AFIS: Annotation-Free Induction of Full Schema for Detail-Pages

by Tesis MM

Submission date: 11-Nov-2023 05:08PM (UTC+0700)
Submission ID: 2224669495
File name: AFIS.pdf (1.3M)
Word count: 8397
Character count: 38219

Data Min Knowl Disc manuscript No.
(will be inserted by the editor)

AFIS: Annotation-Free Induction of Full Schema for Detail-Pages

Oviliani Yenty Yuliana · Chia-Hui Chang

Received: date / Accepted: date

Abstract Web data extraction is an essential task for web data integration. Most researches focus on data extraction from list-pages by detecting data-rich section and record boundary segmentation. The problem of data alignment in records is small scale since only a couple data attributes need to be aligned. However, for detail-pages which contain all-inclusive product information in each page, the number of data attributes need to be aligned is much larger. In this paper, we formulate the data extraction problem as alignment of leaf nodes from DOM Trees. We propose AFIS, an Annotation-Free Induction of full Schema for detail-pages. AFIS applies Divide-and-Conquer and Longest Increasing Sequence (LIS) algorithms to mine landmarks from input. The experiments show that AFIS outperforms Road-Runner, FivaTech and TEX (with precision **0.994**, recall **0.987**, and F1 **0.990**) in terms of selected (data) columns. For full schema evaluation (all data columns), AFIS also represents the highest average performance (with precision **0.946**, recall **0.930**, and F1 **0.937**) compared with TEX and RoadRunner.

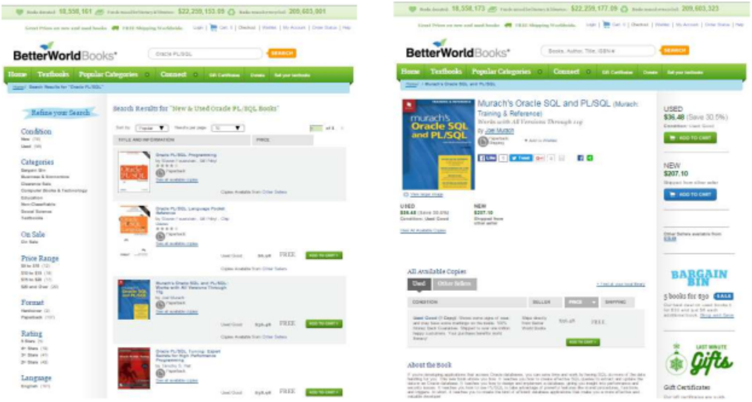
Keywords Web data extraction · Semi-structured data · Detail-pages alignment · Divide-conquer alignment · Landmark equivalence class

1 Introduction

Currently, a lot of web pages are generated dynamically from predefined template upon users' query, which we called deep web. Deep webs ware estimated 400 to 550 times larger than surface webs (Bergman 2001). Extraction of embedded data from deep web requires substantial efforts because these web pages are not generated for data exchange. Therefore, generating a tool for extracting data automatically for information integration is an essential task.

Oviliani Yenty Yuliana · Chia-Hui Chang
Department of Computer Science and Information Engineering
National Central University, Taoyuan 32001, Taiwan
E-mail: oviliani@gmail.com

Chia-Hui Chang
E-mail: chia@csie.ncu.edu.tw



(a) list-page (b) detail-page

Fig. 1: Examples of list-page and detail-page

In general, there are two kinds of template web pages. The first type is list-pages that contain a list of records in a web page. For instance, a list-page with book records is shown in Fig. 1(a). Another type is detail-pages which contain various kinds of information for a pariticular item or product in a page. For example, a detail-page containing the information of a book is shown in Fig. 1(b). Most researches (e.g. WebSets (Dalvi et al. 2012), CTVS (Su et al. 2012), etc.) focus on search result record extraction, which requires data-rich section detection, record boundary segmentation, and data attribute alignment. Since the performance is evaluated on selected data items (usually the data-rich section), not much effort has been put on the extraction of side information, especially for detail-pages. While some researches (e.g. FivaTech (Kayed and Chang 2010), TEX (Sleiman and Corchuelo 2013), etc.) induce full schema for the complete pages, the performance on detail-pages still has room for improvement.

The difficulties of aligning detail-pages come from several aspects: First, the number of data attributes need to be aligned is much larger than that of data records in list-pages. Second, there are more data types and more optional data to be processed. Third, the leaf nodes with the same content could have different paths because of decorative tags. On the other hand, the leaf nodes with the same content and equal path might function differently. In addition, it is more likelily to have multiple attribute-value pairs which leads to multi-order data rendering.

In this paper, we propose an Annotation-Free Induction of full Schema (abbreviated as AFIS) for detail web pages. The proposed technique operates efficiently on leaf nodes of DOM trees of input pages by recording all of the required information to identify templates and detect data. We implement several algorithms to achieve our goal. First, AFIS uses Divide-and-Conquer and Longest Increasing Sequence (LIS) algorithms for our novel template mining via Landmark Equivalence Class (LEC). Furthermore, AFIS uses *LECTable* for aligning leaf nodes into attribute-value pairs. Finally, AFIS rearranges the leaf nodes to achieve the challenging alignment task. Compared with TEX (Sleiman and Corchuelo 2013) which has very high performance for the selected data items in the form of tagged-list, the average performance of AFIS outperform the result with 0.994 F1-measure. Moreover, we evaluate the full schema inducted by AFIS, TEX, and RoadRunner with manaually annotated golden answer. AFIS outperforms other techniques with 0.937 F1-measures.

The rest of the paper is organized as follows. In the next section, we compare AFIS with related Web data extraction techniques. In Section 3, we provide our information extraction model as well as the detailed solution of AFIS with example. The performance evaluations are presented and analysed in Section 4. Finally, we conclude our current research and mention our future work in Section 5.

2 Related work

Web information extraction has been a hot topic for a decade. A number of information extraction approaches have been proposed with diverse degree of automation, i.e. supervised, semi supervised, and unsupervised. SoftMealy (Hsu and Chang 1999) and WIEN (Kushmerick 1997) are example of supervised learning. Currently, most researches focus on unsupervised approaches to reduce manual efforts for improving effectiveness of information extraction. Some representatives of unsupervised approaches are EXALG (Arasu and Garcia 2003), Zhao et al. (2007), FiVaTech (Kayed and Chang 2010), Hao et al. (2011), CETD (Sun et al. 2011), WebSets (Dalvi et al. 2012), CTVS (Su et al. 2012), Zheng et al. (2012), Uzun et al. (2013), Lu et al. (2013), and TEX (Sleiman and Corchuelo 2013). This paper also applies unsupervised approach to extract information. Although all the researches are unsupervised, they targeted on different input and extraction output. Meanwhile, they applied different granularity of processing unit for template mining and schema induction as described below.

First of all, depending on input pages, the extraction target may be the search result records from list-pages or the detail data items for the product specified in a page. Meanwhile, considering the efficiency and the number of input pages for training, various researches also consider different processing units. For example, CTVS (Su et al. 2012), Lu et al. (2013), FiVaTech (Kayed and Chang 2010), and CETR (Weninger et al. 2010) operate on DOM trees. IEPAD (Chang and Lui 2001) and RoadRunner (Crescenzi and Mecca 2004) consider only HTML tag tokens and a special TEXT token to denote script blocks, style blocks or #PCDATA. EXALG (Arasu and Garcia 2003) and TEX (Sleiman and Corchuelo 2013) use not only HTML tag tokens but also word tokens as their processing unit.

Furthermore, extraction rules can be induced by either top-down or bottom-up approaches (Chang et al. 2006; Sarawagi 2008). Top-down learning algorithm induces from general to specific concept in various ways to get set of rules with high precision. For instance, FiVaTech (Kayed and Chang 2010) traverses from root to leaf nodes for finding template and detecting data. On the other hand, bottom-up learning algorithm starts inducing with the most specific instances and then replaces them progressively with more general rules. For example EXALG (Arasu and Garcia 2003) and TEX (Sleiman and Corchuelo 2013) induce schemas from common word and HTML tag tokens based on bottom-up learning algorithm.

In this paper, we are particularly interested in algorithms that are designed for full schema induction such as RoadRunner and TEX.

- RoadRunner (Crescenzi and Mecca 2004) learns a union-free regular expression by generating a base template from the first web page then it compares literately with another web page using a string alignment algorithm. Meanwhile, RoadRunner applies a backtracking algorithm for detecting optional and repetitive patterns.
- EXALG (Arasu and Garcia 2003) proposed equivalence classes among a string of tokens to find out a template from the given web pages. Afterwards the discovered template is used by EXALG for extracting data.
- FiVaTech (Kayed and Chang 2010) identifies a tree template and detects a data schema from DOM tree automatically. For that purpose, it applies a clustering algorithm using tree-edit distance for aligning the sibling nodes at the same level. FiVaTech also employs a mining technique to mine repetitive patterns and several heuristics to detect optional information.
- TEX (Sleiman and Corchuelo 2013) finds and discards the shared longest sequence tokens (Texts) amongst web documents (TextSet) until finding the relevant information that should be extracted from them. In other words, TEX extracts varies information from web documents and it removes information that belong to the template.

3 System architecture

A system architecture for our proposed approach AFIS can be seen in Fig. 2. It consists of three modules, i.e. data preprocessing, divide-and-conquer alignment, and wrapper generation. The first module is data preprocessing which parses all given web pages into DOM trees and arranges all leaf nodes of DOM tree into a table for alignment. The second module is divide-and-conquer alignment which is the core of this paper for mining template and splitting attribute-and-value pairs. We leave the last module in our future work.

3.1 Data preprocessing

First, the given web pages must be parsed into DOM trees. For scalability consideration, we use leaf nodes of the DOM trees as our basic processing units since the average number of leaf nodes (798) is much smaller than the average number of tag content tokens (4,548, consist of 3,756 tag tokens and 792 content tokens) for pages with average size 77KB. For each leaf node l, we maintain the following properties including *LeafIndex*, *Path*, *Content*, *IDSet*, and *ClassSet*.

- *LeafIndex* is a unique number of the ordered leaf nodes in a document.
- *Path* is the sequence of tag names from a root node to a leaf node l, $Path = path[1]/path[2]\cdots/path[|l.path|]$. Note that if the last tag name is a decorative tag, $DecorativeTag \equiv \{a, b, big, cite, dfn, font, em, i, mark, small, span, sub, sup, strike, strong, u\}$, we will remove it from the path to the text content.
- *Content* refers to the text content of a leaf node.

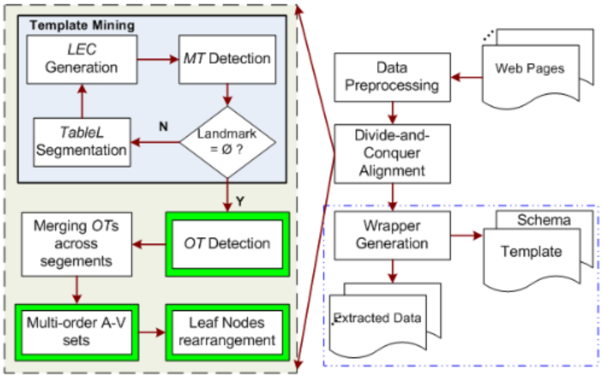


Fig. 2: System architecture: blue background denotes recursive call while green background denotes iterative process for each segment

- *IDSet* is the union of *id* attributes from all tags in the *Path*, see Eq. (1), where $ID(tag)$ is an *id* attribute from a given tag.

$$IDSet = \bigcup_{i=1}^{|l.Path|} ID(l.path[i]) \qquad (1)$$

- *ClassSet* is the union of *class* attributes from all tags in the *Path*, see Eq. (2), where $Class(tag)$ is an *class* attribute from a given tag.

$$ClassSet = \bigcup_{i=1}^{|l.Path|} Class(l.path[i]) \qquad (2)$$

- *TypeSet* is the union of all encoded tokens of *Content*, see Eq. (3), where $type(l.content[i])$ is a function to encoded a given token and i is an token index in $l.content$.

$$TypeSet = \bigcup_{i=1}^{|l.Content|} type(l.content[i]) \qquad (3)$$

  The procedure to tokenize $l.Content$ and obtain the *TypeSet* is described as follows.
  1. Split $l.Content$ based on spaces into tokens, $l.Content = \langle l.content[i] \rangle$, where $i = 1 \cdots |l.content|$.
  2. If a token $l.content[i]$ has prefix or suffix of punctuation marks, remove the marks and union type "13" to $l.TypeSet$.
  3. Encode $l.content[i]$ based on token type as shown in Table 1 via regular expressions.
  4. Union all encoded tokens into the *TypeSet*, i.e. Eq. (3).

  As shown in Example 1, we obtain a set of nine token types for the given text content with 9 tokens.

Table 1: Token type encoding

name	type
mixed characters	1
all capital letters	2
small letters	3
first capital letters	4
percentage	5
date	6
time	7
url	8
email	9
currency	10
decimal	11
integer	12
punctuation	13

Example 1 Given a $l.Content$ = "eBay item 1330190403 (02-09-2013 17:09:30 PST) Great N275 \$16.00". The tokens and the related types are listed as follows.

- $l.content[1]$ = "$eBay$" → $l.typeset[1] = \{1\}$
- $l.content[2]$ = "$item$" → $l.typeset[2] = \{3\}$
- $l.content[3]$ = "1330190403" → $l.typeset[3] = \{12\}$
- $l.content[4]$ = "(02-09-2013" → $l.typeset[4] = \{6, 13\}$
- $l.content[5]$ = "17:09:30" → $l.typeset[5] = \{7\}$
- $l.content[6]$ = "$PST)$" → $l.typeset[6] = \{2, 13\}$
- $l.content[7]$ = "$Great$" → $l.typeset[7] = \{4\}$
- $l.content[8]$ = "$N275$" → $l.typeset[8] = \{1\}$
- $l.content[9]$ = "\$16.00" → $l.typeset[9] = \{10\}$

$\therefore l.TypeSet = \{1, 2, 3, 4, 6, 7, 10, 12, 13\}$

Definition 1 **(*TableL*)** Let D be the set of all DOM trees, $D = d[1], d[2], \cdots, d[m]$. For each DOM tree $d[j]$, we number the leaf nodes from 1 to $|d[j]|$ and refer to each leaf node by $d[j][i]$ where i is the leaf node index ($LeafIndex$) in $d[j]$, $1 \leq i \leq |d[j]|$. Therefore, the goal of the preprocessing module is to generate a table of all leaf nodes for the next step. We denote the table by $TableL = \bigcup_{j=1}^{|D|} \bigcup_{i=1}^{|d[j]|} d[j][i]$, where j is a document index in D.

In summary, the preprocessing module generate $TableL$ for a given set of web pages by the following five steps.

1. Parse an input page using **CyberNeko** (Clark and Guillemot 2013) into **DOM Tree** $d[j]$.
2. Characterize each leaf node by $LeafIndex$, $Path$, $Content$, $IDSet$ (see Eq. (1)), and $ClassSet$ (see Eq. (2)) features.
3. Encode a given $l.Content$ into a $l.TypeSet$ as described above.
4. Encode a given $l.Path$ if there exist $DecorativeTag$ or $\langle br \rangle$ as follows.
   - Remove $\langle span \rangle$ and $\langle font \rangle$ tags from $l.Path$.
   - Remove $\langle br \rangle$ and $DecorativeTag$ tags from $l.Path$ into $l.Content$.
5. Arrange all leaf nodes in DOM trees into $TableL$.

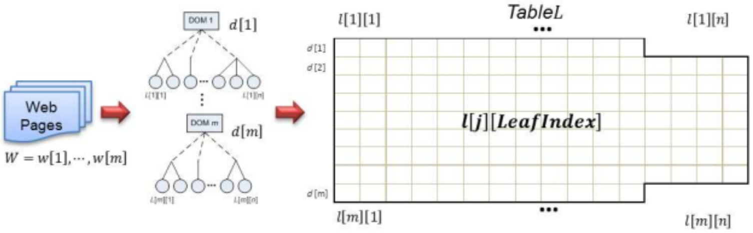


Fig. 3: Data preprocessing framework: j: document index, i: leafnode index.

3.2 Divide-and-conquer alignment

Divide-and-conquer alignment is the core of our work in this paper. The major steps include landmark detection, mine template recurrently, generate extractor, and rearrange leaf nodes in $TableL$.

3.2.1 Landmark equivalence class

Similar to EXALG (Arasu and Garcia 2003) and TEX, we define equivalence class to discover landmarks for the division procedure. In this paper, we use *Path* and *Content* of leaf nodes for generating *Landmark Equivalence Class* (LEC). We then use an $LECTable$ to mine base template for web page division. An illustration of generating LEC from $TableL$ can be shown in Fig. 4(a)(b) and Example 2.

Definition 2 (Equivalence Leaf Nodes) Two leaf nodes u and v are considered as equivalence if $u.Path \cong v.Path$, i.e. $Similarity(u.Path, v.Path)$ by Eq. (4) is greater than θ_{path}, and $u.Content = v.Content$. Note that if a content contains decorative tags, we consider <b>≊<strong> and <i>≊<em> and

$$StrSimilarity(s_1, s_2) = \frac{LCS(s_1, s_2)}{max(|s_1|, |s_2|)} \tag{4}$$

where LCS is the longest common subsequence and θ_{path} is a given $Path$ threshold.

Definition 3 (Landmark Equivalence Class) A set of equivalence leaf nodes form an equivalence class. For each LEC, we assign an $LECId$ and keep the $Path$, $Content$. In addition, we also calculate the *Occurrence Vector* (see Definition 4) and *First Position* (see Definition 5) for each LEC. As a note, $Index$ is a sequence number and $LECId$ is a sequence number of uniqueness compound of a $Path$ and a $Content$.

$$LEC_e(Path_e, TC_e) = \{\, d[j][i] \mid d[j][i].Path \cong Path_e, d[j][i].Content = TC_e \,\} \tag{5}$$

where $j = 1 \cdots |D|, i = 1 \cdots |d[j]|$.

Definition 4 (Occurrence Vector) An Occurrence Vector (OV) for an LEC_e is a vector of occurrence count O_j of LEC_e in each $d[j]$, i.e. $OV_e = [o_1, o_2, \cdots, o_m]$.

Definition 5 (First Position) A First Position (FP) of a Landmark Equivalence Class(LEC_e) is a vector of the first occurrence position P_j (or **-1** if missing in $d[j]$) of the LEC_e in each $d[j]$, i.e. $FP_e = [p_1, p_2, \cdots, p_m]$, where p_j is the smallest i for all $d[j][i]$ in LEC_e.

Example 2 For the given $TableL$ in Fig. 4(a), the first leaf nodes in all five DOM trees are quivalent to each other. Thus, they form an equivalence class with $LECId$=1. Similarly, leaf nodes with $Content$= “$\langle b \rangle$Job ID:$\langle /b \rangle$” and $Path$= “html/body/table/tr/td/table/tr/td/text” form equivalence class with $LECId$=2. Both LEC_1 and LEC_2 has the same occurrence vector $OV_1 = OV_2 = [1, 1, 1, 1, 1]$ and $FP_1 = [1, 1, 1, 1, 1]$, $FP_2 = [2, 2, 2, 2, 2]$. Also note $LECId$=14 which are comprised of leaf nodes with $Content$ “$\langle b \rangle$Location(s):$\langle /b \rangle$” and $Path$ “html/body/

table/tr/td/table/tr/td/text", and has occurrence vector $OV = [1, 1, 0, 1, 1]$ even if they occur in different positions in four documents $FP = [6, 12, -1, 6, 12]$.

Consider the third leaf nodes in each document, which have different text content, i.e 8490290, 8497839, ... They will become singular equivalence classes LEC_3 to LEC_7 with similar occurrence count $OV = [1, 0, 0, 0, 0]$ to $OV = [0, 0, 0, 0, 1]$ and $FP = [3, -1, -1, -1, -1]$ to $FP = [-1, -1, -1, -1, 3]$.

A special case is for leaf nodes without text content like $\langle br/\rangle$ tags, since we have moved these tags into *Content*, we can define equivalence class in a similar way. For instance, there are two leaf nodes with *Content* $\langle br/\rangle$ in $d[3], d[4], d[5]$, therefore, the system assign $LECId$=31 with $OV_{31} = [1, 1, 2, 2, 2]$ and $FP_{31} = [14, 14, 12, 14, 14]$.

Let $maxcol$ be the maximum number of leaf nodes from D, i.e. $\max_j |d[j]|$. To construct LEC table from $TableL$, the system applies a single link clustering to leaf nodes in $TableL$ column-wise. Initially, $d[1][1]$ forms an LEC with $LECId = 1$, $OV = [1, 0, \cdots, 0]$, $FP = [1, -1, \cdots, -1]$. For each following $d[j][i]$ in $TableL$, where $i = 1 \cdots maxcol$, $maxcol = max(d[j])$, and $j = 1 \cdots |D|$, the system compares it with existing LECs.

1. If there exists an $LEC_f = (Path_f, TC_f)$ such that $d[j][i]$ is equivalence to LEC_f, i.e. $d[j][i].Content = TC_f$ and $d[j][i].Path \cong Path_f$, we increment $OV_f[j]$ and update $FP_f[j]$ if $d[j][i]$ is the first leaf node in $d[j]$ equivalence to LEC_f, i.e. $OV_f = [o_1, \cdots, o_j{+}{+}, \cdots, o_m]$ and assign $FP_f[j]$ with i if $FP_f[j] = -1$.
2. If $d[j][i]$ is not equivalence to any existing LEC, the system adds a new LEC_e with $LECId = e$, $FP_e[j] = i, OV_e[j] = 1$, and $FP_e[k] = -1, OV_e[k] = 0$ for $k \neq j$.

Finally, the system orders the generated LECs by the minimum of nonnegative first positions, i.e. $min_i\{FP[i]|FP[i] \geq 0, 1 \leq i \leq |D|\}$. If the minimum of nonnegative first positions is the same, we then sort LEC by $LECId$. After reordering, each LEC also has an index in the $LECTable$ in addition to $LECId$. In the following, we use $LEC[i]$ to refer the i-th LEC in the $LECTable$ and LEC_e to refer an LEC with $LECId = e$.

3.2.2 Mandatory template detection

The purpose of mandatory template detection (MTD) is to identify landmarks for web page segmentation and divide-and-conquer. A mandatory template MT is defined as follows:

Definition 6 (Mandatory Template) A Mandatory Template (MT) is an LEC with the same occurrence count K in each $d[j]$, i.e. $LEC_e.OV[j] = K$ where $K \in N, j = 1 \cdots |D|$.

After the sorting of LECs during $LECTable$ construction, it is possible that the first positions of LECs in $d[j]$ are not in order. Thus, we apply LIS (Longest Increasing Subsequence Fredman (1975)) to select as more MTs by maintaining an increasing sequence of first positions for the selected MTs. For each $d[j]$, find the longest increasing subsequence from the sequence of first positions $LEC[k].FP[j]$ for each $LEC[k]$ $(k = 1, \cdots |LECTable|)$, i.e. $S = \{LEC[1].FP[j], LEC[2].FP[j],$

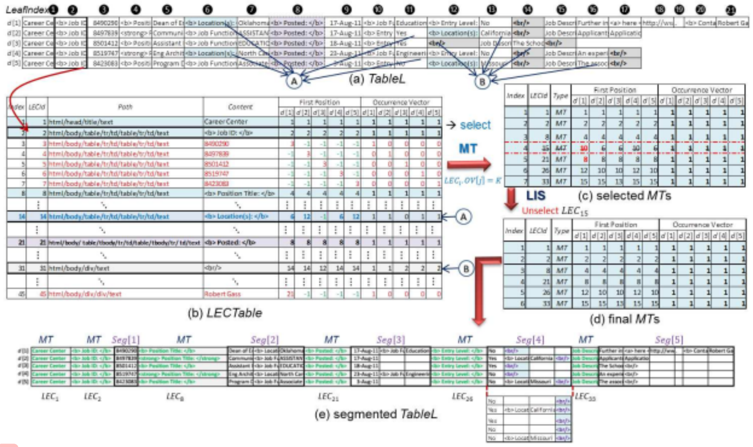


Fig. 4: An example of *LEC* generation, *MT* selection, and *TableL* segmentation

$\cdots, LEC[k].FP[j]\}$, such that for every $k < k^{'}$, $LEC[k].FP[j] < LEC[k^{'}].FP[j]$. By applying the LIS algorithm shown below, the system selects important MTs for further analysis.

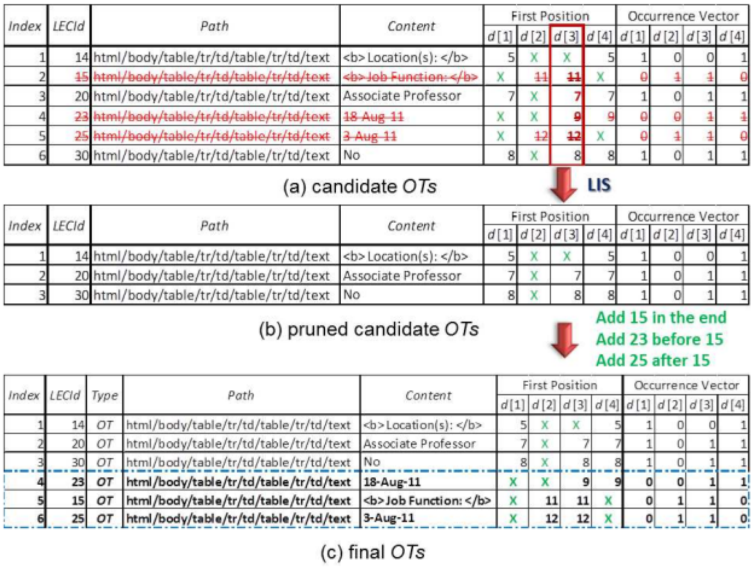
1. Clone the sequence S into $S^{'}$.
2. Sort $S^{'}$ in ascending order.
3. Find the Longest Increasing Subsequence by $LCS(S, S^{'})$.

An illustration of LEC selection for constructing MT and $TableL$ segmentation can be seen in Fig. 4(b)(c) and Example 3. A MT mining is done as follows.

1. Prune $LEC[k]$ if $LEC[k]$ is not an MT.
2. Apply LIS on the first positions of MTs for each $d[j]$ $(j = 1 \cdots |D|)$ to select as many MTs as possible.
3. Use the selected MTs for segmenting $TableL$ into new $TableL$s.
4. Recursively call MT Detection for each new $TableL$.

Example 3 As shown in Fig. 4(b)(c), the system first selects MT from $LECTable$ and reduces the number of LECs from 45 into 7. Next, the system applies LIS on the first positions of MT in each document. Since the first positions of these selected MT in $d[1]$, i.e. {1,2,4,10,8,12,15}, are not order, LEC_{15} (with FP[1]=10) is removed from selected MTs as shown in Fig. 4(d). These selected MT are then used for segmenting $TableL$ for following optional template detection.

For each segment, we re-evaluate the FP and OV for each LEC since an LEC can occur in multiple segments on a web page. For example, leaf nodes corresponding to LEC_{14} (i.e. <b>Location(s): </b>) occur both at Seg[2] and Seg[4] in Fig. 4(e) because of the MT LEC_{21} and LEC_{26}. The new FP for LEC_{14} in $seg[2]$ and $seg[4]$ will be [6, -1, -1, 6, -1] and [-1, 12, -1, -1, 12] respectively as shown in Fig. 6(a).



Index	LECId	Path	Content	First Position				Occurrence Vector			
				d[1]	d[2]	d[3]	d[4]	d[1]	d[2]	d[3]	d[4]
1	14	html/body/table/tr/td/table/tr/td/text	<b> Location(s): </b>	5	X	X	5	1	0	0	1
2	~~15~~	~~html/body/table/tr/td/table/tr/td/text~~	~~<b> Job Function: </b>~~	X	~~11~~	~~11~~	X	~~0~~	~~1~~	~~1~~	~~0~~
3	20	html/body/table/tr/td/table/tr/td/text	Associate Professor	7	X	7	7	1	0	1	1
4	~~23~~	~~html/body/table/tr/td/table/tr/td/text~~	~~18-Aug-11~~	X	X	~~9~~	~~9~~	~~0~~	~~0~~	~~1~~	~~1~~
5	~~25~~	~~html/body/table/tr/td/table/tr/td/text~~	~~3-Aug-11~~	X	~~12~~	~~12~~	X	~~0~~	~~1~~	~~1~~	~~0~~
6	30	html/body/table/tr/td/table/tr/td/text	No	8	X	8	8	1	0	1	1

(a) candidate *OTs*

Index	LECId	Path	Content	First Position				Occurrence Vector			
				d[1]	d[2]	d[3]	d[4]	d[1]	d[2]	d[3]	d[4]
1	14	html/body/table/tr/td/table/tr/td/text	<b> Location(s): </b>	5	X	X	5	1	0	0	1
2	20	html/body/table/tr/td/table/tr/td/text	Associate Professor	7	X	7	7	1	0	1	1
3	30	html/body/table/tr/td/table/tr/td/text	No	8	X	8	8	1	0	1	1

(b) pruned candidate *OTs*

Index	LECId	Type	Path	Content	First Position				Occurrence Vector			
					d[1]	d[2]	d[3]	d[4]	d[1]	d[2]	d[3]	d[4]
1	14	OT	html/body/table/tr/td/table/tr/td/text	<b> Location(s): </b>	5	X	X	5	1	0	0	1
2	20	OT	html/body/table/tr/td/table/tr/td/text	Associate Professor	7	X	7	7	1	0	1	1
3	30	OT	html/body/table/tr/td/table/tr/td/text	No	8	X	8	8	1	0	1	1
4	**23**	**OT**	**html/body/table/tr/td/table/tr/td/text**	**18-Aug-11**	X	X	**9**	**9**	**0**	**0**	**1**	**1**
5	**15**	**OT**	**html/body/table/tr/td/table/tr/td/text**	**<b> Job Function: </b>**	X	**11**	**11**	X	**0**	**1**	**1**	**0**
6	**25**	**OT**	**html/body/table/tr/td/table/tr/td/text**	**3-Aug-11**	X	**12**	**12**	X	**0**	**1**	**1**	**0**

(c) final *OTs*

Fig. 5: An illustration of selecting and adding OTs

3.2.3 Optional template detection

Once the FP and OV are re-evaluated, the system then selects for each segment as many OTs as possible while keeping $FP[j]$ of the OTs consistent in each $d[j]$.

Definition 7 (Optional Template) An Optional Template (OT) is an LEC with the same occurrence count K in each $d[j]$ or null (0) otherwise, and the ratio of non-null documents is greater than a threshold θ_{OT}.

$$Support(LEC_e) = \frac{\sum_{j=1}^{|D|} I(LEC_e.OV[j])}{|D|}, \text{ and } I(x) = \begin{cases} 1 \text{ , } x > 0 \\ 0 \text{ , } otherwise \end{cases}$$

The detail steps of optional template detection are similar to that of mandatory template detection. However, since there are null occurrence (i.e. with first position equals to -1) for some OTs, we ignore documents with null occurrence and focused on comparable pages. In addition, the OTs that are not selected by LIS will be added back to the $OTTable$ as shown below.

1. Prune LEC_e if $Support(LEC_e) < \theta_{OT}$
2. Apply LIS for selecting OTs in each segment as defined in MT detection.
3. For each OT_r that is not selected by LIS, find a position p to add back the removed OT_r. In other words, find an $LEC[p]$ such that $LEC[p].FP[j] < LEC_r.FP[j]$ and $LEC[p+1].FP[j] > LEC_r.FP[j]$ for each comparable page $d[j]$.

Example 4 For the optional templates in Fig. 5(a)[1], LIS selects three OTs with $LECId = 14, 20, 30$ and removes OTs with $LECId = 15, 23, 25$ as shown in Fig. 5(b). The removed OT_{15} is inserted after OT_{30} since $OT_{30}.FP[3] < OT_{15}.FP[3]$, while OT_{23} is inserted between OT_{30} and OT_{15} since $OT_{30}.FP[3] < OT_{23}.FP[3] < OT_{15}.FP[3]$. Finally, OT_{25} is inserted after OT_{15} since $OT_{15}.FP[3] < OT_{25}.FP[3]$ are shown in Fig. 5(c).

In summary, optional template detection first selects OTs via LIS and then reorders the remaining OTs based on the information from comparable pages.

3.2.4 Merging OTs across segments

After optional template detection in each segment, the system concatenates the *final OTs* generated from each segment with the separating MTs for schema induction. As an LEC can occur in multiple segments, it may be detected as an OT in multiple segments as well. Some of such OTs that are generated from the pruned MTs. Therefore, the system tries to merge such recurring OTs by removing MTs that seperates the recurring OTs in order to generate a common schema.

Definition 8 (Recurring OT) A recurring OT is an LEC which occurs in two segments separated by some MT and has complement occurrence vector, i.e. there exists i and i' $(i < i')$ such that $LEC[i].LECId = LEC[i'].LECId$ and $LEC[i].OV \barwedge LEC[i'].OV = true$, where $\barwedge$ denotes *NOT AND* operator[2].

If a recurring OT is detected, the system will merge $LEC[i]$ and $LEC[i']$ and remove the MTs between them by changing the type into OT.

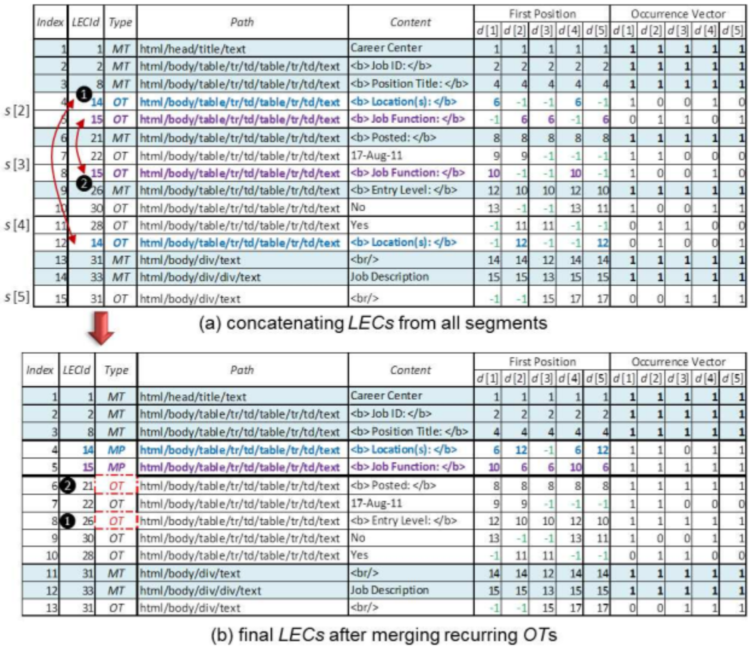
1. Change $LEC[c].Type$ to OT for $i < c < i^{'}$
2. Change $LEC[i].Type$ to MP
3. Update the first position of $LEC[i]$ by $\max\{LEC[i].FP, LEC[i'].FP\}$ and the occurrence vector of $LEC[i]$ by $LEC[i].OV + LEC[i'].OV$
4. Remove $LEC[i^{'}]$
5. Apply LIS to ensure $LEC[i].FP$ in order for $LEC[i].Type = MP$ and $support(LEC[i] < 1)$, otherwise remove the $LEC[i].Type = MP$.

Example 5 Consider LEC_{15} in Fig. 6(a), which is a recurring OT since it occurs in $LEC[5]$ and $LEC[8]$ and $LEC[5].OV \barwedge LEC[8].OV = true$. Thus, the system merges $LEC[5]$ with $LEC[8]$ and changes the type of $LEC[6].Type$ from MT to OT. The first positions and occurrence counts of $LEC[5]$ are then updated accordingly, i.e. $LEC[5].FP = [10, 6, 6, 10, 6]$, $LEC[5].OV = [1, 1, 1, 1, 1]$. Similarly, LEC_{14} in Fig. 6(a) is also a recurring OT since it occurs in $LEC[4]$ and $LEC[12]$ and $LEC[4].OV \barwedge LEC[12].OV = true$. Therefore, the system merges $LEC[4]$ with $LEC[12]$ and changes the type of $LEC[6].Type$ and $LEC[9].Type$ from MT to OT. The first positions and occurrence counts of $LEC[4]$ are then updated accordingly, i.e. $LEC[4].FP = [6, 12, -1, 6, 12]$, $LEC[4].OV = [1, 1, 0, 1, 1]$. Finally, $LEC[5]$ and $LEC[4]$ are assigned with type MP as shown in Fig. 6(b).

After merging every recurring OTs, the system adds two sentinel LECs at the start and the end of the template LECTable as shown in Fig. 7(b).

[1] Note that this example is independent of Fig. 4.

[2] Or equivalently $LEC[i].OV + LEC[i'].OV = \mathbf{1}$.



	Index	LECId	Type	Path	Content	First Position					Occurrence Vector				
						d[1]	d[2]	d[3]	d[4]	d[5]	d[1]	d[2]	d[3]	d[4]	d[5]
	1	1	MT	html/head/title/text	Career Center	1	1	1	1	1	1	1	1	1	1
	2	2	MT	html/body/table/tr/td/table/tr/td/text	<b> Job ID: </b>	2	2	2	2	2	1	1	1	1	1
	3	8	MT	html/body/table/tr/td/table/tr/td/text	<b> Position Title: </b>	4	4	4	4	4	1	1	1	1	1
s[2]	4	❶ 14	OT	html/body/table/tr/td/table/tr/td/text	<b> Location(s): </b>	6	-1	-1	6	-1	1	0	0	1	0
	5	15	OT	html/body/table/tr/td/table/tr/td/text	<b> Job Function: </b>	-1	6	6	-1	6	0	1	1	0	1
s[3]	6	21	MT	html/body/table/tr/td/table/tr/td/text	<b> Posted: </b>	8	8	8	8	8	1	1	1	1	1
	7	22	OT	html/body/table/tr/td/table/tr/td/text	17-Aug-11	9	9	-1	-1	-1	1	1	0	0	0
	8	15	OT	html/body/table/tr/td/table/tr/td/text	<b> Job Function: </b>	10	-1	-1	10	-1	1	0	0	1	0
	9	❷ 26	MT	html/body/table/tr/td/table/tr/td/text	<b> Entry Level: </b>	12	10	10	12	10	1	1	1	1	1
	10	30	OT	html/body/table/tr/td/table/tr/td/text	No	13	-1	-1	13	11	1	0	0	1	1
s[4]	11	28	OT	html/body/table/tr/td/table/tr/td/text	Yes	-1	11	11	-1	-1	0	1	1	0	0
	12	14	OT	html/body/table/tr/td/table/tr/td/text	<b> Location(s): </b>	-1	12	-1	-1	12	0	1	0	0	1
	13	31	MT	html/body/div/text	<br/>	14	14	12	14	14	1	1	1	1	1
	14	33	MT	html/body/div/div/text	Job Description	15	15	13	15	15	1	1	1	1	1
s[5]	15	31	OT	html/body/div/text	<br/>	-1	-1	15	17	17	0	0	1	1	1

(a) concatenating *LECs* from all segments

Index	LECId	Type	Path	Content	First Position					Occurrence Vector				
					d[1]	d[2]	d[3]	d[4]	d[5]	d[1]	d[2]	d[3]	d[4]	d[5]
1	1	MT	html/head/title/text	Career Center	1	1	1	1	1	1	1	1	1	1
2	2	MT	html/body/table/tr/td/table/tr/td/text	<b> Job ID: </b>	2	2	2	2	2	1	1	1	1	1
3	8	MT	html/body/table/tr/td/table/tr/td/text	<b> Position Title: </b>	4	4	4	4	4	1	1	1	1	1
4	14	MP	html/body/table/tr/td/table/tr/td/text	<b> Location(s): </b>	6	12	-1	6	12	1	1	0	1	1
5	15	MP	html/body/table/tr/td/table/tr/td/text	<b> Job Function: </b>	10	6	6	10	6	1	1	1	1	1
6	❷ 21	OT	html/body/table/tr/td/table/tr/td/text	<b> Posted: </b>	8	8	8	8	8	1	1	1	1	1
7	22	OT	html/body/table/tr/td/table/tr/td/text	17-Aug-11	9	9	-1	-1	-1	1	1	0	0	0
8	❶ 26	OT	html/body/table/tr/td/table/tr/td/text	<b> Entry Level: </b>	12	10	10	12	10	1	1	1	1	1
9	30	OT	html/body/table/tr/td/table/tr/td/text	No	13	-1	-1	13	11	1	0	0	1	1
10	28	OT	html/body/table/tr/td/table/tr/td/text	Yes	-1	11	11	-1	-1	0	1	1	0	0
11	31	MT	html/body/div/text	<br/>	14	14	12	14	14	1	1	1	1	1
12	33	MT	html/body/div/div/text	Job Description	15	15	13	15	15	1	1	1	1	1
13	31	OT	html/body/div/text	<br/>	-1	-1	15	17	17	0	0	1	1	1

(b) final *LECs* after merging recurring *OT*s

Fig. 6: Merging recurring OTs to obtain template LECs

In the above example, LEC_{15}, which is a pruned MT in Fig. 4(c), and LEC_{14} present two recurring OTs which could not be aligned well. This is a scenario caused by multi-order attribute-value pairs, which will be addressed by random procedure described below.

3.2.5 *Dealing with multi-order attribute-value sets*

As mentioned above, while most template and data are arranged in a particular order for all pages, attribute-value pairs can sometimes break the rule and could be rendered arbitrarily in different pages. For such multi-order attribute-value pairs, we need a random procedure for data extraction. Therefore, we propose sequential and random aligning procedure based on the final template LECs.

In this paper, we consider all the final LECs after merging recurring OT as templates, while others are data values. The system then aligns the corresponding leaf nodes in *TableL* into an aligned $TableA$ based on the first position vectors of these LECs. In other words, each column of $TableA$ represents either a template attribute or a data value.

For two adjacent MTs: $LEC[b]$ and $LEC[b']$ (where $b < b'$), in the final template LECs, if there is no LEC of *Type* MP and OT between $LEC[b]$ and $LEC[b']$, the leaf nodes with index between $LEC[b].FP[j]$ and $LEC[b'].FP[j]$ in each $d[j]$ are clustered based on their similarity defined by Eq. (6). For leaf node $f1$ and $f2$:

$$\begin{aligned} LeafSimilarity(f1, f2) = {} & StrSimilarity(f1.Path, f2.Path) \times \omega_P \\ & + Similarity(f1.IDSet, f2.IDSet) \times \omega_I \\ & + Similarity(f1.ClassSet, f2.ClassSet) \times \omega_C \\ & + Similarity(f1.TypeSet, f2.TypeSet) \times \omega_T \end{aligned} \tag{6}$$

where each similarity is calculate by Eq. (4) and ω_P, ω_I, ω_C, and ω_T are weights of *Path*, *IDSet*, *ClassSet*, and *TypeSet* respectively. In our experiment we define $\omega_P = 0.3$, $\omega_I = 0.2$, $\omega_C = 0.2$, $\omega_T = 0.3$ and $LeafSimilarity(f1, f2)$ is greater than θ_{Col}.

If, however, there exists $LEC[f]$ of *Type* MP or OT between $LEC[b]$ and $LEC[b']$, i.e. $b < f < b'$, we align leaf nodes based on $LEC[f]$ as follows.

- Pass 1: For each $LEC[f]$, assign the corresponding leaf nodes, the $LEC[f].FP[j]$-th leaf node in $d[j]$, to the same attribute column.
- Pass 2: For each $LEC[f]$, let $startIndex[j]$ be the index after the first position, i.e. $LEC[f].FP[j] + 1$, and $endIndex[j]$ be the smallest first posistion larger than $LEC[f].FP[j]$, i.e.

  $$endIndex[j] = \min\{LEC[c].FP[j] | b \leq c < b', LEC[c].FP[j] > LEC[f].FP[j]\} \tag{7}$$

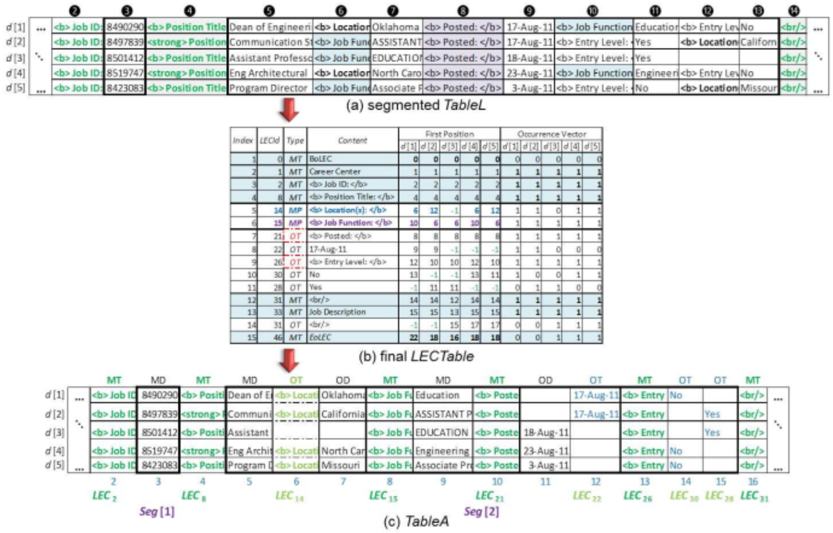
  We cluster leaf nodes with index between $startIndex$ and $endIndex$ in each $d[j]$ based on their similarity defined by Eq. (6) and insert each cluster to a data column after the attribute column corresponding to $LEC[f]$.

In summary, the system alternates between two procedures to deal with unordered FP vectors of the template LECs as follows:

1. **Single-pass alignment**: if there is no LEC of *Type* MP and OT between $LEC[b]$ and $LEC[b']$
   (a) For $LEC[b]$, align the $LEC[b].FP[j]$-th leaf node in each $d[j]$ to the same column in $TableA$.
   (b) For leaf nodes with index greater than $LEC[b].FP[j]$ and smaller than $LEC[b'].FP[j]$ in $d[j]$, if $LEC[b'].FP[j] - LEC[b].FP[j] > 2$, the system clusters leaf nodes based on their similarities, otherwise all leaf nodes are considered to be in one cluster. Finally, append a new data column after $LEC[b]$ for each cluster.
2. **Multi-pass alignment**: if there exists $LEC[f]$ of *Type* MP or OT between $LEC[b]$ and $LEC[b']$, i.e. $b < f < b'$
   (a) For each $LEC[f]$, where $b \leq f < b'$, align the corresponding leaf nodes, $LEC[f].FP[j]$-th leaf node in each $d[j]$ to the same column in $TableA$.
   (b) For each $LEC[f]$, if $endIndex[j] - startIndex[j] > 2$ for some j, the system clusters leaf nodes between $startIndex[j]$ and $endIndex[j]$ for all $d[j]$ based on their similarities, otherwise all leaf nodes are considered to be in one cluster. Finally, append the new data column after $LEC[f]$ for each cluster.

Example 6 For $LEC[3]$ and $LEC[4]$ in Fig. 7(b), there is no LEC of *Type* MP and OT, thus *TableL* will be processed by sequential extraction. All leaf nodes specified by $LEC[3].FP = [2, 2, 2, 2, 2]$, i.e. all second leaf nodes in each $d[j]$ (see column 2 for each document in Fig. 7(a)) will be aligned in an attribute column (see column LEC_2 in Fig. 7(c)). Other leaf nodes whose $LeafIndex < 4$, i.e. $d[j][3]$, become a data column and are aligned in column 3, i.e. $Seg[1]$, in Fig. 7(c) because similarities of their *Path*, *IDSet*, *ClassSet*, and *TypeSet* are less or equal to θ_{col}.

Another example, consider two adjacent MT $LEC[4]$ and $LEC[12]$ in Fig. 7(b). There are two MP, i.e. $LEC[5]$ and $LEC[6]$, and several OTs, i.e. $LEC[7] \sim LEC[11]$. Therefore, the system adopts multipass alignment for alignment. First, leaf nodes corresponding to the same template LEC are aligned in the same attribute column. For instance, the leaf nodes specified by $LEC_{14}.FP = [6, 12, -1, 6, 12]$, i.e. $d[1][6]$, $d[2][12]$, $d[4][6]$, and $d[5][12]$ in Fig. 7(a), are aligned in LEC_{14}, i.e. Column 6 in Fig. 7(c). To find the correponding values in the second pass, the following leaf nodes, i.e. $d[1][7]$, $d[2][13]$, $d[4][7]$, and $d[5][13]$ are considered as its value node and are aligned together in column 7 as shown in Fig. 7(c). Note that $d[1][8]$ and $d[2][14]$ are not futher considered because $endIndex[1] = 8$ and $endIndex[2] = null$.



		❷	❸	❹	❺	❻	❼	❽	❾	❿	⓫	⓬	⓭	⓮	
d[1]	...	<b> Job ID:	8490290	<b> Position Title	Dean of Engineeri	<b> Location	Oklahoma	<b> Posted: </b>	17-Aug-11	<b> Job Function	Education	<b> Entry Lev	No	<br/>	...
d[2]		<b> Job ID:	8497839	<strong> Position	Communication St	<b> Job Func	ASSISTANT	<b> Posted: </b>	17-Aug-11	<b> Entry Level:	Yes	<b> Location	Califom	<br/>	
d[3]	⋱	<b> Job ID:	8501412	<b> Position Title	Assistant Professo	<b> Job Func	EDUCATION	<b> Posted: </b>	18-Aug-11	<b> Entry Level:	Yes			<br/>	⋱
d[4]		<b> Job ID:	8519747	<strong> Position	Eng Architectural	<b> Location	North Caro	<b> Posted: </b>	23-Aug-11	<b> Job Function	Engineeri	<b> Entry Lev	No	<br/>	
d[5]	...	<b> Job ID:	8423083	<b> Position Title	Program Director	<b> Job Func	Associate F	<b> Posted: </b>	3-Aug-11	<b> Entry Level:	No	<b> Location	Missour	<br/>	...

(a) segmented *TableL*

Index	*LECId*	*Type*	*Content*	First Position					Occurrence Vector				
				d[1]	d[2]	d[3]	d[4]	d[5]	d[1]	d[2]	d[3]	d[4]	d[5]
1	0	*MT*	BoLEC	0	0	0	0	0	0	0	0	0	0
2	1	*MT*	Career Center	1	1	1	1	1	1	1	1	1	1
3	2	*MT*	<b> Job ID: </b>	2	2	2	2	2	1	1	1	1	1
4	8	*MT*	<b> Position Title: </b>	4	4	4	4	4	1	1	1	1	1
5	14	*MP*	<b> Location(s): </b>	6	12	-1	6	12	1	1	0	1	1
6	15	*MP*	<b> Job Function: </b>	10	6	6	10	6	1	1	1	1	1
7	21	*OT*	<b> Posted: </b>	8	8	8	8	8	1	1	1	1	1
8	22	*OT*	17-Aug-11	9	9	-1	-1	-1	1	1	0	0	0
9	26	*OT*	<b> Entry Level: </b>	12	10	10	12	10	1	1	1	1	1
10	30	*OT*	No	13	-1	-1	13	11	1	0	0	1	1
11	28	*OT*	Yes	-1	11	11	-1	-1	0	1	1	0	0
12	31	*MT*	<br/>	14	14	12	14	14	1	1	1	1	1
13	33	*MT*	Job Description	15	15	13	15	15	1	1	1	1	1
14	31	*OT*	<br/>	-1	-1	15	17	17	0	0	1	1	1
15	46	*MT*	*EoLEC*	22	18	16	18	18	0	0	1	1	1

(b) final *LECTable*

		MT	MD	MT	MD	OT	OD	MT	MD	MT	OD	OT	MT	OT	OT	MT	
d[1]	...	<b> Job ID	8490290	<b> Positi	Dean of E	<b> Locati	Oklahoma	<b> Job Fu	Education	<b> Poste		17-Aug-11	<b> Entry	No		<br/>	...
d[2]		<b> Job ID	8497839	<strong> F	Communi	<b> Locati	California	<b> Job Fu	ASSISTANT P	<b> Poste		17-Aug-11	<b> Entry		Yes	<br/>	
d[3]	⋱	<b> Job ID	8501412	<b> Positi	Assistant			<b> Job Fu	EDUCATION	<b> Poste	18-Aug-11		<b> Entry		Yes	<br/>	⋱
d[4]		<b> Job ID	8519747	<strong> F	Eng Archit	<b> Locati	North Car	<b> Job Fu	Engineering	<b> Poste	23-Aug-11		<b> Entry	No		<br/>	
d[5]	...	<b> Job ID	8423083	<b> Positi	Program D	<b> Locati	Missouri	<b> Job Fu	Associate Pr	<b> Poste	3-Aug-11		<b> Entry	No		<br/>	...
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
		LEC_2		LEC_8		LEC_{14}		LEC_{15}		LEC_{21}		LEC_{22}	LEC_{26}	LEC_{30}	LEC_{28}	LEC_{31}	
			Seg[1]							Seg[2]							

(c) *TableA*

Fig. 7: Alignment of leaf nodes in *TableL* into *TableA*

3.2.6 Leaf nodes rearrangement

In reality, the system may misclassified data columns as false negative attribute during OT detection. On the contrary, there are also false positive attributes because of different occurrence count in documents or the support is lower then a given threshold. Therefore we rearrangement leaf nodes in $TableA[f]$ between two adjacent MTs in $TableA[b]$ and $TableA[b']$ $(b < f < b')$, by merging: disjunctive columns, similar columns, and low density columns as follows.

Before we start, we define the density of a column (Eq. 8), the density for a section of contiguous columns (Eq. 9), and the similarity between two columns (Eq. 10) as follows:

$$colDensity[f] = \frac{\#Nonnull\,Contents\,in\,f}{|D|} \tag{8}$$

$$secDensity[f1 \sim f2] = \frac{\#Nonnull\,Contents\,in\,(f1 \sim f2)}{(f2 - f1 + 1) \times |D|} \tag{9}$$

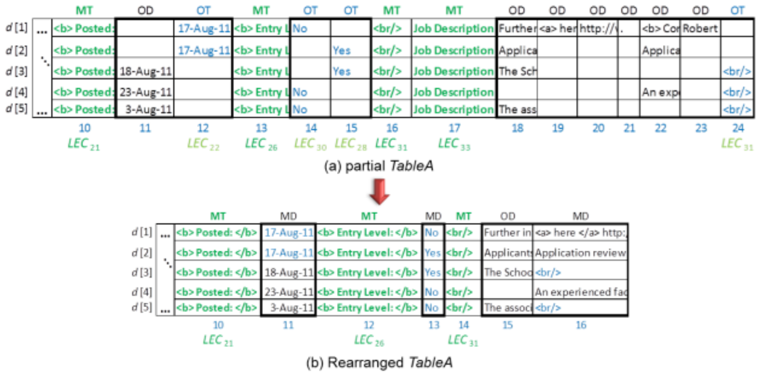
$$\begin{aligned} ColSimilarity(f1, f2) = (&StrSimilarity(f1.Path, f2.Path) \\ &+ Similarity(f1.IDSet, f2.IDSet) \\ &+ Similarity(f1.ClassSet, f2.ClassSet))/3 \end{aligned} \tag{10}$$

For each segment, i.e. the columns between two adjacent MTs in $TableA[b]$ and $TableA[b']$, we apply the following procedure to merge disjunctive columns, similar columns, and low density columns:

1. Merge disjunctive columns
   – For each column f $(b < f < b')$, if $TableA[f].OV \barwedge TableA[f+1].OV = true$, merge $TableA[f+1].Content$ into $TableA[f].Content$ and delete $TableA[f+1]$.
2. Merge similar columns
   – For two adjacent column f and $f+1$, if both have $colDensity$ smaller than $\theta_{Den} = 0.7$, and the column similarity between $TableA[f]$ and $TableA[f+1]$ is greater than θ_{Col}, we merge $TableA[f].Content$ with $TableA[f+1].Content$ and delete $TableA[f+1]$.
3. Merge low density columns
   – For optional column $TableA[f] \sim TableA[f']$ $(f < f')$ with density less than θ_{Den} (both $colDensity$ or $secDensity$), we update $TableA[f].Content$ with $\cup_{f \le b \le f'} TableA[b].Content$ and delete $TableA[b]$ for $f < b \le f'$.

Example 7 Consider $TableA[5 \sim 7]$ between two mandatory tempalte LEC_8 and LEC_{15} in Fig. 7(c), since they are not disjunctive and all $colDensity > \theta_{Dens}$, there is no need to rearrange the leaf nodes.

As shown in Fig. 8(a), there are three segments that should be processed. First, since $TableA[11].OV \barwedge TableA[12].OV = true$, they are merged into column 11 in 8(b) and column 12 is then removed. Similarly, since $TableA[14].OV \barwedge TableA[15].OV = true$, they are merged into column 13 in Fig. 8(b) and 15 is removed. Finally, consider $TableA[18 \sim 24]$ in Fig. 8(a), since $colDensity$ of $TableA[18] > 0.7$, it will not be processed. However, $secDensity$ of $TableA[19 \sim 24]$ is $0.33 < \theta_{Dens}$, therefore $TableA[19 \sim 24]$ are merged into column 16 in Fig. 8(b).



		MT	OD	OT	MT	OT	OT	MT	MT	OD	OD	OD	OD	OD	OD	OT
d [1]	...	<b> Posted:		17-Aug-11	<b> Entry L	No		<br/>	Job Description	Further	<a> her	http://	.	<b> Cor	Robert	
d [2]	⋱	<b> Posted:		17-Aug-11	<b> Entry L		Yes	<br/>	Job Description	Applica				Applica		
d [3]		<b> Posted:	18-Aug-11		<b> Entry L		Yes	<br/>	Job Description	The Sch						<br/>
d [4]		<b> Posted:	23-Aug-11		<b> Entry L	No		<br/>	Job Description					An exp		<br/>
d [5]	...	<b> Posted:	3-Aug-11		<b> Entry L	No		<br/>	Job Description	The ass						<br/>
		10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
		LEC_{21}		LEC_{22}	LEC_{26}	LEC_{30}	LEC_{28}	LEC_{31}	LEC_{33}							LEC_{31}

(a) partial *TableA*

		MT	MD	MT	MD	MT	OD	MD
d [1]	...	<b> Posted: </b>	17-Aug-11	<b> Entry Level: </b>	No	<br/>	Further in	<a> here </a> http:
d [2]	⋱	<b> Posted: </b>	17-Aug-11	<b> Entry Level: </b>	Yes	<br/>	Applicants	Application review
d [3]		<b> Posted: </b>	18-Aug-11	<b> Entry Level: </b>	Yes	<br/>	The Schoo	<br/>
d [4]		<b> Posted: </b>	23-Aug-11	<b> Entry Level: </b>	No	<br/>		An experienced fac
d [5]	...	<b> Posted: </b>	3-Aug-11	<b> Entry Level: </b>	No	<br/>	The associ	<br/>
		10	11	12	13	14	15	16
		LEC_{21}		LEC_{26}		LEC_{31}		

(b) Rearranged *TableA*

Fig. 8: Merging disjunctive columns

4 Performance evaluation

In this paper, we focus on detail-pages and do not take set and repetitive data, such as a table, into account. Therefore, we select datasets from two resources. The first resource is TEX (Sleiman and Corchuelo 2013)[3] dataset (T). We exclude websites containing tables and select only 22 from 41 websites. A total of 660 pages are used in the following experiments. The other resource is EXALG (Arasu and Garcia 2003)[4] dataset (E), where we select 4 websites (out of 9) which consist of 152 pages without tables. Overall, we have 26 websites which consist of 812 pages as shown in Table 2.

Since TEX does not output template and only provides golden answer of selected data columns (an average of 5 columns), we first consider the evaluation in terms of data columns. We define a column to be correctly extracted if

$$\frac{\#correct\,nonnull\,cells\,in\,extracted\,column}{\#nonnull\,cells\,in\,golden\,answer\,column} \geq 0.85 \tag{11}$$

Let cc be the number of correctly extracted columns, gc be the number of golden answer columns, and ec be the number of data columns extracted, we compute precision, recall and F1-measure, accordingly.

$$P = \frac{cc}{ec}, R = \frac{cc}{gc}, F = \frac{2PR}{P+R}$$

In addition to the evaluation on selected items, we also extend the golden answer to more data columns for more comprehensive performance evaluation. We label an average of 250 columns for each website based on the output of AFIX with further manually effort. Since TEX does not output template columns, we will follow the custom to focus on 47 data columns as shown in Table 2. The average column density for the data columns is 0.89.

[3] http://www.tdg-seville.info/Hassan/TEX
[4] http://infolab.stanford.edu/ arvind/extract/

Table 2: Data description

dataset id	website	#page	avg. column in *TableL*	#column *golden ans.*	#data columns	#columns by TEX	Density
T01	www.abebooks.com	30	311	285	44	5	0.85
T02	www.awesomebooks.com	30	231	207	21	5	0.89
T03	www.manybooks.net	30	256	158	39	4	0.90
T04	www.autotrader.com	30	526	455	117	8	0.88
T05	www.carmax.com	30	392	406	101	7	0.90
T06	www.classiccarsforsale.co.uk	30	590	559	51	9	0.93
T07	www.internetautoguide.com	30	304	299	92	8	0.94
T08	www.mbendi.com	30	46	46	13	4	1.00
T09	www.rdlearning.org.uk	30	69	55	15	4	0.82
T10	extapps.ama-assn.org	30	280	280	21	4	0.88
T11	www.drscore.com	30	102	107	31	4	0.85
T12	www.steadyhealth.com	30	393	395	106	3	0.96
T13	careers.insightintodiversity.com	30	143	119	14	3	0.93
T14	www.6figurejobs.com	30	600	563	48	3	0.87
T15	www.careerbuilder.com	30	273	228	38	3	0.85
T16	www.jobofmine.com	30	159	93	13	3	0.99
T17	www.albaniam.com	30	85	91	17	5	0.65
T18	www.allmovie.com	30	314	153	70	5	0.92
T19	www.citwf.com	30	111	105	36	5	0.76
T20	www.disneymovieslist.com	30	138	128	30	4	0.88
T21	www.remax.com	30	268	254	60	5	0.89
T22	www.atpworldtour.com	30	823	798	78	6	0.94
E01	teams.uefa.com	20	38	38	14	6	0.99
E02	www.ausopen.com	32	174	134	35	7	0.95
E03	www.ebay.com	50	190	171	46	3	0.75
E04	www.netflix.com	50	448	380	70	4	0.95
	average	**31**	**279**	**250**	**47**	**5**	**0.89**

We compare the proposed technique AFIS with the state-of-the-art annotation-free full page schema induction, such as TEX, FiVaTech, and RoadRunner. Table 3 gives the average performance of 26 websites on average 5 selected data columns provided by TEX. AFIS presents the best performance with 0.994 precision, 0.987 recall and 0.990 F1-measure. Note that the numbers are averaged from Sleiman and Corchuelo (2013), where FiVaTech deals with only 22 datasets and RoadRunner can process only 11 datasets.

If all data columns are considered, the performance drops for all algorithms as shown in Table 4 where detail statistics of AFIS, TEX, and RoadRunner are listed for reference. AFIS still outperforms others with 0.95 precision, 0.93 recall and 0.94 F1-measure. The *ec* columns shows the number of data columns output by each algorithm where AFIS outputs an average 46 data columns, TEX outputs 59 files (each corresponds to one column), and RoadRunner generates 8 data columns. Note that the golden answer for dataset E01 includes 14 data columns, where TEX outputs 18 columns. Technically, we can merge the data columns splitted by TEX, therefore, we still give a precision and recall of 1 for TEX on E01.

Table 3: Performance evaluation on selected data provided by TEX

	performance			standard error		
technique	P	R	F1	P	R	F1
AFIS	**0.994**	**0.987**	**0.990**	**0.003**	**0.005**	**0.003**
TEX	0.963	0.980	0.970	0.014	0.009	0.010
FivaTech	0.700	0.756	0.716	0.068	0.067	0.066
RoadRunner	0.339	0.358	0.348	0.085	0.090	0.087

5 Conclusions and future work

Annotation-Free training for web data extraction has been an important task for web data mining. Mining landmarks for aligning data is often hindered by false positive landmarks since the way to judge whether two processing units has the same function is often heuristic. Therefore, a more flexible design is necessry to solve the difficult alignment problem for detail-pages.

In this paper, we present an unsupervised design for annotation-free web data extraction on detail-pages. To discover landmarks for template mining, we adopt LIS (Longest Increasing Sequence) algorithm for mandatory template and optional template mining. Since there might be false positive *LEC* (Landmark Equivalence Class) detected in the template mining phase, we also allow the merging of recurring *LEC* to remove such false positive template. One more trick is the alignment of multi-order attribute-value pairs, where AFIS alternates between single-pass and multi-pass alignment to generate a consistent output. Finally, AFIS adopts various similarity measure for *LEC* detection, leaf node clustering and column clustering based on *Path*, *IDSet*, *ClassSet*, and *TypeSet*. Therefore, the system can decide when to merge leaf nodes to get final result.

We conducted experiments on real-world dataset from EXALG and TEX modified real-world datasets. Overall, AFIS outperforms TEX and RoadRunner not only on the small selected data (with 0.99 F1 measure) but also on all data columns from full schema (with 0.94 F1 measure), compared to 0.63 F1 measure for TEX and 0.29 F1-measure for RoadRunner.

For future study, since there are still lists and tables in detail-pages, we will include sequential and repetitive patterns for extracting list and table in detail-pages. Furthermore, we will generate leaf node abstraction via dynamic encoding for wrapper generation to reduce the extraction.

Table 4: Performance evaluation on 47 selected data columns from full schema

dataset		AFIS				TEX				RoadRunner		
id	ec	P	R	F1	ec	P	R	F1	ec	P	R	F1
T01	43	0.95	0.93	0.94	52	0.63	0.59	0.61	-	-	-	-
T02	21	1.00	1.00	1.00	37	0.64	0.76	0.70	20	1.00	0.95	0.98
T03	37	0.95	0.90	0.92	44	0.34	0.33	0.34	-	-	-	-
T04	113	0.94	0.91	0.92	156	0.55	0.54	0.54	-	-	-	-
T05	100	0.87	0.86	0.87	119	0.35	0.36	0.35	8	0.50	0.04	0.07
T06	43	0.98	0.82	0.89	93	0.52	0.84	0.65	-	-	-	-
T07	90	0.90	0.88	0.89	101	0.67	0.65	0.66	-	-	-	-
T08	13	1.00	1.00	1.00	13	0.61	0.85	0.71	13	1.00	1.00	1.00
T09	16	0.88	0.93	0.90	11	0.77	0.87	0.81	1	1.00	0.07	0.13
T10	19	0.95	0.86	0.90	24	0.70	0.67	0.68	6	0.67	0.19	0.30
T11	30	0.90	0.87	0.89	64	0.49	0.58	0.53	-	-	-	-
T12	106	1.00	1.00	1.00	54	0.65	0.52	0.58	3	1.00	0.03	0.05
T13	14	0.93	0.93	0.93	52	0.12	0.43	0.19	1	1.00	0.07	0.13
T14	48	0.92	0.92	0.92	55	0.70	0.83	0.76	-	-	-	-
T15	37	0.95	0.92	0.93	35	0.78	0.74	0.76	-	-	-	-
T16	13	1.00	1.00	1.00	22	0.60	0.92	0.73	12	1.00	0.92	0.96
T17	17	1.00	1.00	1.00	17	1.00	0.94	0.97	17	1.00	1.00	1.00
T18	67	0.96	0.91	0.93	66	0.70	0.73	0.71	-	-	-	-
T19	35	0.97	0.94	0.96	21	0.89	0.92	0.90	35	1.00	0.97	0.99
T20	30	0.90	0.90	0.90	53	0.44	0.71	0.54	-	-	-	-
T21	60	0.90	0.90	0.90	83	0.54	0.70	0.61	-	-	-	-
T22	78	0.95	0.95	0.95	104	0.50	0.64	0.56	-	-	-	-
E01	14	1.00	1.00	1.00	18	1.00	1.00	1.00	14	1.00	1.00	1.00
E02	35	1.00	1.00	1.00	54	0.55	0.63	0.59	35	1.00	1.00	1.00
E03	47	0.83	0.85	0.84	81	0.38	0.54	0.45	-	-	-	-
E04	70	0.99	0.99	0.99	95	0.42	0.60	0.49	-	-	-	-
Avg	**46**	**0.95**	**0.93**	**0.94**	59	0.60	0.69	0.63	8	0.43	0.28	0.29
std err		**0.01**	**0.01**	**0.01**		0.04	0.03	0.04		0.10	0.09	0.09

References

Arasu A, Garcia MH (2003) Extracting structured data from web pages. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, pp 337-348

Bergman MK (2001) White paper: the deep web: surfacing hidden value. Journal of electronic publishing 7(1)

Bronzi M, Crescenzi V, Merialdo P, Papotti P (2013) Extraction and integration of partially overlapping web sources. In: Proceedings of the VLDB Endowment 6(10), pp 805-816

Chang CH, Kayed M, Girgis MR, Shaala KF (2006) A survey of web information extraction systems. Journal of IEEE Transactions on Knowledge and Data Engineering 18(10), pp 1411-1428

Chang CH, Lui SC (2001) IEPAD: information extraction based on pattern discovery. In: Proceedings of the 10th international conference on World Wide Web, pp 681-688

Clark A, Guillemot M (© 2013) CyberNeko HTML Parser. http://nekohtml.sourceforge.net

Crescenzi V, Mecca G (2004) Automatic information extraction from large websites. Journal of the ACM (JACM) 51(5), pp 731-779

Dalvi BB, Cohen WW, Callan J (2012) Websets: Extracting sets of entities from the web using unsupervised information extraction. In: Proceedings of the 5th ACM international conference on Web search and data mining, pp 243-252

Fredman ML (1975) On computing the length of longest increasing subsequences. Journal of Discrete Mathematics 11(1), pp 29-35

Hao Q, Cai R, Pang Y, Zhang L (2011) From one tree to a forest: a unified solution for structured web data extraction. In: Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, pp 775-784

Hsu CN, Chang CC (1999) Finite-state transducers for semi-structured text mining. In: Proceedings of IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications, pp 38-49

Kayed M, Chang CH (2010) FiVaTech: Page-level web data extraction from template pages. Journal of IEEE Transactions on Knowledge and Data Engineering 22(2), pp 249-263

Kushmerick N (1997) Wrapper induction for information extraction. University of Washington

Lu Y, He H, Zhao H, Meng W, Yu C (2013) Annotating search results from Web databases. Journal of IEEE Transactions on Knowledge and Data Engineering 25(3), pp 514-527

Peters ME, Lecocq D (2013) Content extraction using diverse feature sets. In: Proceedings of the 22nd international conference on World Wide Web companion, pp 89-90

Sarawagi S (2008) Information extraction. Journal of Foundations and trends in databases 1(3), pp 261-377

Sleiman HA, Corchuelo R (2013) TEX: An efficient and effective unsupervised web information extractor. Journal of Knowledge-Based Systems 39, pp 109-123

Su W, Wang J, Lochovsky FH, Liu Y (2012) Combining tag and value similarity for data extraction and alignment. Journal of IEEE Transactions on Knowledge and Data Engineering 24(7), pp 1186-1200

Sun F, Song D, Liao L (2011) Dom based content extraction via text density. In: Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, pp 245-254

Uzun E, Agun HV, Yerlikaya T (2013) A hybrid approach for extracting informative content from web pages. Information Processing & Management 49(4), pp 928-944

Weninger T, Hsu WH, Han J (2010) CETR: content extraction via tag ratios. In: Proceedings of the 19th international conference on World wide web, pp 971-980

Zhao H, Meng W, Yu C (2007) Mining templates from search result records of search engines. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 884-893

Zheng X, Gu Y, Li Y (2012) Data extraction from web pages based on structural-semantic entropy. In: Proceedings of the 21st international conference companion on World Wide Web, pp 93-102

AFIS: Annotation-Free Induction of Full Schema for Detail-Pages

ORIGINALITY REPORT

47%	19%	48%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Oviliani Yenty Yuliana, Chia-Hui Chang. "AFIS: Aligning detail-pages for full schema induction", 2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI), 2016 Publication	42%
2	scholars.ncu.edu.tw Internet Source	3%
3	repository.petra.ac.id Internet Source	3%

Exclude quotes	On	Exclude matches	< 1%
Exclude bibliography	On		