Peer Matrix Alignment: A New Algorithm

Mohammed Kayed

Faculty of Science, Beni-Suef University, Egypt
mskayed@yahoo.com

Abstract. Web data extraction has been one of the keys for web content mining that tries to understand Web pages and discover valuable information from them. Most of the developed Web data extraction systems have used data (string/tree) alignment techniques. In this paper, we suggest a new algorithm for multiple string (peer matrix) alignment. Each row in the matrix represents one string of characters, where every character (symbol) corresponds to a subtree in the DOM tree of a web page. Two subtrees take the same symbol in the peer matrix if they are similar, where similarity can be measured using either structural, content, or visual information. Our algorithm is not a generalization of 2-strings alignment; it looks at multiple strings at the same time. Also, our algorithm considers the common problems in the field of Web data extraction: missing, multi-valued, multi-ordering, and disjunctive attributes. The experiments show a perfect alignment result with the matrices constructed from the nodes closed to the top (root) and an encourage result for the nodes closed to the leaves of the DOM trees of the test web pages.

Keywords: Text Alignment, Tree Alignment, Web Data Extraction, Information Extraction.

1 Introduction

Deep web presents a huge amount of useful information which is usually formatted for its users. So, extracting relevant data from various sources to be used in web applications becomes not easy. Therefore, the availability of robust, flexible Information Extraction (IE) or Wrapper Induction (WI) systems that transform Web pages into program-friendly structures such as a relational database will become a great necessity. For relevant data extraction, unsupervised IE systems exploit the fact that web pages of the same Web site share the same template since they are encoded in a consistent manner across all the pages; i.e., these pages are generated with a predefined template by plugging data values. Finding such a common template requires as input multiple pages (page-wide IE systems; e.g., RoadRunner [1], EXALG [2], and FiVaTech [3]) or a single page containing multiple records (record-wide IE systems; e.g., IEPAD [4], DeLa [5], and DEPTA [6]).

A crucial step for most web data extraction systems (record/page level systems) is alignment: either string alignment (e.g., IEPAD and RoadRunner) or tree alignment (e.g., DEPTA). Alignment of attribute values in multiple data objects (strings/trees) is a challenging task as these attributes are subject to the following variations [7]:

P.-N. Tan et al. (Eds.): PAKDD 2012, Part II, LNAI 7302, pp. 268–279, 2012.
© Springer-Verlag Berlin Heidelberg 2012

- An attribute may have zero or more values in a data-object. If the attribute has zero value, it is called a missing attribute; if it has more than one value, it is called a multi-valued attribute. A book's author may be an example of multi-valued attribute, whereas a special offer is an example of missing attribute.
- The set of attributes (A_1, A_2, …, A_k) may have multiple ordering, i.e., an attribute A_i may have variant positions in different instances of a data-object. We call this attribute a multi-ordering attribute. For example, a movie site might list the release date before the title for movies prior to 1999, but after the title for recent movies.
- An attribute may have variant formats along with different instances of a data object. If the format of an attribute is not fixed, we might need disjunctive rules to generalize all cases. For example, an e-commerce site might list prices in bold face, except for sale prices which are in red. So, price would be an example of a variant-format (disjunctive) attribute in this site. On the other hand, different attributes in a data-object may have the same format, especially in table presentation, where single <TD> tags are used to present various attributes.
- Most IE systems handle input documents as strings of tokens as they are easier to process than strings of characters. Depending on the used tokenization methods, sometimes an attribute cannot be decomposed into individual tokens. Such an attribute is called an untokenized attribute.

Untokenized attributes can be processed by further processing after the alignment step, while missing, multi-valued, multi-ordering, and disjunctive attributes are handled during the alignment step. The effectiveness of an alignment algorithm relies on its capability to handle such problems. In this paper, we suggest a new alignment algorithm to be used as a part of web data extraction systems. Our algorithm considers the above four mentioned problems. Also, to align multiple data objects, our algorithm processes all objects at the same time. Our algorithm doesn't consider the problem of multiple-objects alignment as a generalization of 2-objects alignment. So, it has a global (better) view for the input objects (strings/trees).

The rest of the paper is organized as follows. Section 2 defines the peer matrix alignment problem. Our proposed alignment algorithm and different examples that we suggest to clarify the algorithm are discussed in sections 3 and 4, respectively. Section 5 describes our experiments. The related works are presented in Section 6. Finally, section 7 concludes our work.

2 Problem Definition

String alignment is simpler than tree alignment. Since web pages used with IE systems are tree structured (DOM trees), tree alignment will become a great necessity. Like FiVaTech, our algorithm considers the tree structure of the objects to be aligned, but we simplify the problem by converting it into string alignment as follows. We consider all of the objects (web pages) to be aligned are tree structured and have the same root p. Our algorithm collects all (first-level) child nodes of p in a matrix M, where each column keeps the child nodes for one root node p. Every node in the

matrix takes a symbol which actually denotes a subtree. All similar subtrees (peer nodes) are denoted by the same symbol. Similarity here can be measured by any suitable technique such as tree-edit distance. The peer matrix shown in Fig. 1b is constructed from the three trees of root *p* in Fig. 1a. The matrix has three columns; each one includes all child nodes of one root *p*, where two similar nodes take the same symbol. Now, the problem is transformed into multiple-string (peer matrix) alignment, where each string corresponds to one column in the matrix. Handling of the problems: missing, multi-valued, multi-ordering, and disjunctive attributes is based on this very important alignment step. The output of this step is an aligned list in which missing, repetitive, disjunctive, and multi-ordering patterns can be identified very easily. For example, as shown in Fig. 1d, the output aligned list has one repetitive pattern (BCD), where the two symbols B and D are optional (missing attributes) in this pattern.

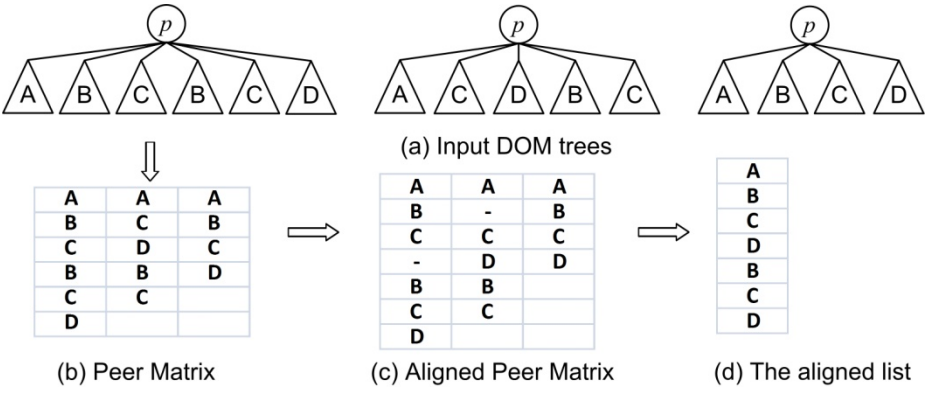

Fig. 1. Peer matrix construction

Definition (Aligned Row): A row in the peer matrix is called an aligned row in two cases. The first case is occurred when all symbols in this row are the same (or disappear in some columns of the row). The second case is occurred when the row has different symbols, but these symbols correspond to leaf nodes (text or <img>) such that each symbol appears only in its residing column (i.e., if a symbol exists in a column c, then all other columns outside the current row in the matrix do not contain this symbol). All of the rows above an aligned row must be also aligned.

Definition (Aligned Peer Matrix): A peer matrix is aligned if all of the rows in the matrix are aligned. Fig. 1c is an example of an aligned peer matrix. As shown in the aligned peer matrix, the symbol '-' refers to a null value which has been added after shifting some symbols down to patch an empty space.

Definition (Aligned List): If M is an aligned peer matrix, so each aligned row in the matrix M is represented by a symbol which is either the same as the symbol in the row (if the row has a same symbol) or an asterisk (if the row has different symbols correspond to leaf nodes). Fig. 1d is the aligned list corresponds to the one in Fig. 1c.

Definition (Peer Matrix Alignment Problem): Given a peer matrix as input, the alignment problem is to modify the content of the matrix by shifting down, swapping, or replacing symbols in the matrix to transform it into an aligned peer matrix.

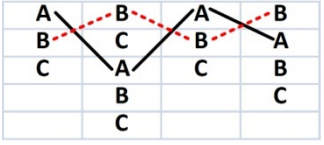

Fig. 2. Examples of regular and irregular zigzag lines L_B and L_A, respectively

3 The Proposed Algorithm

Given a peer matrix as input, we start the matrix row by row (up-down fashion) and align each row using its contents and the contents of the other rows below it, to get an aligned peer matrix. Each row r is aligned by processing of zigzag lines that have been drawn to connect among the symbols in r and other symbols down r in the matrix. Zigzag lines are drawn as follows. For each symbol s in r, we draw a sequence of lines (a zigzag line) L_s that connect among all first occurrences of the symbol s in each column of the current row r or the rows below r. The zigzag line L_s of the symbol s at row r passes through different occurrences of s in different columns in r or below r with at most one occurrence of s in each column. Each occurrence of s is connected by a line with the first occurrence in the right/left hand side column. If the symbol does not appear in the right/left column, the line will pass to connect the first occurrence in the next column, and so on. All occurrences of s in the row r belong to the same zigzag L_s. If the symbol does not appear again in the row or below the row, we call it a *non-zigzag symbol*. This means, a non-zigzag symbol is not belonging to any zigzag line in the row. Fig. 2 shows two zigzag lines L_A and L_B at the first row. By drawing zigzag lines for the symbols in the row r, the challenge here is how we can process these zigzags to align the row r. Although, zigzag lines are different for different columns order in the matrix, our algorithm solves the problem in general and covers all suitable cases. Before we go further in this section to discuss our proposed alignment algorithm, we give some definitions that are important to the algorithm.

Let l is a line in a zigzag L_s that connects two occurrences of s at the two locations (r_1,c_1) and (r_2,c_2) in the matrix, we define the vertical span of l as $(r_2 - r_1+1)$ and the horizontal span as $(c_2 - c_1+1)$. Also, we call a zigzag L_s horizontally covers the peer matrix if it is started at the first column and terminated at the last column of the matrix. The vertical spans of the three lines of L_A in Fig 2 are 3, 3, and 2, respectively. Also, the two zigzag lines L_A and L_B in Fig 2 horizontally cover the peer matrix.

Definition (A Repetitive Pattern): A pattern P (a sequence of one or more consecutive symbols in a column) is called repetitive if it has more than one occurrence in some column of the matrix. If P appears at most once in each column of the matrix, we call it a free (non-repetitive) pattern.

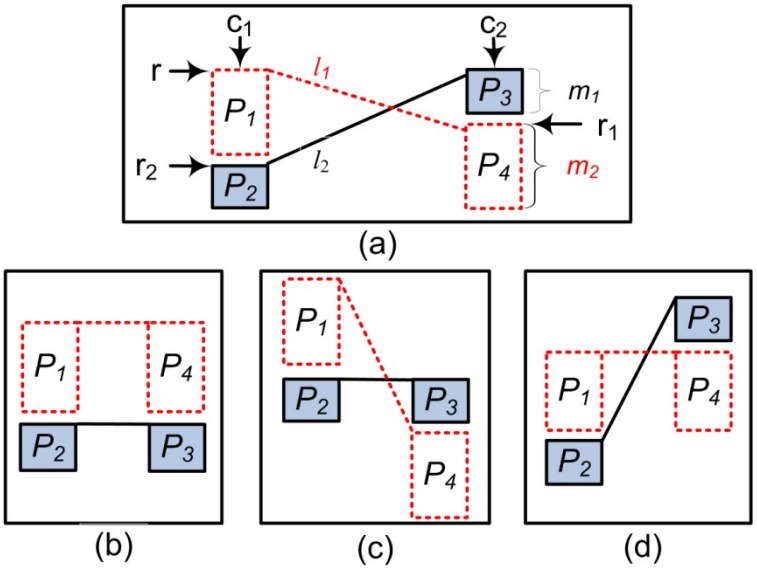

Fig. 3. A left-most cross (a) and three alignment solutions (b), (c), and (d)

Definition (Symbol Span): The span of a symbol *s* is defined as the maximum number of different symbols (without repetition) between any two consecutive occurrences of *s* in each column plus one; i.e., span of *s* represents the maximum possible cycle length of *s*. If *s* is a non-repetitive symbol, then its span value will be 0. For example, the span of the symbol B in the matrix in Fig. 2 is 3 (there are two different symbols, C and A, between the two occurrences in the second column).

In our proposed algorithm, we use the calculated span value for a symbol *s* to control (restrict) the process of shifting down *s* in the matrix. Shifting down the symbol *s* must not violate the calculated symbol span value. This means, if *s* occurred only once in a column, it can be shifted down to anywhere in the column. But, the symbol *s* which is occurred at row *r* and column *c* cannot be shifted down if it appears up in *c* at a row r′ (i.e., r > r′) such that r-r′ ≥ span(*s*). For example, the symbol B in the fourth row and the second column of the matrix in Fig. 2 cannot be shifted down because its span is 3 and it appears up at the first row of the column.

Definition (Regular Zigzag): a zigzag L_s is *regular* if the vertical spans of all of its lines are equal, and it horizontally covers the matrix. Otherwise it is called *irregular*. Fig 2 shows examples of regular and irregular zigzag lines L_B and L_A, respectively.

Definition (A Top Horizontal Zigzag): a zigzag L_s is called horizontal if all of its lines are horizontal. If there is only one horizontal zigzag in a row, we call it a top horizontal. If there is more than one horizontal zigzag line, we call the one with a maximum number of lines a top horizontal zigzag.

Definition (Left-most Cross): Let l_1 is a line ((r,c_1), (r_1,c_2)) which belongs to a zigzag L_a in a row *r* as shown in Fig. 3(a). We call the cross between the line l_2 ((r,c_2),

(r_2,c_1)) in a zigzag L_b and l_1 a left-most cross if the vertical distance between r_2 and r (m_2= r_2-r) is a minimum. As shown in Fig. 3(a), the cross makes four patterns in the peer matrix: P_2 and P_3 are the two patterns of lengths m_1 under the two ending points of l_2, while P_1 and P_4 are the two patterns of lengths m_2 under l_1.

To align the row r which has a left-most cross as in Fig.3 (a), we have three possible different alignment cases. The first case, case I, is occurred if either P_1 or P_3 is optional (missing pattern in c_2 or c_1, respectively, as shown in Figures 3(c) and (d)). The second case, case II, is occurred when the two patterns P_1 (P_3) and P_2 (P_4) are multi-ordering in the column c_1 (c_2), respectively, as shown in Fig. 3(b). Finally, the third case, case III, is occurred when the two patterns P_1 and P_3 have disjunctive attributes (i.e., same data presented in different formats). Our algorithm works as follows to distinguish between these three cases I-III.

If either $P_1 \neq P_4$ or $P_2 \neq P_3$, we align the row as case I (P_1 or P_3 is optional). Particularly, if $P_1 = P_4$ (while $P_2 \neq P_3$), we identify P_3 as optional (i.e., P_1 is shifted down in the column c_1 a distance m_1). If $P_2 = P_3$ (while $P_1 \neq P_4$), we identify P_1 as optional (i.e., P_3 is shifted down in the column c_2 a distance m_2). Finally, if both $P_1 \neq P_4$ and $P_2 \neq P_3$, we shift down either P_1 or P_3 based on some criteria as we will discuss later.

If both $P_1 = P_4$ and $P_2 = P_3$, the algorithm deals with the problem as follows. As in Fig. 3 (c) and (d), to identify the pattern P_1 (P_3) as optional in c_2 (c_1), it is necessary (but not sufficient) that P_1 (P_3) is repetitive in the matrix, respectively. So, if both P_1 and P_3 are non-repetitive patterns, we identify P_1 (P_3) and P_2 (P_4) as multiple-ordering patterns (case II) in the column c_1 (c_2), respectively. The challenge here is occurred when either P_1 or P_3 is repetitive. Experimentally, we have observed that case I (either P_1 or P_3 is optional) is mostly the correct alignment choice. An exception is occurred when P_1 and P_3 have disjunctive patterns (the row is aligned using case III).

Figure 4 shows an example of case III (disjunctive attributes) when a search engine web site presents a list of resulted web pages as links (<A> tag), except the current page is presented using <STRONG> tag (i.e., the two tags <A> and <STRONG> are disjunctive). So, the two cases I and III are possible when either P_1 or P_3 is repetitive. We handle this problem and decide the correct alignment case based on the following assumption. The two patterns a_1 and a_2 are disjunctive, if one of the two patterns (say a_1) appears randomly among different consecutive occurrences of the other one (a_2) in each column of the matrix. So, we assume that, there is a left-most cross in two columns c_1 and c_2 such that one of the two patterns P_1 or P_3 is a sequence of the repetitive pattern a_2, while the other one is a_1. If so, we replace all occurrences (in the matrix) of a_1 by a_2 and mark a_1 as disjunctive with a_2. In the example shown in Fig. 4, P_3 is a sequence of the repetitive pattern a_2 (<A>), and P_1 = <STRONG>. So, we identify a_1 (<STRONG>) and a_2 (<A>) as disjunctive patterns. Our proposed algorithm which handles all of these cases and others to align the row r is shown in Fig. 5.

As shown in Fig. 5, to align the row r in M, the algorithm recursively tries to align r and stops at three base cases: first (lines 2-3), when r is an aligned row as defined before, second (lines 4-6), when r only has disjunctive attributes, and third (lines 7-11) when r has a top-horizontal zigzag. In the first case, the algorithm returns a symbol s when r has either a horizontal zigzag L_s or a non-zigzag symbol s. But, if r has leaf nodes (img/text), the algorithm returns "*". In the second case, the algorithm checks if r only has disjunctive attributes by using DisjunctiveAttributes(r, M). The function returns true if both of the following two conditions are satisfied:

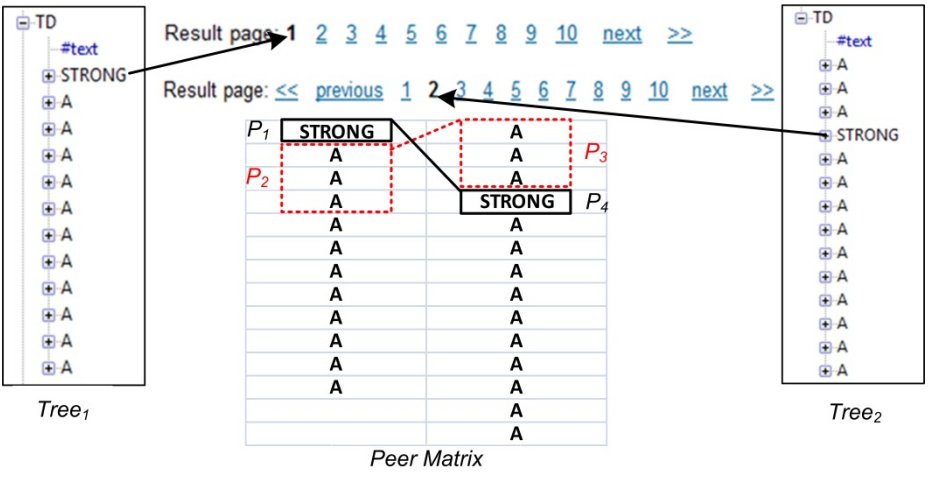

Fig. 4. An example of disjunctive attributes in the matrix constructed from $Tree_1$ and $Tree_2$

```
Algorithm AlignRow(r, M)                    // To align the row r of the peer matrix M.

1.  for each different symbol s_i (i=1, 2, …, c) in the row r, draw a zigzag line L_si;
2.  if (r is an aligned row)                // r is aligned if it has only one horizontal zigzag L_s, r has only one
3.      return s;                           // non-zigzag symbol s, or r has variant leaf nodes (i.e., s="*").
4.  else if (DisjunctiveAttributes(r, M) ))                                   // Case i
5.      s = s_1 || s_2 || …|| s_c ;                    // Disjunctive Attributes
6.      return s;
7.  else if (there is a top horizontal zigzag line L_s)                       // Case ii
8.      shiftNonZigzagSymbols(r, M);                   // Missing attributes
9.      shiftOtherHorizontalZigzags(r, M);             // Missing attributes
10.     stretchZigzag(L_k) for each zigzag L_k (k≠s) in r;   // Missing attributes
11.     return s;
12. else if (there is only ONE non-horizontal regular/irregular zigzag L_s)   // Case iii
13.      stretchZigzag(L_s);                           // Missing attributes
14.      AlignRow(r, M);
15. else                                    // there is a left-most cross.
16.     let m_1, m_2 and P_1, P_2, P_3, and P_4 are defined for L_a and L_b as shown in Fig. 3(a);
17.     if ( either (P_2≠ P_3) or (P_1≠P_4) )          // Missing attributes     Case I
18.         stretchZigzag( selectAZigzag(L_a, L_b, r, M) );
19.         AlignedRow(r, M);
20.     else if (both P_1 and P_3 are non-repetitive)  // Multiple-ordering attributes   Case II
21.         exchange the two patterns P_3 (P_1) and P_4 (P_2) in the column c_2 (c_1), respectively;
22.         AlignRow(r, M);
23.     else if (one pattern (P_1 or P_3) is a seq. of a repetitive pattern a_2, the other equals a_1)   // Case III
24.         replace all occurrences of a_1 by a_2, mark a_1 as disjunctive with a_2;  // Disjunctive patterns
25.         AlignRow(r, M);
26.     else                                           // Missing attributes     Case I
27.         stretchZigzag( selectAZigzag (L_a, L_b, r, M) );
28.         AlignRow(r, M);
29.     endif
30. endif
```

Fig. 5. Our proposed algorithm to align a row r in a peer matrix M

- The row r has a sequence of different symbols s_1, s_2, …, s_k; $k > 1$, where each symbol s_i is either a non-zigzag symbol or belonging to a horizontal zigzag line.
- For each different symbol s_i in r, there is a zigzag line L_s that connects among different occurrences of some symbol s below s_i.

The first condition makes sure the symbols in r have no occurrences below r, while the second condition gives a guarantee that there exists a zigzag which prevents any of these symbols to be shifted down. The third stop case is occurred when the row r has a combination of horizontal zigzag lines (one of them is a top-horizontal), non-zigzag symbols, and other regular/irregular zigzag lines. If so, the algorithm only keeps the top-horizontal zigzag in the row r and considers all others as missing attributes. Therefore, it shifts all non-zigzag symbols (line 8) and all horizontal zigzag lines (line 9) down a distance 1, and stretches all remaining regular/irregular zigzag lines (line 10) using the function stretchZigzag(L_s). The function stretchZigzag works as follows. If L_s is regular (i.e., L_s connects among different occurrences of s in the two rows r and r'; $r<r'$), it shifts all occurrences of s in r downward a distance r'-r in the matrix M and patch empty spaces with a null value. If L_s is irregular (i.e., L_s connects among different occurrences of s in r and other different rows r_1, …, r_k below r), the function shifts each occurrence of s at each row r_i above r´ downward a distance r'-r in the matrix M and patch empty spaces with a null value, where the row r´∈ $\{r_1, \ldots, r_k\}$ satisfies that r'-r is min(r_1-r, …, r_k-r).

When the row has one regular/irregular zigzag line, the algorithm identifies it as missing attribute (lines 12-14) and uses the function stretchZigzag to stretch it. Finally, if the row has a left-most cross (lines 16-29), the algorithm handles it as we discussed above. The function selectAZigzag(L_a, L_b, r, M) returns one of the two zigzag lines L_a or L_b to be stretched based on either of the following three ordered criteria: First, a zigzag that has a line of non-zero minimum vertical span is returned. Second, the one with a maximum number of horizontal lines in the row r of the matrix M is returned. Third, the algorithm returns the right-most one.

For an $n \times m$ matrix M, the running time to draw zigzag lines for each row is $O(n \times m)$. Also, the running time to check whether some pattern is repetitive or not is $O(n \times m)$. As a preprocessing step, for each row, the running time for calculating symbols scan values is $O(n \times m)$. Therefore, the running time for each call of the recursive algorithm to align a row r in the matrix M is $O(n^2 \times m^2)$. Experimentally, a row is aligned after 2-3 calls.

4 Examples

The two matrices in Fig. 6 give two examples of disjunctive attributes. To align the first row of the matrix in Fig. 6(a), the row has one horizontal zigzag L_A and one non-zigzag symbol F, and at the same time there is a zigzag L_C (in the third row) which has occurrences of C below A and F. So, the algorithm identifies A and F as disjunctive attributes (returns s=A||F). To align the first row of the matrix in Fig. 6(b), the row has two horizontal zigzag lines L_A and L_F, and at the same time there is a zigzag L_C which has occurrences of C below both A and F. So, our algorithm also identifies A and F as two disjunctive attributes, and then returns s=A||F.

Fig. 7 gives two examples of a top horizontal zigzag (case ii in Fig. 5). To align the first row of the first matrix (Fig. 7(a)), the row has two horizontal zigzag lines L_A (the top one because it has the maximum number of lines: 2) and L_F. However, there is no zigzag lines that have occurrences below A and F at the same time (i.e., A and F are not identified as disjunctive attributes). So, the algorithm shifts down L_F to the next

row in the matrix. For the matrix in Fig. 7(b), the first row has one horizontal zigzag L_A (the top one) and one regular zigzag L_B. So, the regular zigzag L_B is stretched at the second row (i.e., all occurrences of B in the first row will be shifted down a distance 1 and patch empty spaces with a null value).

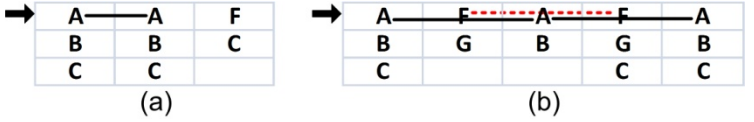

Fig. 6. Two examples of disjunctive attributes, case i

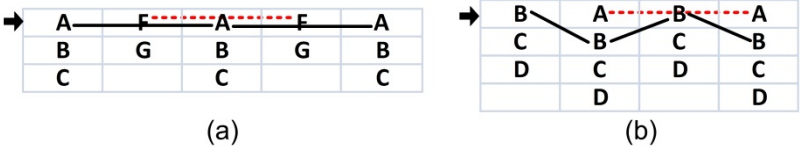

Fig. 7. A row has a top horizontal zigzag, case ii

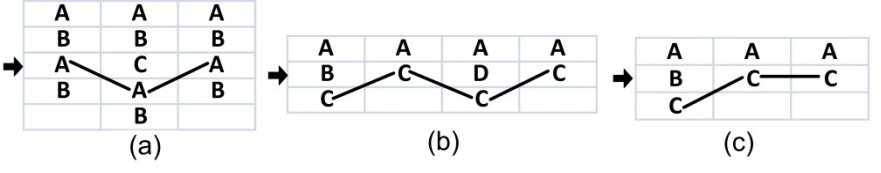

Fig. 8. A row has either one regular zigzag (a) and (b), or one irregular zigzag (c); case iii

Fig. 8 gives three examples when the row has only one non-horizontal regular zigzag (case iii). To align the third (second) row of the matrix in Fig. 8(a) (Fig.8(b)), the algorithm (line 13 in Fig. 5) stretches down L_A (L_C) at the fourth (third) row, respectively. Fig. 8(c) gives an example when the row has only one irregular zigzag (L_C). To align this row, L_C is stretched down (line 13 in Fig. 5) at the third row.

Fig. 9 gives two examples of missing attributes (case I in Fig. 5), when there is a combination of regular and irregular zigzag lines that form one or more crosses. The first step here is to identify the left-most cross and calculate m_1, m_2, and P_1- P_4 as shown in Fig. 3(a). Fig. 9 (a) presents an example of missing attributes when either $P_1 \neq P_4$ or $P_2 \neq P_3$, while Fig. 9 (b) presents another example when both $P_1 = P_4$ and $P_2 = P_3$, either P_1 or P_3 is repetitive, and none of the two patterns P_1 and P_2 is a sequence of some repetitive pattern a_2. In the two examples, the function selectAZigzag in Fig. 5 is used to select one of the two zigzag lines (that make a left-most cross) to be stretched down. For the first matrix in Fig. 9, the function selects L_C to be stretched at the third row because it has a line of the minimum vertical span (2). Also, it selects L_A to be stretched down for the same reason in the second matrix.

Fig. 10 discusses the case of multiple-order attributes, where $P_1 = P_4$="ABC" and $P_2 = P_3$="FGH", but both of the two patterns P_1 and P_3 are non-repetitive. So, to align the first row of the matrix, our algorithm exchanges the two patterns P_2 and P_3 in the third column because they are multiple-ordering patterns. We shall not give here any example of case III, because we already presented one in the previous section.

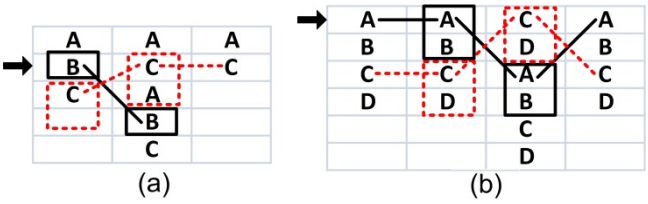

Fig. 9. Two examples of missing attributes, case I

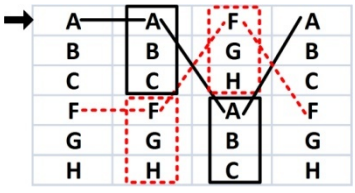

Fig. 10. Multiple-ordering attributes, case II

5 Experiments

We measure the performance of the proposed algorithm by collecting 300 peer matrices taken from a data set of 10 web sites (selected from the manually labeled Testbed for Information Extraction from Deep Web TBDW [8] Version 1.02) as follows. For each web site, we randomly select 30 matrices: 10 matrices from top levels (closed to the root), 10 matrices from the levels closed to the leaves, and 10 matrices from the whole DOM tree. The performance of the algorithm is measured by calculating recall and precision for each selected matrix as follows. Precision is the proportion of symbols predicted by the algorithm as missing, disjunctive, or multiple-ordering that are targets (correctly identified). Recall is the proportion of missing, disjunctive, or multiple-ordering symbols that are predicted by the algorithm. The terms true positives (T_p), true negatives (T_n), false positives (F_p) and false negatives (F_n) compare the predicted class of matrix symbols with the actual class. T_p is the number of missing, disjunctive, or multiple-ordering symbols that are correctly identified by the algorithm. F_n is the number of missing, disjunctive, or multiple-ordering symbols that cannot be identified by the algorithm. F_p is the number of missing, disjunctive, or multiple-ordering symbols that are incorrectly identified by the algorithm. Finally, T_n is the number of symbols that are correctly not identified by the algorithm as missing, disjunctive, or multiple-ordering. Formally, we define recall and precision as follows:

$$Recall = \frac{T_p}{T_p + F_n}; \qquad Precision = \frac{T_p}{T_p + F_p}$$

The performance of the algorithm with the 30 web sites is shown in Table 1. For each web site, the average of the calculated recall and precision for the selected 10 matrices constructed closed to the root (closed to the leaves and from the whole DOM tree) is shown in column 2-3 (4-5 and 6-7, respectively). As shown in the table, the algorithm performs perfectly with matrices near to the root of a DOM tree (columns 2-3), and

gives a good result near to leaves (columns 4-5) as the matrices near to the leaves are complicated and contain many missing, disjunctive, and multi-ordering symbols. The perfect results for the matrices near to the root was not a surprise because many of the matrices are already/easy to be aligned. In general, the results are encouraged for the whole DOM tree (columns 6-7).

Table 1. The performance of the algorithm with a data set of 10 web sites

Site	Closed to root		Closed to leaves		Whole DOM tree	
	Recall	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**
Allnealthnet	1.00	1.00	0.95	0.92	0.99	0.95
G. Unlimited	1.00	1.00	1.00	0.99	1.00	0.97
IUMA	1.00	1.00	0.93	0.92	0.95	0.92
Picsearch	1.00	1.00	1.00	1.00	1.00	1.00
Sun-Sentinel	1.00	1.00	0.98	0.96	1.00	0.97
amazon.co.uk	1.00	1.00	0.91	0.90	0.91	0.91
Amazon	1.00	1.00	0.99	0.95	0.98	0.96
Gene surname	1.00	1.00	0.95	0.95	0.95	0.97
HomePopular	1.00	1.00	0.99	0.97	1.00	0.98
NAMI	1.00	1.00	0.97	0.96	0.98	0.97
Average	**1.00**	**1.00**	**0.97**	**0.95**	**0.98**	**0.96**

6 Related Works

IEPAD [4] and OLERA [9] generalize extraction patterns from unlabeled Web pages. Repetitive patterns in IEPAD are discovered using the binary suffix tree PAT tree. PAT trees compute only exact match patterns, templates with exceptions cannot be discovered through PAT trees. Patterns with inexact or approximate matching are discovered using multiple string alignment technique. IEPAD applies center star algorithm to align multiple strings. In OLERA [9], user marks a record to be extracted to discover other similar records and generalize them using multiple string alignment. OLERA handles the problem in IEPAD when several alignments exist by proposing a matching function to compare the primitive data for text tokens.

RoadRunner [1] considers the site generation process as encoding of the original database content into strings of HTML code. The system uses the ACME matching (alignment) technique to compare HTML pages of the same class and generate a wrapper based on their similarities and differences.

DEPTA [6] discovers repetitive patterns by comparing adjacent substrings with starting tags having the same parent in the HTML tag tree. The recognition of data items or attributes in a record is accomplished by partial tree alignment. The algorithm first chooses the record tree with the largest number of data items as center and then matches other record trees to the center tree. ViPER [10] assumes that repetitive patterns have variant lengths rather than they are of fixed length as in Depta. ViPER applies a tandem repeats algorithm before computing the edit-distance to handle missing and multiple values data. It applies a data alignment technique that is based on

global matching and text content information. The alignment method uses a divide-and-conquer fashion to reduce the multiple-alignment problem.

Finally, FiVaTech [3] conducts four steps: peer node recognition, matrix alignment, pattern mining, and optional node detection in turn. In the matrix alignment step, the system handles the two problems of disjunctive and multiple-ordering attributes as a case of missing attributes.

7 Conclusions

In this paper, we proposed a new algorithm for multiple string (peer matrix) alignment. Our algorithm looked at all of the multiple strings at the same time, so it has a global view for the inputted strings. Also, our algorithm considered the common problems in the field of web data extraction: missing, multi-valued, multi-order, and disjunctive attributes. To align a row in the peer matrix, our algorithm drew some virtual zigzag lines for each symbol in the row, and then tried to stretch/shift some lines to accomplish the task.

References

1. Crescenzi, V., Mecca, G., Merialdo, P.: Knowledge and Data Engineerings. In: Proc. Int'l Conf. Very Large Databases (VLDB), pp. 109–118 (2001)
2. Arasu, A., Garcia-Molina, H.: Extracting Structured Data from Web Pages. In: Proc. ACM SIGMOD, pp. 337–348 (2003)
3. Kayed, M., Chang, C.-H.: Page-level web data extraction from template pages. IEEE Trans. Know. and Data Eng. 22(2), 249–263 (2010)
4. Chang, C.-H., Lui, S.-C.: IEPAD: Information Extraction Based on Pattern Discovery. In: Proc. Int'l Conf. World Wide Web (WWW-10), pp. 223–231 (2001)
5. Wang, J., Lochovsky, F.H.: Data Extraction and Label Assignment for Web Databases. In: Proc. Int'l Conf. World Wide Web (WWW-12), pp. 187–196 (2003)
6. Zhai, Y., Liu, B.: Web Data Extraction Based on Partial Tree Alignment. In: Proc. Int'l Conf. World Wide Web (WWW-14), pp. 76–85 (2005)
7. Chang, C.-H., Kayed, M., Girgis, M., Shaalan, K.: Survey of Web Information Extraction Systems. IEEE Trans. Know. and Data Eng. 18(10), 1411–1428 (2006)
8. Yamada, Y., Craswell, N., Nakatoh, T., Hirokawa, S.: Testbed for Information Extraction from Deep Web. In: Proc. WWW-13, pp. 346–347 (2004)
9. Chang, C.-H., Kuo, S.-C.: OLERA: A semi-supervised approach for Web data extraction with visual support. IEEE Intelligent Systems 19(6), 56–64 (2004)
10. Simon, K., Lausen, G.: ViPER: Augmenting Automatic Information Extraction with Visual Perceptions. In: Proc. CIKM (2005)