed out several unique features inherent to a data record. We have also made several observations on the constitution of a data record. Based on these observations, we come out with a way to apply heuristic techniques to correctly extract a data region. The following are the observations made by several authors as presented in their papers:

Observation 1 [17, 21, 23, 31-33]:

The size of the data records is usually large in relation to the size of the whole page

Observation 2 [4-5, 21, 35]:

Data Records usually occur more than three times in a given web page.

Observation 3 [20-21, 35]:

Data Records usually conform to a specific regular expression rule to represent their individual data, hence they have nearly similar tree structure.

Our Observation 4:

Data Records usually consist of three HTML tags that make up their tree structure.

In this paper, four stages of filtering rules are proposed; each of them considers the above observations. After the completion of BFS extraction, Tree Wrap will have a list of data regions. Our examination shows that data regions fall into one of the several groups. We group the first set of potential data regions as menus; these typically determine the layout of HTML pages and are usually large in size and highly dissimilar. The second data region group is advertisements, regions of this group are highly similar but with simple structures. The third group of data regions consists of menu bars; these are simple but are nearly similar in structure. The fourth and last group in these groups of data records is relevant to our work, they are the search engine results output. This group of data records is highly similar in structure and large in size. We aim to design our wrapper so that it can extract the last group of data regions, while removing the other irrelevant ones. We used filtering stage 1 to remove advertisements, filtering stage 2 to remove menus which determine the layout of the HTML page, and finally filtering stage 4 to remove the remaining irrelevant data records. Filtering stage 3 is designed to remove data records which occur less frequently, as observed by author of [35].

3.2.2.2.2. Stage 1: HTML Tag Structures

In this rule, Tree Wrap performs the filtering process based on Observation 4. Once the list of the data regions are obtained from BFS Extraction, Stage 1 involves removing data records that have less than three HTML tags in each and every group. The purpose of this filtering stage is to remove advertisement related information. We observe that advertisement usually contains simple structure to present its content (usually a list of hyperlinks as its content).

3.2.2.2.3. Stage 2: Similarity

In this section, we introduce the Dummy Tree Matching Algorithm which is developed to check the similarity of data records. We derive this method based on Observation 3 and our finding that data records share an important characteristic, i.e. the distinct tags of a tree and the total number of distinct tags in each level of the tree are nearly similar to those of the other trees of the group. Thus we are able to formulate a similarity check algorithm which can mimic the behavior of a full tree matching algorithm. Our approach is to carry out the similarity check of two trees by examining the distinct tags and comparing the total number of distinct tags in all levels of the trees. Our algorithm is simple but efficient and it can obtain similar results as those of a tree matching algorithm but it has a reduced time complexity. Details of our algorithm and its use in detecting similarity of data records and filtering dissimilar data regions are presented in the following subsections.

Our Dummy Tree Matching algorithm consists of a two stage screening procedure to check the similarity of a group of trees. Given a number of trees, our algorithm first examines the distinct tags of the first tree and that of the second tree. If almost all the distinct tags occur concurrently in the two trees (overall with say only one element different), then the trees pass the similarity test of the first stage and they are used for the second stage similarity test. In the second stage, we calculate the total number of distinct tags in all the levels of the first tree and that of the second tree. If the first two trees have almost equal number of distinct tags in all levels (overall with a difference of only one tag), then the two trees are considered similar according to the stage two criterion.

The first two trees are similar only if they pass the screening procedures of both stages. If the first two trees are similar, the first tree is retained for further processing and the second tree is then compared with the third tree of the group to check their similarity using Stages 1 and 2 of our screening algorithm. On the other hand, if the first two trees are not similar, the first tree will be removed and the second tree will be compared with the third tree to check their similarity. The screening procedures for both the above cases are repeated until the last tree is used for comparison.

Figure 6, Figure 7 and Figure 8 show data records presented in a tree form obtained from the DOM Tree of HTML pages. For simplicity, we show only two trees in each figure. We calculate the similarity of the two trees of Figure 6, Figure 7 and Figure 8 using our Dummy Tree Matching algorithm. In Figure 6, the distinct tags of the left tree are <table, tr, td, p> which are exactly similar to those of the right tree, so the left tree is similar to the right tree according to the rules of stage 1 of our similarity check. Further check using rules of stage 2 shows that the total number of distinct tags for all levels is 8 for the left tree (1 <table> tag in level 1, 1 <tr> tag in level 2, 1 <td> tag in level 3, 1 <p> and 1 <table> tag in level 4, 1 <tr> tag in level 5, 1 <td> tag in level 6, and 1 <p> tag in level 7 of the tree) and 4 for the right tree (1 <table> tag in level 1, 1 <tr> tag in level 2, 1 <td> tag in level 3, 1 <p> in level 4 of the tree). The overall similarity checks considering rules of both stage 1 and stage 2 indicate that the two trees are not similar and therefore the left tree will be removed. For data records of Figure 7, the distinct tags are <table, tr, td, div, a, p, b> for both the left and the right trees. The first screening procedure shows that the trees are similar. The total number of distinct tags in all levels is 7 for the left tree and 7 for the right tree respectively (1 <table> tag in level 1, 1 <tr> tag in level 2, 1 <td> tag in level 3, 1 tag in level 4, 1 <a> tag and 1 <p> tag in level 5, 1 <b> tag in level 6 of the trees). Therefore, the left tree is retained for further processing as the two trees are similar. For Figure 8, the distinct tags of the left and right trees are <tr, td, div, a> and <tr, td, p, b>, the rule in the first step says that the left tree should be deleted as the trees are considered not similar (out of 4 distinct tags, only tr and td tags are similar tags that exist in both the left and right trees). The screening procedures will be repeated using the second tree and third tree and so on until the last tree of the group is used if there are more than 2 trees.

```
1  Algorithm Similarity Check
2  for(int i:1 to numDataRecords){
3     //there are n nodes in a data record (O(n) complexity)
4     //total number of distinct tags (Step 1)
5                  int firstNumDistinctTags = getNumDistinctTags
6     (record(i));
7                  int secondNumDistinctTags = getNumDistinctTags
8     (record(i+1));
9     //compare left and right tree
10    if(abs(firstNumDistinctTags –
11    secondNumDistinctTags) > 1) {
12       //remove record(i); delete the left tree if not
13       similar
14    }//end if
15     //total number of distinct tags in all level (Step 2)
16    int firstDistinctTagsAllLevel =
17    getNumDistinctTagsAllLevel(record(i));
18    int secondDistinctTagsAllLevel =
19    getNumDistinctTagsAllLevel(record(i+1));
20    //compare left and right tree
21    if(abs(firstDistinctTagsAllLevel –
22    secondDistinctTagsAllLevel)>1){
23       //remove record(i); delete the left tree if not
24       //similar
25    }//end if
26}//end for
```

Fig. 5. The Dummy Tree similarity check algorithm

Calculations using the tree matching algorithm (e.g. DEPTA) show that trees of Figure 7 are similar and those of Figure 6 and Figure 8 are dissimilar. This algorithm gives results consistent with our Dummy Tree Matching algorithm. In general, single data record is usually represented by a regular expression which is applicable to all the data records.

In summary, the procedures used in our Dummy Tree Matching Algorithm to check the similarity of a group of trees are:

1. Examine the distinct tags of the first and second trees and if the trees have similar distinct tags, they pass the first test and will be used for the second test.

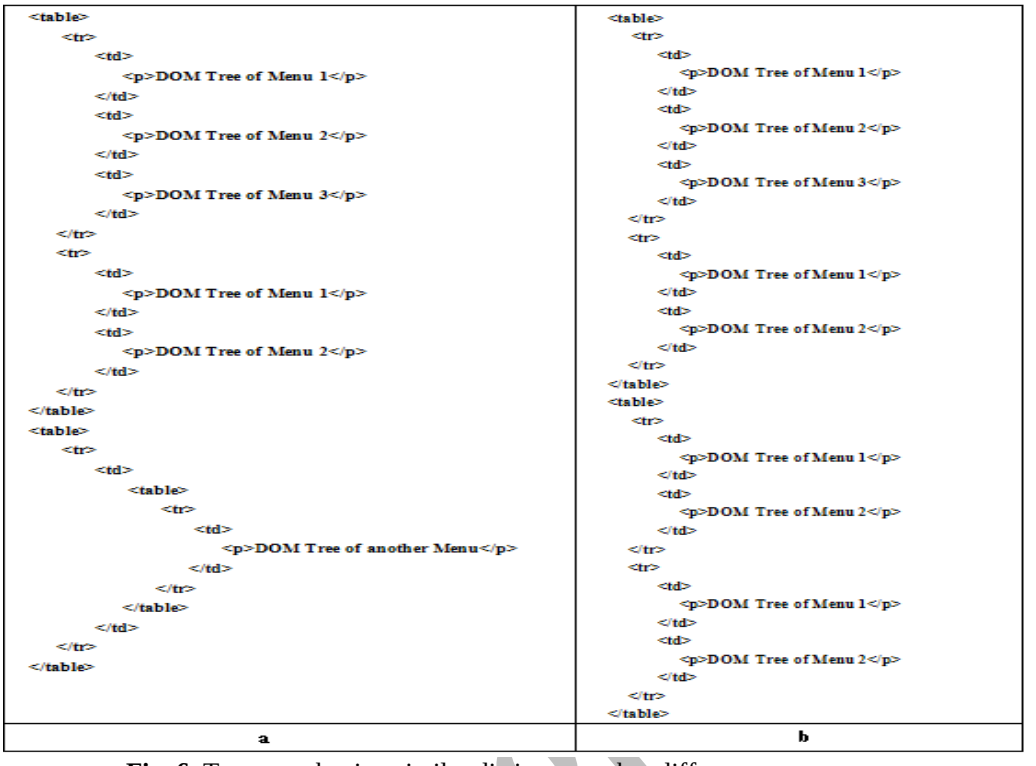

Fig. 6. Two trees having similar distinct tags, but different tree structures

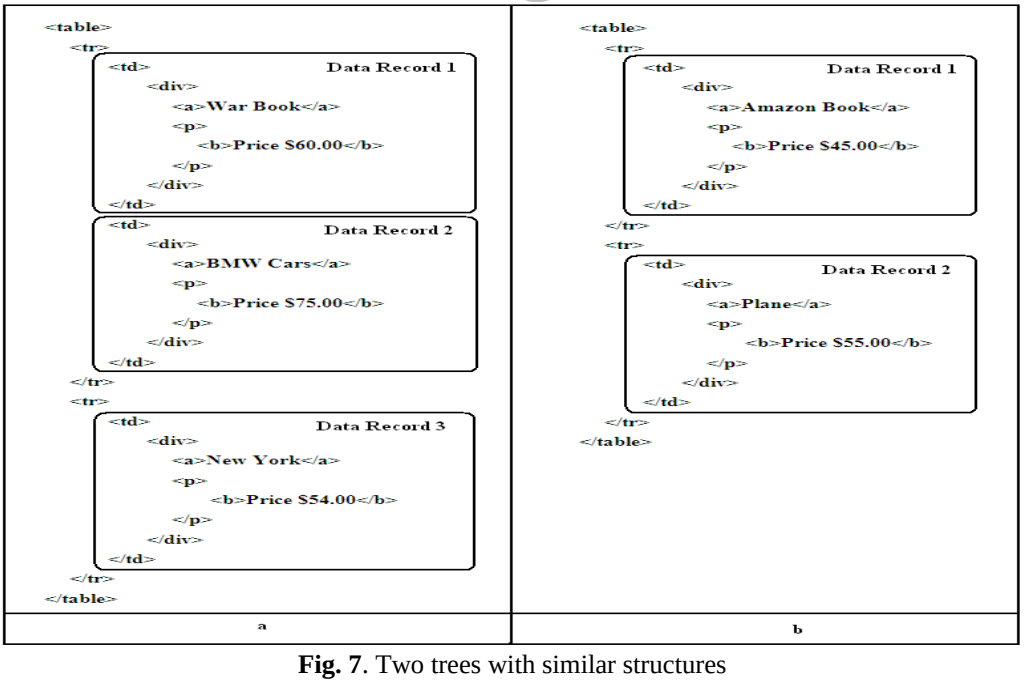

Fig. 7. Two trees with similar structures

2. Calculate and compare the number of distinct tags in all levels of the trees passing the first test, the trees are considered to pass second test if they have the same number of distinct tags in all levels of the trees.
3. The first tree and second tree are considered similar if they pass both the tests, for such a case, the first tree will be retained for further use. The trees are considered not similar if they fail to pass one of the tests carried out in Steps 1 and 2, therefore the first tree will be removed from the group. For both cases, the second tree will be compared with the third tree and the similarity tests are repeated for tree 2 and tree

3 and so on until the last tree in the group is used for comparison.

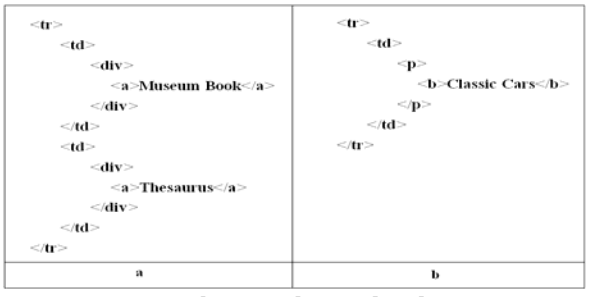

Fig. 8. Two trees having dissimilar distinct HTML tags

In general, there are two types of data regions left after the BFS stage, namely data regions with similar data records and data regions with entirely dissimilar data records. Dummy Tree Matching algorithm is designed to work by checking the data records of a data region and if they are not similar, they will be removed one by one and thus a data region with dissimilar data records will finally be filtered out. For data regions with similar data records, all these data records will be retained for further processing. The aim of this filtering stage using Dummy Tree Matching algorithm is to detect data regions with structurally similar data records normally exist in search engine results page, which are relevant to our study. Dissimilar data regions such as menus which determine the layout of a HTML page have structurally dissimilar data records and will be removed by our filtering algorithm.

We use Figure 9 and Figure 10 to demonstrate how our Dummy Tree Matching algorithm is used to remove dissimilar data regions and retain the similar data regions. Figure 9 shows the Lycos search engine results page. Figure 10 is the similar page presented in a tree form. As can be seen from Figure 9, Data Region 1 (solid rectangles in Figure 9, nodes <table> of Data Region 1 in Figure 10) contains repetitive nodes but these nodes are considered not similar (first <table > tag contains different sub tree from those of second and third <table> tags) because they have subtree structure with different sizes. Data records in Data Region 2 (Figure 9 and Figure 10), which are represented by the dotted rectangles in Figure 9 are similar because they have subtree structures of similar sizes. The same applies to Data Regions 3, 4 and 5. Using the Dummy Tree Matching algorithm, Data Region 1 is removed while other data regions (Data Regions 2, 3, 4, 5) are retained.

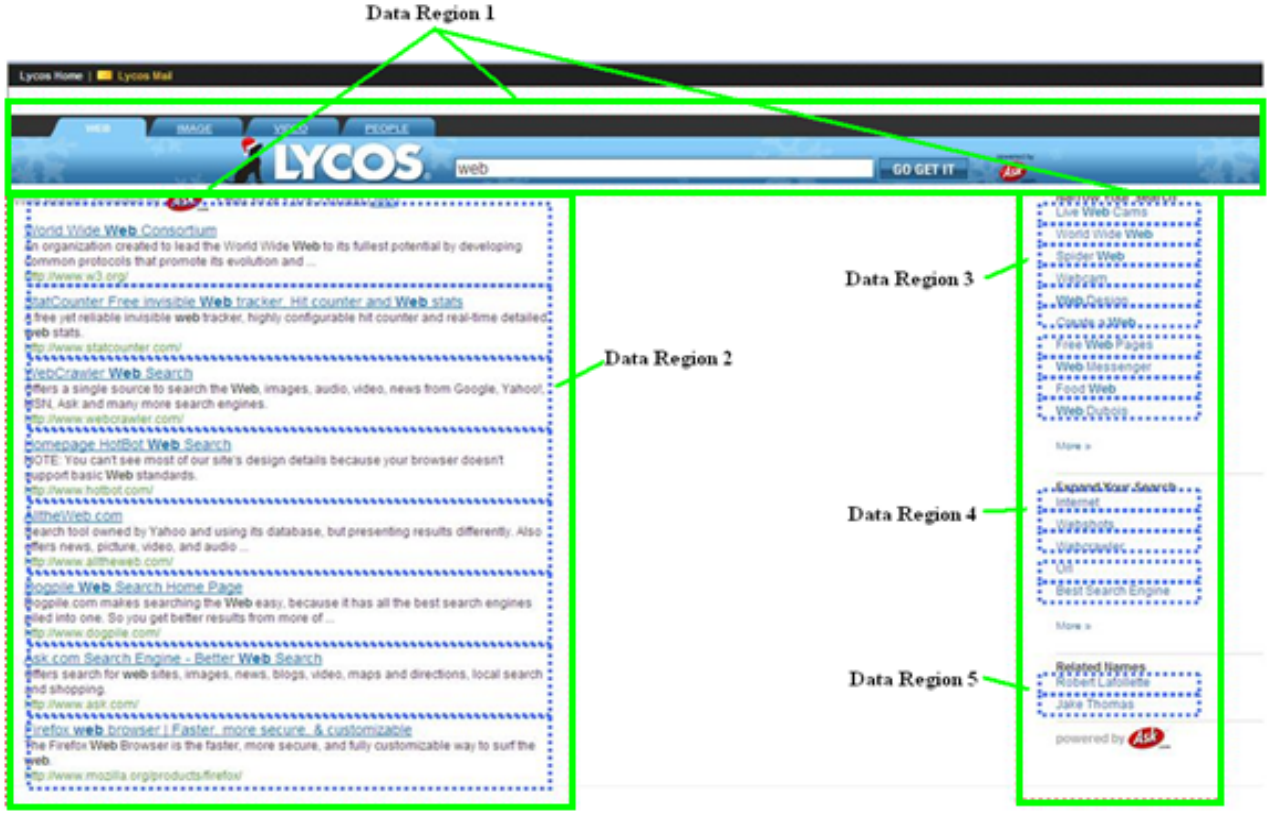

Fig. 9. An example of HTML page containing data regions.

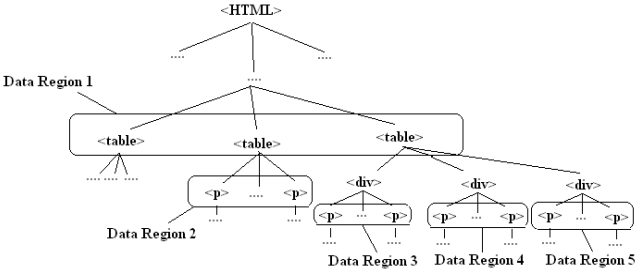

Fig. 10. The DOM Tree for the web page in Figure 9

3.2.2.2.4. Stage 3: Number of Nodes

In this stage, Tree Wrap will filter out irrelevant data records based on Observation 2. Data records occurring less than 3 times will be filtered out and excluded for further processing.

3.2.2.2.5. Stage 4: Scoring Function

After the completion of Stage 3, Tree Wrap will have a filtered list of data regions. From the list of available data regions, only one data region is chosen based on the scoring function of this stage assigned to each of the data regions. Filter Rule in Stage 4 is the most important component of the data extraction phase because a good scoring function is needed to differentiate the correct data region from incorrect ones.

This filter rule is derived based on Observations 1 and 2. Since data records occupy most of the space in a web page, we need to represent this property in our implementation. The best way to deal with this is to look at the text and images of the data records. It is noted that the correct data records have more text and images than the rest of the data records. Therefore we take into account the total length of the text and the number of images.

A constant value of 15 is added to the scoring function for every image detected in the data records. We also add a value of 1 to the scoring function for every character encountered in the data records. We decide to normalize the size of images with respect to the size of a character. Therefore, we choose a value of 15 to be added to the scoring function for each image detected assuming that one image has the size of 15 characters on average.

We notice that correct data records usually have more parent nodes than the rest of the potential data records. Therefore, we give a value of 150 for every parent node of the data records. There are several reasons for the adoption of the various values for the scoring function. A value of 150 is assigned for the data records' parent nodes as these nodes occur less frequently than the total text length and number of images. A relatively much smaller value is assigned for every character encountered in data records as characters tend to occur in large quantities. Images are generally larger than character, hence they are given a value of 15 instead of 1. Tree Wrap also recognizes separator tags such as
 and <hr> that tend to occupy space in data records. Therefore, whenever Tree Wrap encounters these tags, it will assign a value of 50 to them, assuming that each tag contains 50 characters on the average.

Tree Wrap calculates the value of the scoring function according to the following equation:

a=NumParentNodesLevel
b=TotalTextLength
c=NumImages
d=NumSeparatorTags
x=Data Region

$$Score(x) = (a*150) + ((b + (c*15) + (d*50))*5)$$

4. EXPERIMENTAL TESTS

4.1. Preparation of datasets

The datasets used in this study are taken from web pages that contain search engine results. These datasets are divided into four groups: Dataset 1 with 150, Dataset 2 with 119, Dataset 3 with 50 and Dataset 4 with 51 web pages. The data distribution for each of the datasets varies, ranging from academic sites, general sites to governmental sites.

The first dataset is prepared by the authors. The first dataset is randomly chosen from the internet. Dataset 1 is publicly available at http://hawksbill.infotech.monash.edu.my/~jlhong/WISH.html. The second and third datasets were taken from ViNT test bed, available at http://www.data.binghamton.edu:8080/vints/testbed.html. The fourth dataset is the TDBW v1.02, obtained from http://daisen.cc.kyushu-u.ac.jp/TBDW/.

It is worth noting that for all the datasets, each web page belongs only to a single web site. For example, Dataset 1 has 150 web pages, therefore there are 150 distinct web sites in it. The datasets contain different web pages, where none of the web pages chosen for one of the datasets will occur in any other datasets. Datasets 2 and 3 are the data originally used to test the performance of ViNT wrapper. These datasets are then used to test our wrapper as a useful indicator to see the accuracy and reliability of our wrapper.

The fourth dataset is the one used for testing in ViPER [21]. The purpose of using this dataset is to see the performance of our wrapper compared to ViNT [17] and DEPTA [35] when tested against a neutral publicly available dataset.

The total number of web pages used to test our wrapper amounts to 370. For all the datasets chosen, all the web pages contain semi structured data records.

We also evaluate the time complexity of our wrapper with respect to DEPTA and ViNT. We use datasets 1 to 4 to measure the running time used to

extract data records for ViNT, DEPTA and our wrapper. The average time to perform data extraction for the respective wrappers is recorded. We do not compare our wrapper with MDR as studies shown in ViNT [17] and DEPTA [35] indicate that both ViNT and DEPTA can perform better than MDR [4] and our experiments show that Tree Wrap is comparatively better than ViNT and DEPTA.

We further evaluate the robustness of our wrapper by replacing several components of our wrapper by available conventional techniques. This step is carried out to measure the reliability of our wrapper when these components are replaced. For example, we use string and tree edit distance algorithms as a replacement for Dummy Tree Matching for measuring the similarity of the structure of data records while we use visual cue (rectangular bounding box of a data region) as a replacement for heuristic scoring function to determine the relevant data region. Experimental results indicate that when using Dummy Tree Matching and heuristic scoring function, our wrapper is able to run faster and more accurately. We also test the reliability of our wrapper by removing the Stage 1 (Filter HTML Tags) and by changing the number of nodes for Stage 3 (Filter Number of Nodes) of our filtering modules. Experimental tests indicate that the above do not affect the accuracy of our wrapper.

4.2. Method of Evaluation

HTML web page parsing is a difficult task. Therefore, it is very unlikely that a publicly available parser can achieve 100% parsing rate. For those web pages that the parser failed to parse, we rule them out from consideration in our evaluation. The experiment was conducted with a PC specification of Pentium 4 2.4 Ghz, with 1GB of RAM memory. The measures of wrapper's efficiency are based on three factors, the number of actual data records to be extracted, the number of extracted data records from the test cases, and the number of correct data records extracted from the test cases. Based on these three values, precision and recall are calculated according to the formulas:

*Recall=Correct/Actual*100*

*Precision=Correct/Extracted*100*

4.3. Data Extraction Results

4.3.1. Dataset 1

TreeWrap outperforms DEPTA and ViNT both in terms of recall and precision rates (Table1). The strength of our wrapper lies in the testing of Dataset 1. The result in Table1 shows that our wrapper significantly outperforms the works of ViNT and DEPTA. Our wrapper incorporates an accurate tree matching algorithm which could detect similarity of data records. Besides, the filtering technique used in Component 2, Stage 4 allows extraction of correct data records. However, there are several odd cases which our wrapper did not consider for. Some search engine results have search identifier (e.g. Search query *"Web"* returns 10 results) which also has similar parent node to that of relevant data records. In some cases, this identifier also has similar tree structures as data records' tree structures. This search identifier will eventually pass through all the filtering stages successfully and extracted as data records.

Table 1. Results of Dataset 1 for Tree Wrap, ViNT and DEPTA

Term	DEPTA	ViNT	Tree Wrap
Actual	1766	1766	1766
Extracted	1258	2015	1742
Correct	1183	1486	1729
Recall	66.99%	84.14%	97.90%
Precision	94.04%	73.75%	99.25%

4.3.2. Dataset 2

The test on Dataset 2 shows that Tree Wrap has improvements over the works of DEPTA and ViNT (Table 2). Our result shows that we obtained better recall and precision rates than that of DEPTA and ViNT. This could be attributed to the fact that our dummy tree matching algorithm is more efficient in detecting the similarity of structured data records than the algorithms in DEPTA and ViNT.

Table 2. Results of Dataset 2 for Tree Wrap, ViNT and DEPTA

Term	DEPTA	ViNT	Tree Wrap
Actual	1655	1655	1655
Extracted	1027	1612	1644
Correct	994	1583	1635
Recall	60.06%	95.65%	98.79%
Precision	96.79%	98.20%	99.45%

4.3.3. Dataset 3

Similar to Dataset 2, test on Dataset 3 shows improvements in our work compared to the works of DEPTA and ViNT (Table 3).

Table 3. Results of Dataset 3 for Tree Wrap, ViNT and DEPTA

Term	DEPTA	ViNT	TreeWrap
Actual	830	830	830
Extracted	655	822	812
Correct	612	806	806
Recall	73.73%	97.11%	97.11%
Precision	93.44%	98.05%	99.26%

4.3.4. Dataset 4

Tree Wrap wrapper performs better than DEPTA and ViNT on this dataset (Table 4). As shown in the table, Tree Wrap produces a recall value of over 10% higher than ViNT.

Table 4. Results of Dataset 4 for Tree Wrap, ViNT and DEPTA

Term	DEPTA	ViNT	TreeWrap
Actual	693	693	693
Extracted	402	661	692
Correct	388	618	688
Recall	55.99%	89.18%	99.28%
Precision	96.52%	93.49%	99.42%

4.3.5. Running time of our wrapper

We evaluate the running time of our wrapper with respect to DEPTA [35] and ViNT [17]. We compare Dummy Tree Matching algorithm with the tree matching algorithm of DEPTA. Experimental results show that our wrapper runs faster than DEPTA (Table 5). This result signifies that our dummy tree matching algorithm runs in a time complexity smaller than that of DEPTA. Our wrapper also runs faster than ViNT, a visual wrapper using visual cue and DOM Tree structure of data records.

Table 5. Running time of our wrapper, DEPTA, and ViNT

Wrappers	Running time (avg in s)
TreeWrap	215
DEPTA	334
ViNT	1136

4.3.6. Evaluation of the Filter HTML Tags

To evaluate the stability and effect of Filter HTML Tags in our wrapper, we test our wrapper on Datasets 1, 2, 3, and 4 by removing the HTML Tags Filter. Results are presented in Table 6. Our results show that removing the HTML Tags Filter has no marked effect on the overall performance of our wrapper. Slightly higher accuracy rates are noted in Datasets 1 and 2. The higher accuracy rate could be attributed to the presence of web pages containing search results with simple structures, hence our wrapper is able to extract these data records. However, as shown in Table 6, with HTML Tags Filter included, there is a decrease in running time of our wrapper although the precision and recall rates are slightly reduced. This additional increase in speed will be helpful in meta search engine application and large scale web comparisons.

4.3.7. Evaluation of the Similarity Check of Data Records

As stated earlier, we test the performance of our wrapper by replacing Dummy Tree Matching with String Edit Distance algorithms. We incorporate two types of string edit distance techniques (the simple Euclidean distance and the more common Levenshtein edit distance [30]) using the work of http://www.dcs.shef.ac.uk/~sam/stringmetrics.html and test our wrapper on Datasets 1 to 4. Experimental results indicate that String Edit Distance algorithms cannot improve the performance of our wrapper as can be seen from the recall and precision rates obtained (Table 7). This test also shows that Dummy Tree Matching algorithm is an important component of our wrapper as it contributes in the accuracy of data extraction.

We also test the performance of our wrapper by replacing our Dummy Tree Matching algorithm by the tree matching algorithm of DEPTA [35]. Results indicate that although the tree matching algorithm of DEPTA is accurate, it is slow in operation as can be seen from the running time results (Table 8). Our Dummy Tree Matching algorithm is not only accurate, but it can also perform the function of a tree matching algorithm at a reduced running time complexity of O(n) (n is the number of nodes in the tree).

4.3.8. Evaluation of Filter Number of Nodes

We also test our wrapper by changing the number of data records to be filtered out from 3 to 2 in the stage 3 filtering process: Number of nodes. Results show that the recall and precision rates will decrease slightly when 2 data records instead of 3 data records are used (Table 9). This is because the menus, a data region containing highly dissimilar data records are extracted instead of relevant data region. As relevant data region which contains only two data records is smaller in size compared to other data region such as menu bars, they are more difficult to be extracted compared to other data region.

4.3.9. Evaluation of Locating Correct Data Region

To further measure the reliability of our wrapper, we test our wrapper by replacing the component which determines the correct data region. We used the algorithm of VSDR (determine the large and centrally located data region) as a replacement for Component 2, Stage 4 of our wrapper (Largest Score Filtration). As the algorithm in VSDR requires visual cue for its implementation, we used ICE browser available at http://www.icesoft.com/ as part of our wrapper design. This browser is able to parse a HTML page and provide visual cue in addition to DOM Tree. Experimental results show that our technique is able to obtain more accurate results than the technique using the rectangular bounding box to locate the relevant data region (Table 10). However, our test also indicates that the technique using rectangular bounding box is slow in

operation, thus it is a limitation for large scale web comparisons. The increase in running time is due to the extra processing needed to obtain visual information from the underlying browser rendering engine during the parsing phase. Experimental results also indicate non visual wrappers like Tree Wrap which use fast heuristic techniques could also attain similar performances as visual wrappers.

Table 6. Test on Filter HTML Tags

Datasets	With HTML Tags Filter			Without HTML Tags Filter		
	Recall	Precision	Time (avg. msec)	Recall	Precision	Time (avg. msec)
Dataset 1	97.90%	99.25%	215	98.19%	99.54%	254
Dataset 2	98.79%	99.45%	187	98.97%	99.64%	201
Dataset 3	97.11%	99.26%	196	97.11%	99.26%	208
Dataset 4	99.28%	99.42%	178	99.28%	99.42%	185

Table 7. Performance of Dummy Tree Matching and String Edit Distance

Datasets	Dummy Tree Matching		Levenshtein distance		Euclidean distance	
	Recall	Precision	Recall	Precision	Recall	Precision
Dataset 1	97.90%	99.25%	80.29%	77.61%	60.79%	69.67%
Dataset 2	98.79%	99.45%	67.61%	81.32%	41.69%	64.13%
Dataset 3	97.11%	99.26%	75.18%	75.72%	36.63%	53.62%
Dataset 4	99.28%	99.42%	79.65%	81.42%	64.21%	75.28%

Table 8. Performance of Dummy Tree Matching and DEPTA Tree Matching algorithm

Datasets	Dummy Tree Matching			DEPTA Tree Matching		
	Recall	Precision	Time (avg. msec)	Recall	Precision	Time (avg. msec)
Dataset 1	97.90%	99.25%	215	93.54%	98.10%	638
Dataset 2	98.79%	99.45%	187	95.83%	98.88%	427
Dataset 3	97.11%	99.26%	196	95.06%	97.17%	441
Dataset 4	99.28%	99.42%	178	94.95%	98.21%	354

Table 9. Test on Filter Number of Nodes

Datasets	Less than 3 Number of Nodes		Less than 2 Number of Nodes	
	Recall	Precision	Recall	Precision
Dataset 1	97.90%	99.25%	95.64%	99.17%
Dataset 2	98.79%	99.45%	96.56%	99.25%
Dataset 3	97.11%	99.26%	94.82%	98.62%
Dataset 4	99.28%	99.42%	95.82%	98.81%

Table 10. Performance of Scoring Function and Visual Cue

Datasets	Scoring Function			Rectangular Bounding Box		
	Recall	Precision	Time (avg.)	Recall	Precision	Time (avg.)
Dataset 1	97.90%	99.25%	215	89.69%	94.00%	552
Dataset 2	98.79%	99.45%	187	89.55%	96.05%	386
Dataset 3	97.11%	99.26%	196	96.63%	98.53%	328
Dataset 4	99.28%	99.42%	178	95.82%	97.36%	383

5. CONCLUSIONS

In this study we propose a non-visual wrapper Tree Wrap which is able to extract data records from structured web pages. Our results show that our wrapper is able to obtain results as well as and in most cases better than the current state of the art visual wrappers such as ViNT and DEPTA. Our approach uses a set of filtering methods based on the DOM Tree structure of data records and a more accurate algorithm to calculate the space occupied by the data region. Our dummy tree matching algorithm simplifies the complicated process of comparing every node of each

tree to check the similarity of two trees as used in the tree matching algorithm. We use the number of nodes of a tree to compare the similarity of two trees. This procedure improves the overall running time without compromising the accuracy, making it suitable for large scale web comparisons. Our stability tests on each of the heuristic data extraction components of our wrapper also show that Dummy Tree Matching and scoring function are the most important components for extracting data records.

REFERENCES

[1] Weiyi Meng H. and Yu, C.; **"Mining Templates from Search Result Records of Search Engines"**, *in Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining San Jose, California, USA*: ACM, (2007)

[2] Zhao H., Meng W., Wu Z., Raghavan V. and Yu C.; **"Fully Automatic Wrapper Generation for Search Engines"**, *in Proceedings of the 14th international conference on World Wide Web Chiba, Japan*: ACM, (2005)

[3] Ricardo A. Baeza-Yates, **"Algorithms for String Searching"** *SIGIR Forum*, Vol. 23, pp. 34-58, (1989)

[4] Zhai Y. and Liu B.; **"Structured Data Extraction from the Web Based on Partial Tree Alignment"** *IEEE Transaction on Knowledge and Data Engineering*, Vol. 18, pp. 1614-1628, (2006)

[5] Liu B. and Zhai Y.; **"NET – A System for Extracting Web Data from Flat and Nested Data Records"**, in Web Information Systems Engineering – WISE 2005, pp. 487-495, (2005)

[6] Chang,Ch.H, Kayed M., Ramzy Girgis M. and Shaalan Kh.; **"A Survey of Web Information Extraction Systems"**, *Transactions on Knowledge and Data Engineering*, Vol. 18, pp. 1411-1428, (2006)

[7] Tao Cui and David W. Embley, **"Automatic Hidden-Web Table Interpretation, Conceptualization, and Semantic Annotation",** *Data Knowl. Eng*., Vol. 68, pp. 683-703, (2009)

[8] Gusfield D.; **Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology**: Cambridge University Press, (1997)

[9] Sankoff David and Kruskal Joseph, **Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison:** Technical Report of Center for the Study of Language and Inf, (1999)

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

[11] Valiente G.; **"An Efficient Bottom-up Distance between Trees",** *in Proc. Eighth Int'l Symp. String Processing and Information Retrieval*, pp. pp. 212-219, (2001)

[12] Das G., Fleischer R., Gasieniec L., Gunopulos D. and Karkkainen Juha**; "Episode Matching"**, *in Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, (1997)

[13] Miao G., Tatemura J., Hsiung W.P., Sawires A. and Louise E. Moser**; "Extracting Data Records from the Web Using Tag Path Clustering",** *in Proceedings of the 18th international conference on World Wide Web, Spain, Madrid*, (2009)

[14] Navarro Gonzalo; **"A Guided Tour to Approximate String Matching"**, *ACM Comput. Surv*., Vol. 33, pp. 31-88, (2001)

[15] Alberto H.F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira; **"A Brief Survey of Web Data Extraction Tools"**, *SIGMOD Rec*., Vol. 31, pp. 84-93, (2002)

[16] Apostolico A. and Guerra C.; **The Longest Common Subsequence Problem Revisited,** Algorithmica 2, (1987)

[17] Hong J.L., Siew E. and Egerton S.; **"DTM-Extracting Data Records from Search Engine Results Page using Tree Matching Algorithm",** *in Proceedings of the 1st international conference on Soft computing and pattern recognition: IEEE*, (2009)

[18] Hong J.L., Siew E. and Egerton S.; **"Information Extraction for Search Engines Using Fast Heuristic Techniques",** *Data Knowledge Engineering*, Vol. 69, pg 169-196, (2010)

[19] Wang J. and Frederick H. Lochovsky; **"Data Extraction And Label Assignment For Web Databases",** *in Proceedings of the 12th international conference on World Wide Web Budapest, Hungary: ACM*, (2003)

[20] Simon K. and Lausen G.; **"Viper: Augmenting Automatic Information Extraction with Visual Perceptions",** *in Proceedings of the 14th ACM international conference on Information and knowledge management Bremen, Germany: ACM*, (2005)

[21] Tai K. Ch.; **"The Tree-to-Tree Correction Problem",** *J. ACM*, Vol. 26, pp. 422-433, (1979)

[22] Li L., Liu Y., Obregon A. and Weatherston M.; **"Visual Segmentation-Based Data Record Extraction from Web Documents",** *in Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference*, pp. 502-507, (2007)

[23] lvarez M., Pan A., Raposo J., Bellas F. and Cacheda F.; **"Extracting Lists Of Data Records From Semi-Structured Web Pages",** *Data Knowl. Eng*., Vol. 64, pp. 491-509, (2008)

[24] Song M., Song Il-Yeol, Hu Xiaohua, and Robert B. Allen; **"Integration of Association Rules and Ontologies for Semantic Query Expansion**", *Data Knowl. Eng*., Vol. 63, pp. 63-75, (2007)

[25] Arasu A. and Garcia-Molina H.; **"Extracting Structured Data from Web Pages",** *in Proceedings of the 2003 ACM SIGMOD international conference on Management of data San Diego, California: ACM*, (2003)

[26] Saul B. Needleman and Christian D. Wünsch; **"A General Method Applicable To the Search for Similarities In The Amino Acid Sequences Of Two Proteins",** *Journal of Molecular Biology*, (1970)

[27] Jiang T., Wang L. and Zhang K.; **"Alignment of Trees - An Alternative to Tree Edit**", **"***in Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching: Springer-Verlag*, (1994)

[28] Crescenzi V., Mecca G. and Merialdo P.;

"RoadRunner: Towards Automatic Data Extraction from Large Web Sites", *in Proceedings of the 27th International Conference on Very Large Data Bases: Morgan Kaufmann Publishers Inc.*, (2001)

[29] Levenshtein Vladimir I; **"Binary Codes Capable Of Correcting Deletions, Insertions, And Reversals"** *Soviet Physics Doklady*, Vol. 10, pp.707, (1966)

[30] Liu Wei, Meng Xiaofeng, and Meng,Weiyi; **"Vision-based Web Data Records Extraction"**, *ACM Ninth International Workshop on the Web and Databases* (WebDB 2006), (2006)

[31] Liu W., Meng X., and Meng W.; **"ViDE: A Vision-based Approach for Deep Web Data Extraction",** *IEEE Transaction on Knowledge and Data Engineering*, (2009)

[32] Su W., Wang J. and Frederick H. Lochovsky; **"ODE: Ontology-assisted Data Extraction",** *ACM Transactions on Database Systems*, (2009)

[33] Yang W.; **"Identifying Syntactic Differences between Two Programs",** *Softw. Pract. Exper.*, Vol. 21, pp. 739-755, (1991)

[34] Zhai Y. and Liu B.; **"Web Data Extraction Based on Partial Tree Alignment",** *in Proceedings of the 14th international conference on World Wide Web Chiba, Japan: ACM*, (2005)

[35] Liu B., Grossman R. and Zhai Y.; **"Mining Data Records in Web Pages"**, *in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining Washington, D.C.: ACM*, (2003)