A novel alignment algorithm for effective web data extraction from singleton-item pages

by Oviliani Yenty Yuliana

Submission date: 12-Jan-2023 08:19PM (UTC+0700)
Submission ID: 1991734759
File name: Revision_2018-04-20.pdf (4.47M)
Word count: 11910
Character count: 50828

Applied Intelligence manuscript No.
(will be inserted by the editor)

A novel alignment algorithm for effective web data extraction from singleton-item pages

Oviliani Yenty Yuliana · Chia-Hui Chang

Received: date / Accepted: date

Abstract Automatic data extraction from template pages is an essential task for data integration and data analysis. Most researches focus on data extraction from list pages. The problem of data alignment for singleton pages, which contain detail information of a single item is less addressed and is more challenging because the number of data attributes to be aligned is much larger than list pages. In this paper, we propose a novel alignment algorithm working on leaf nodes from the DOM trees of input pages for singleton pages data extraction. The idea is to detect mandatory templates via the longest increasing sequence from the landmark equivalence class leaf nodes and recursively apply the same procedure to each segment divided by mandatory templates. By this divide-and-conquer approach, we are able to efficiently conduct local alignment for each segment, while effectively handle multi-order attribute-value pairs with a two-pass procedure. The results show that the proposed approach (called Divide-and-Conquer alignment, DCA) outperforms TEX [23] and WEIR [4] 2% and 12% on selected items of TEX and WEIR dataset respectively. The improvement is more obvious in terms of full schema evaluation, with 0.95 (DCA) versus 0.63 (TEX) F-measure, on 26 websites from TEX and EXALG [1].

Extended paper of TAAI 2016 [28]

O. Y. Yuliana
CSIE, National Central University
Taoyuan 32001, Taiwan
E-mail: oviliani@gmail.com

C.-H. Chang ✉
CSIE, National Central University
Taoyuan 32001, Taiwan
E-mail: chia@csie.ncu.edu.tw

Keywords Web data extraction · Template pages · Singleton pages · Full-schema · Divide-conquer alignment · Multiple string alignment

1 Introduction

The World Wide Web, along with static web pages, contains a tremendous number of dynamic web pages which are generated through web query interfaces upon users' requests. These online databases (structured data) make up the deep web and generate search result pages (semi-structured data) by embedding data into their predefined templates. Deep web contains valuable data and resources for knowledge harvesting and decision making. Therefore, automatic data extraction from the deep web has been an important technique for information integration and data analysis in various applications from commercial to social web application [13].

As reported in [16], Internet-accessible databases contain up to 500 times more data than the static Web and roughly 70% of websites are backed by relational databases. A recent study in [18] indicates that there exist more than 450 billion deep web pages. If the data hidden in deep Web can be effectively and efficiently reverse engineered to the original database, we can apply direct mapping to translate the relational database to an RDF graph with OWL vocabulary as suggested in [22]. Compared with literatures that extract information from static pages for populating cross-domain knowledge bases [2] or harvest structured facts to automatically add novel statements to DBpedia [14] deep web data extraction can speed up the extraction procedure for data instances of the same relational schema (even though we need to construct one wrapper for each website).

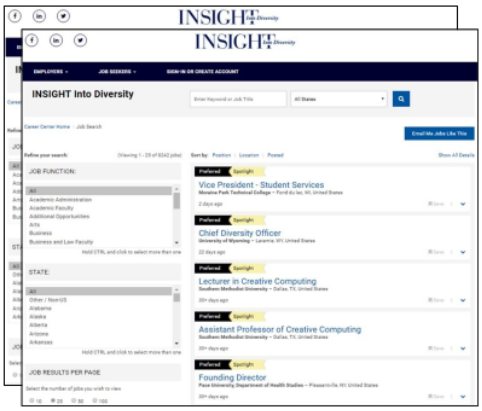
In this paper, we study the problem of web data extraction from dynamic web pages which are generated through web query interfaces upon users' requests. We focus on singleton pages, which contain details of a single item in a page as opposed to list pages, which contain a list of items in a web page. For example, a page containing search result of a query is a list page, while a page containing description of a item (like job vacancy) is a singleton page (see Fig. 1). For the past decade, most researches tried to solve **record extraction** from list pages, e.g. CTVS [24] and Lu at al. [20]. Very few researches focus on **full schema induction** for singleton pages according to the surveys in [5,13]. In other words, the performance evaluation usually focused on selected data items of list pages, the performance on full schema and item details still has a room for improvement.

The difficulties of aligning singleton pages come from several aspects. First, the number of data attributes needs to be aligned is much larger than that of data records in list pages. For example, there are more rich data types and more optional data to be processed in singleton pages as shown in Fig. 2 ①.

Second, it is more likely to have multiple order attribute-value pairs which lead to unordered data rendering. For example, "Job Function" and "Entry Level" attributes come either before or after "Location(s)" and are rendered on either left or right side in Fig. 2 ②. For such cases, even visual position or layout feature extraction (as proposed by Hao at al. [15]) cannot solve multi-order attribute-value pair problem completely.

Third, the data-rich section for singleton pages is hard to define and can span to the whole page. Therefore, we aim to induce the full schema for the whole page. Because of this, there are more situations we need to consider. For example, while the same text contents usually play the same role in a page, some of them might have different functions in the different position of the page when considering the whole page. On the other hand, while the same text contents usually have the same path, some of them might have different paths because of decorative tags. Last but not least, list data (if any) inside the singleton pages usually occupies a small area and has less evidence for discovery as in list pages.

In this paper, we propose a novel algorithm for unsupervised web data extraction from singleton pages. The proposed technique operates on leaf nodes of input DOM trees. We adopt a Divide-and-Conquer approach to recursively detect landmarks (called mandatory templates) via longest increasing sequence (LIS) for template mining. By focusing on leaf nodes with the same text content (which are defined as landmark equivalence class (LECs)) and selecting LECs with consistent ordering in all pages as landmarks, we are able to rearrange leaf nodes to achieve a better alignment for data extraction.



(a) List pages

(b) Singleton pages

Fig. 1: Examples of list pages and singleton pages

We conduct experiments using real-world web pages from the following three papers: WEIR [4], TEX [23], and EXALG [1]. In terms of selected data items with golden answer (5% of data columns) on WEIR dataset, our approach ($F = 0.96$) outperforms the state-of-the-art approaches like RoadRunner [9] ($F = 0.66$), WEIR ($F = 0.88$) and TEX ($F = 0.5$). For full schema evaluation with our manually annotated golden answers on TEX and EXALG dataset, the proposed approach ($F = 0.95$) shows even larger gap with TEX ($F = 0.63$) and RoadRunner ($F = 0.29$).

The main contributions in this paper are (i) we propose a novel data alignment technique for singleton

(a) Multi-order attribute-value pair problem in singleton pages

MT	MD	MT	MD	MT	MD	MT	MD	MT	MD	MT	MD	OT	OD	MT	MD	MT	MD	
Back to Results	Post Doctoral Researcher in Hydrology	Job ID:	8512218	Position Title:	Post Doctoral Researcher in Hydrology	Company Name:	New Mexico Tech	Job Function:	Academic	Entry Level:	Yes	Job Type:	Full-Time	Location(s):	Socorro, New Mexico, 87801	Posted:	August 20, 2011	...
Back to Results	Professors, Associate Professors, and Assistant Professors	Job ID:	8296386	Position Title:	Professors, Associate Professors, and Assistant Professors	Company Name:	Masdar Institute of Science and Technology	Job Function:	Education, Non-Consulting	Entry Level:	No			Location(s):	Aub Dhabi, Other / Non-US, U.A.E.	Posted:	July 11, 2011	...

(b) Aligned multi-order attribute-value pair for both singleton pages

Fig. 2: An example of multi-order attribute-value pair in singleton pages and the aligned result

pages, (*ii*) we apply LIS to deal with inconsistent landmarks, i.e. multi-order templates, and (*iii*) we extract full schema from singleton pages and compare the effectiveness and efficiency of the divide-and-conquer alignment (DCA) with state-of-the-art techniques on three bench datasets from several domains in real-world websites.

The rest of paper is organized as follows. In the next section, we compare the proposed problem with related Web data extraction techniques. A formal problem statement and the motivation behind the algorithm is introduced in Section 3. We describe our proposed method in Section 4 followed by alignment phase in Section 5. The performance evaluations are presented and analyzed in Section 6. Finally, we conclude our paper and propose our future work in Section 7.

2 Related work

The research of Web information extraction can be traced back to the early stage of Web development in 1996. Many information extractions (IE) approaches have been proposed with diverse degree of automation (i.e. supervised [10], semi-supervised [3,12], and unsupervised [11]) [19]. Supervised IE approaches, which are originally designed for input pages from different websites (i.e. with various structure), require annotated training web-pages to build a model. Unsupervised IE approaches, which are designed for input pages generated from the same website (i.e. with the same template), accept annotation-free deep web-pages as training set and discover data-rich section for data extraction.

The possibility of unsupervised web data extraction relies on the regularity of semi-structured web pages or template pages. From early approaches like EXALG [1], RoadRunner [9], RBM-TD [25], FiVaTech [19], ONDUX [7], and JUDIE [8], the challenging task continues to attract increasing attention from recent work like CTVS [24], DE-SSE [30], TEX [23], WEIR [4], and CLG [27], etc.

While all these researches claim to be unsupervised, they target on different extraction tasks specified by input and output. For example, an extraction target may be the search result records from list pages (e.g. DEPTA[29]) or attribute-value pairs from singleton pages (e.g. DE-SSE). As another example, CETR focuses on the extraction of main (news) content from article web pages [26], while CLG deals with non-article (template) pages. In addition, ONDUX and JUDIE aim to extract continuous text containing implicit semi-structured records.

In this paper, we are particularly interested in algorithms that are designed for full page web data extraction rather than record or major content extraction. Some of the major work include RoadRunner, EXALG, FivaTech, and TEX. According to the web page generation model defined in EXALG, a web page is generated by encoding an instance x of its data schema S into a predefined template $T(S)$, where the template for a schema $T(S)$ is defined as a function that maps each type constructor of S into an ordered set of strings.

RoadRunner learned a union-free regular expression by generating a base template from the first web page then it compared literately with another web page using a string alignment algorithm. Meanwhile, RoadRunner applied a backtracking algorithm for detecting optional and repetitive patterns.

EXALG operates on strings of word and HTML tag tokens. The idea is to detect templates from large and frequent equivalence classes (LFEQs) that have the

same occurrence vectors across the input pages. By iteratively removing some of the invalid LFEQs that violate the ordered and nesting properties, the algorithm then uses the ordered set of nested LFEQs for template construction. The major problem with EXALG is the limitation to deal with an inconsistent data sequence and choose the right equivalence classes resulted from optional templates.

FiVaTech induced the template and schema for a set of given pages from a common DOM tree structure generated from input. It applied a tree edit distance to measure the similarity between two sibling nodes at the same level of alignment. FiVaTech also employed a mining technique to mine the repetitive patterns and several heuristics to detect optional information.

TEX found and discarded the shared longest sequence tokens amongst web documents until finding the relevant information that should be extracted from them. In other words, TEX extracted various information from web documents and removed information that belongs to the template.

3 Problem definition

In this paper, we formulate the problem of Web data extraction as aligning leaf nodes from the DOM trees of m input pages (each DOM tree becomes a row of input $TableL$) into a table ($TableA$) of m rows and l columns such that leaf nodes with the same role are aligned in the same column. Intuitively, template columns usually have same text contents while data columns often contain various contents. In addition, columns could be mandatory or option if there are missing elements in pages. Note that the problem definition ignores list data and treats each record independently as multiple columns. In other words, we define the output of the problem to be the aligned matrix such that either template columns or data columns could be mapped directly with some schema for data extraction.

Compare to the problem definition of EXALG or TEX, which aligns HTML tags and word tokens, leaf nodes are more complete as a processing unit, whereas tags and word tokens are usually part of a larger information unit. Operating directly on them usually results in more noise and complexity. The challenge here is that (i) leaf nodes of the same template may have different path because of decorative tag or CSS, (ii) leaf nodes with the same text content or path in input pages may play the different role, (iii) aligning leaf nodes of the same role may rely on various similarity measures, and (iv) multi-order attribute-values pairs need to be reordered for a consistent alignment.

4 The proposed method

In this paper, we propose a divide-and-conquer alignment (DCA) algorithm that processes all input sequences at the same time (like EXALG and TEX), rather than iteratively merging two input web pages as RoadRunner does. We define leaf nodes with the same text content and similar paths as landmark equivalence class (LEC) and select landmarks with the same occurrences across all pages for possible templates. We then examine the first occurrence positions of these candidate templates in each page for order-consistency checking. The insight here is that we enforce the order constraint on landmark selection via the longest increasing sequence (LIS) algorithm and break down the problem into several subproblems. Therefore, the proposed DCA algorithm only needs to focus on one equivalence class that involve all landmarks with the same occurrence count across all pages, instead of dealing with many LFEQs and checking for their validities (ordered and nested) with respect to other LFEQs like EXALG.

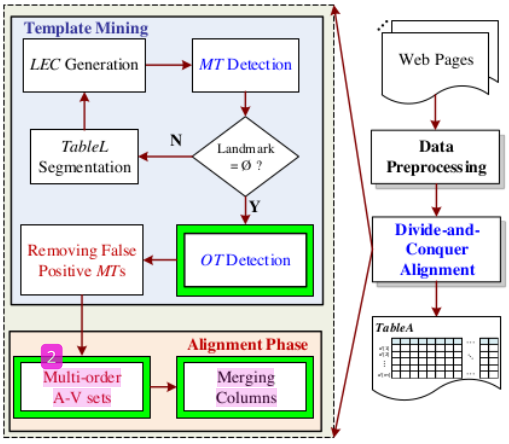


Fig. 3: System architecture

The proposed algorithm first performs data preprocessing and then divide-and-conquer alignment as shown in Fig. 3. In data pre-processing, we parse all the given web pages into DOM trees, collect features for each leaf node from DOM trees, and arrange all leaf nodes into a table called $TableL$ as an input for next step. The divide-and-conquer algorithm can be further divided into template mining phase and alignment phase. The template mining phase discovers LECs from $TableL$ and divides the table into segments by detecting Mandatory Template (MT) from LECs with the same occur-

rence count across all input pages. To avoid incorrect segmentation, the template mining phase further detects Optional Templates (OT) and merges them across segments to remove false positive mandatory templates. Finally, in the alignment phase, we align leaf nodes that are not templates to generate a consistent output for multi-order attribute-value pairs and merge similar/disjunctive columns to generate the output matrix *TableA*.

4.1 Data preprocessing

As mentioned above, we use leaf nodes of DOM trees as our basic processing unit. The reason is that the number of leaf nodes is much smaller than the number of tag and word tokens. As an illustration, the average number of leaf nodes for TEX dataset (with an average size of 77KB from 26 websites) is 798, while the average number of tag and word tokens is 4,548. Beside, leaf nodes carry information that can be used to differentiate the roles of each leaf node. The features that we collect for each leaf node include *LeafIndex*, *Path*, *IDSeq*, *ClassSeq*, *Content*, and *TypeSet* as shown below.

- *LeafIndex* is the index of the leaf node l in a page.
- *Path* is the sequence of tags from the root to the leaf node l.
- *IDSeq* is the sequence of id attributes from all tags in the *Path*.
- *ClassSeq* is the sequence of class attributes from all tags in the *Path*.
- *Content* is the text content of a leaf node.
- *TypeSet* is the union of token types for all tokens in *Content* as defined in Eq. (1), where $|l.Content|$ is the number of tokens in the text content.

$$TypeSet(l) = \bigcup_{c=1}^{|l.Content|} type(l.Content[c]) \quad (1)$$

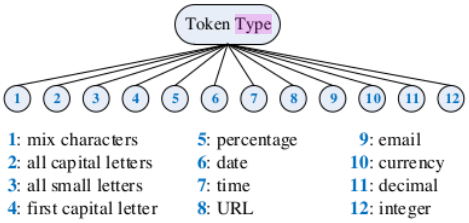


Fig. 4: Token types

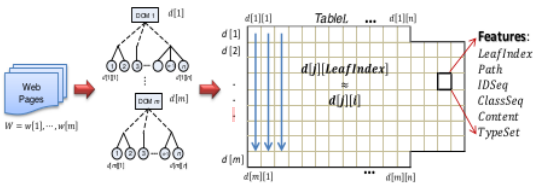


Fig. 5: Data preprocessing framework

Note that *type*() function is implemented based on regular expression and returns one of the 12 token types for input as defined in Fig. 4.

In summary, given m input pages with the same template, we parse each input page $p_j, 1 \leq j \leq m$ using **CyberNeko**[1] into a **DOM Tree** and collect all leaf nodes $d[j][1]$, $d[j][2], \ldots, d[j][n_j]$ of this page into $d[j]$ in one row in *TableL* as shown in Fig. 5. Note that if the last tag of a given *Path* is a decorative tag[2] or $\langle br/\rangle$, we remove such tags from *Path* to *Content*.

4.2 DCA algorithm overview and definitions

Divide-and-conquer alignment (DCA) is the core of our work in this paper. In the template mining phase, we regard leaf nodes with the same content and similar path as landmarks and prioritize landmarks into Mandatory Template (MT), Optional Template (OT), and data nodes types. The divide-and-conquer alignment is done by a recursive call to the MT alignment (4.3) followed by OT alignment (4.4) for each segment. The procedure stops when no MT is detected in a segment. While the heuristic definition of MT, i.e. landmarks with the same occurrence count in each page, and OT, i.e. landmarks with the same occurrence count or missing in a page, may include false positive templates, the utilization of LIS and merging reoccurring OT (4.5) can detect such mistakes by maintaining a consistent order of templates.

In the alignment phase, the proposed algorithm handle mul-tiorder attribute-value pairs via a two-pass procedure during data nodes clustering (5.1), where leaf nodes with different contents are clustered into groups based on node similarities. Finally, disjunctive columns or low-density and similar columns are merged to reduce the number of output columns (see 5.2). The complete algorithm is shown in Algorithm 1.

[1] http://nekohtml.sourceforge.net, CyberNeko HTML Parser, accessed 10 January 2017

[2] $DecorativeTag \equiv \{a, b, big, cite, dfn, font, em, i, mark, small, span, sub, sup, strike, u, strong\}$

Algorithm 1 DCA

```
1: procedure DCA(TableL)
2:     LECTable ← LECGeneration(TableL)
3:     final MTTable ← MTDetection(LECTable)
4:     final OTTable ← OTDetection(LECTable)
5:     TLEC[a] ← RemoveFPMT[b](final MTTable, final OTTable)
6:     TLEC ← MO_AV_Set[c](TLEC)
7:     TableA ← MergeCol[d](TLEC, TableL)
8:     return TableA
9: end procedure
```

[a] $TLEC$ denotes $TemplateLEC$
[b] RemoveFPMT denotes Removing False Positive MTs
[c] MO_AV_Set denotes Multi-order A-V sets
[d] MergeCol denotes Merging Columns

We start with leaf node encoding for LEC generation.

Definition 1 (Equivalent nodes) Two leaf nodes u and v are considered as equivalence leaf nodes if they have the same $Content$ and the similar $Path$, i.e. Sim ($u.Path$, $v.Path$) in Eq. (2) is greater than or equal to a given $Path$ threshold (θ_{Path}).

We define $sim(u, v)$ of two strings s_1 and s_2 as:

$$Sim(u, v) = \frac{LCS(u, v)}{max(|u|, |v|)} \quad (2)$$

where LCS is the longest common subsequence, $|s_1|$ and $|s_2|$ are the length of two strings s_1 and s_2. Note that if a $Content$ contains decorative tags, we consider ≅ and <i>≅ during similarity calculation. This is because people could use either tags to emphasize their idea. For encoding purpose, we consider them to be the same in order to highlight the pattern.

With the definition of equivalent nodes, we can generate $LECs$ from $TableL$ and compute their occurrence vector and first position vector as follows.

Definition 2 (Landmark Equivalence Class) All equivalent leaf nodes of the same class form an LEC with a unique $LECId$.

Definition 3 (Occurrence Vector and First Position) The Occurrence Vector (OV) of a landmark equivalence class LEC_e is a vector of occurrence count o_j of LEC_e leaf nodes in each $d[j]$, i.e. $OV_e = [o_1, o_2, \cdots, o_m]$. The First Position ($FP$) of an LEC_e is a vector $FP_e = [p_1, p_2, \cdots, p_m]$, where p_j is the first occurrence position of LEC_e in $d[j]$ or **-1** if missing in $d[j]$. That is, p_j is the smallest i for all $d[j][i]$ with $LECId = e$.

To generate LEC table ($LECTable$), the system applies a single-pass clustering to all leaf nodes in $TableL$ column-wise (as shown by the vertical blue arrows in Fig. 5). The detail of LEC generation process is shown in Algorithm 2. After reordering, each LEC has an index in the $LECTable$ in addition to $LECId$. In the following, we use $LEC[i]$ and LEC_e to refer the i-th LEC in the $LECTable$ and an LEC with $LECId = e$, respectively.

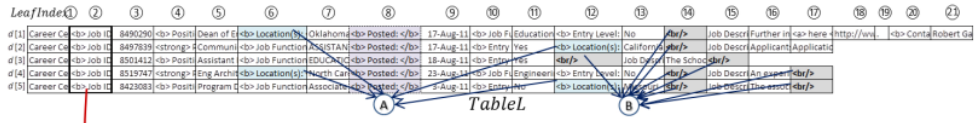
Algorithm 2 LEC generation

```
1: procedure LECGeneration(TableL)
2:     Initialize LECTable[Id = 1] with leaf node d[1][1],
       OV1 = [1, 0, ···, 0], and FP1=[1,-1,···,-1]
3:     for j ← 1, maxc do // the largest leaf index
4:         for i ← 1, m do // the # of input documents
5:             if d[i][j] is equivalent to some LEC_e then
6:                 OV_e = [o1, ···, oi++, ···, om] and
                   assign FP_e[i] with j if FP_e[i] = −1
7:             else
8:                 Initialize a new LEC_e with index e =
                   Id++ and leaf node d[i][j], FP_e[i] = i,
                   OV_e[i] = 1, and FP_e[k] = −1, OV_e[k] = 0
                   for k ≠ i.
9:             end if
10:        end for
11:    end for
12:    MinFP_e = min{FP_e[i]|FP_e[i] ≥ 0, 1 ≤ i ≤ m}
       Reorder LECTable by MinFP_e and LECId if the
       MinFP for two LECs is the same.
13:    return LECTable
14: end procedure
```

Example 1 An illustration of generating $LECTable$ from $TableL$ is shown in Fig. 6, where the top table shows the leaf nodes from 5 input pages and the bottom table shows the constructed $LECTable$. The leaf nodes in the first column of $TableL$ (with $Content$ = "Career Center") and the second column (with $Content$ = "Job ID:") form two equivalence classes $LECId$ = 1 and 2 respectively, while the leaf nodes in the third column, with different text content, form five $LECs$. LEC_{14} (see Ⓐ), with missing Content=" Location(s): " in $d[3]$, has occurrence vector OV_{14} = [1,1,0,1,1] and FP_{14} = [6,12,-1,6,12]. LEC_{32} (see Ⓑ) with two occurrences of $Content$="
" in $d[3]$, $d[4]$, $d[5]$ has OV_{32} vector= [1,1,2,2,2] and FV_{32}=[14,14,12, 14,14].

4.3 Mandatory template (MT) detection

Given the $LECTable$, the next step is to identify mandatory templates for web page segmentation and divide-and-conquer. We first consider $LECs$ with the same occurrence count in each document as a candidate landmark for mandatory template detection.



Index	LECId	Path	Content	First Position d[1]	d[2]	d[3]	d[4]	d[5]	Occurrence Vector d[1]	d[2]	d[3]	d[4]	d[5]
1	1	html/head/title/text	Career Center	1	1	1	1	1	1	1	1	1	1
2	2	html/body/table/tr/td/table/tr/td/text	Job ID:	2	2	2	2	2	1	1	1	1	1
3	3	html/body/table/tr/td/table/tr/td/text	8490290	3	-1	-1	-1	-1	1	0	0	0	0
⋮		⋱	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
7	7	html/body/table/tr/td/table/tr/td/text	8423083	-1	-1	-1	-1	3	0	0	0	0	1
8	8	html/body/table/tr/td/table/tr/td/text	Position Title:	4	4	4	4	4	1	1	1	1	1
⋮		⋱	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	14	html/body/table/tr/td/table/tr/td/text	Location(s):	6	12	-1	6	12	1	1	0	1	1
15	15	html/body/table/tr/td/table/tr/td/text	Job Function:	10	6	6	10	6	1	1	1	1	1
⋮		⋱	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
21	21	html/body/ table/tbody/tr/td/table/tbody/t	** Posted: **	8	8	8	8	8	1	1	1	1	1
⋮		⋱	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
32	32	html/body/div/text	 	14	14	12	14	14	1	1	2	2	2
⋮		⋱	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
45	45	html/body/div/div/text	Robert Gass	21	-1	-1	-1	-1	1	0	0	0	0

LECTable

Ⓐ is LEC_{14} with $FP_{14} = [6, 12, -1, 6, 12]$ and $OV_{14} = [1, 1, 0, 1, 1]$ and Ⓑ is LEC_{32} with $FP_{32} = [14, 14, 12, 14, 14]$ and $OV_{32} = [1, 1, 2, 2, 2]$

Fig. 6: Example of generating *LECTable*

Definition 4 (Candidate Mandatory Template) An *LEC*s with the same occurrence count k (k: a positive integer) in each $d[j]$, i.e. OV = **k** is called an candidate Mandatory Template (*MT*). (**k** denotes a vector.)

However, not all such *LEC*s could be used because of inconsistent order in documents. As shown in Fig. 7, two candidate MT LEC_{15} and LEC_{21} (both with OV=**1**) have $FP_{15} = [10, 6, 6, 10, 6]$ and $FP_{21} = [8, 8, 8, 8, 8]$, which means LEC_{15} may either appear earlier or later than LEC_{21}.

To maintain a consistent order of these *MT*s, we apply LIS, which can be implemented by finding the longest common sequence between the input sequence and the sorted input sequence, to keep an increasing FP order for all selected *MT*s in each page. In other words, LIS will select *MT*s with consistent FP to ensure $MTTable[k].FP[j] < MTTable[k'].FP[j]$ for every index $k < k'$ in all j (where $MTTable$ is the candidate *MT* selected from $LECTable$). The complete algorithm for mandatory template detection and divide-and-conquer procedures are shown in Algorithm 3.

Example 2 As shown in Fig. 7, the system selects candidate *MT*s with OV = **k** into $MTTable$. Next, the system applies LIS on FPs in each document. Since FP of these *MT*s ($LECId$= 1, 2, 8, 15, 21, 26, 33) are unordered in $d[1]$, i.e. {1, 2, 4, 10, 8, 12, 15}, LEC_{15} is removed as shown at the final $MTTable$ in Fig. 7. These selected *MT* are then used for segmenting $TableL$ into 5 segments.

Algorithm 3 *MT* detection

```
1: procedure MTDetection(LECTable)
2:     MTTable ← select candidate MTs from LECTable,
       i.e. LEC_e with OV_e = k
3:     final MTTable ← apply LIS on the FPs of MTs from
       MTTable for each d[i] (where i = 1, ···, m) to remove
       inconsistent MTs
4:     if final MTTable = ∅ then
5:         return
6:     end if
7:     Use the FPs in final MTTable for segmenting TableL
       into subTableLs
8:     for each subTableL do
9:         newLECTable ← LECGeneration(subTableL)
10:        MTDetection(newLECTable)
11:    end for
12:    return final MTTable
13: end procedure
```

For each segment, we need to assemble new $LECTable$ and re-evaluate the FP and OV for each LEC in each segment. This is because an LEC may appear in more than one segment. For example, LEC_{14} with $Content$ = "Location(s):" appears in both segment $s[2]$ and $s[4]$ because of different orders in documents, whereas LEC_{32} with $Content$ ="
" appears in segment $s[4]$ and $s[5]$ due to multiple occurrences. Therefore, the FPs and OVs have to be evaluated for each segment. Once new $LECTable$s are prepared, MT detection is then called for each segment recursively. For example, LEC_{32} in $Seg[4]$ becomes a MT.

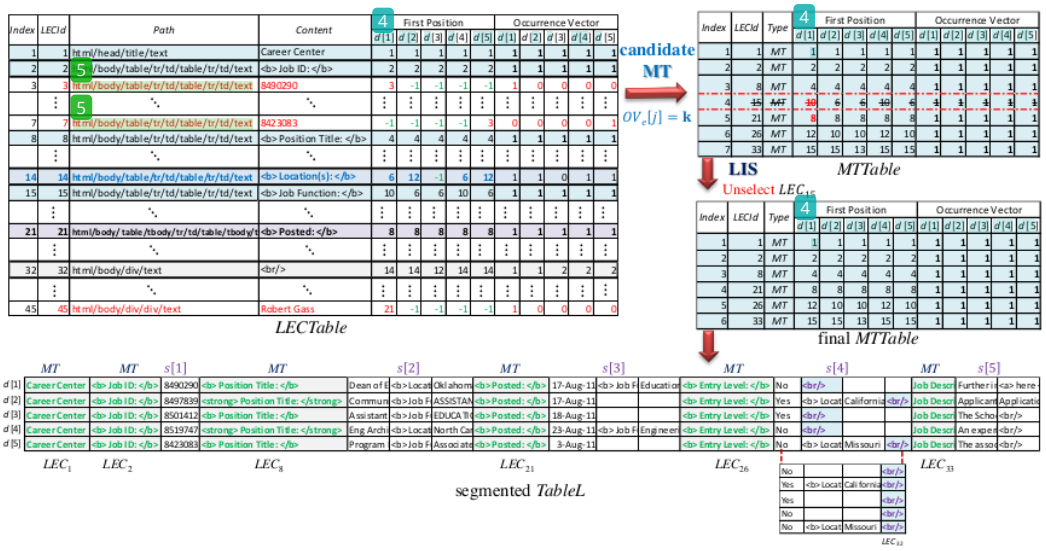


Fig. 7: Example of detecting and selecting MTs

4.4 Optional template (OT) detection

Next, we define optional template and conduct optional template detection for each segment in a way similar to MT detection but dealing with LECs with $OV[j] =$ **k** or 0. For these possible optional templates, we also apply LIS to filter inconsitent OTs.

Definition 5 (Candidate Optional Template) A candidate Optional Template (OT) is an LEC with the same occurrence count **k** (a positive integer) or null (0) in each $d[j]$, and the support of the LEC (i.e. ratio of non-null documents) is greater than or equal to a given threshold (θ_{OT}).

$$Supp(LEC_e) = \frac{\sum_{j=1}^{|D|} I(OV_e[j])}{|D|}, \qquad I(x) = \begin{cases} 1 \ , \ x > 0 \\ 0 \ , \ otherwise \end{cases} \tag{3}$$

The biggest difference between OT detection and MT detection is that the candidate OTs not selected by LIS will be added back to the $OTTable$. For each candidate OT_r that is not selected by LIS, we will find an index p to insert the removed OT_r such that $OT_r.FP[j]$ is between $OTTable[p].FP[j]$ and $OTTable[p+1].FP[j]$ for each $d[j]$ where OT_r occurs (i.e. $OT_r.FP[j] \neq -1$).

After mandatory template and optional template detection, then we combine all the detected MTs and OTs into $TemplateLEC$ table based on the segment sequence as shown in Fig. 8(a) and examine the validity of these templates again as described next. To save space, we use $TLEC[t]$ as a short hand for $TemplateLEC[t]$ in the following.

4.5 Removing false positive MTs

While recursive detecting MT can effectively recover templates that are filtered by LIS, not all detected MT are true templates. In addition, because of multi-ordering attribute-value pairs, some LECs could be separated in several segments resulting inconsistent order of LECs. For example in Fig. 8(a), LEC_{21} with $Content$ =" Posted: " is incorrectly recognized as a MT, which further separates LEC_{14} and LEC_{15} into different segments. Thus, the system tries to remove false positive MTs by detecting recurring OTs across segments to keep a consistent order. Formally, we define recurring OT as follows.

Definition 6 (Recurring OT) A recurring OT is an LEC which occurs in more than one segment and has implement occurrence vector, i.e. two LECs in the $TemplateLEC$ table such that $TLEC[i]$ and $TLEC[i']$ ($i < i'$) has the same $LECId$ and the summation of the occurrence vector is less than **1**, i.e. $TLEC[i].LECId = TLEC[i'].LECId$ and $TLEC[i].OV + TLEC[i'].OV \leq$ **1**.

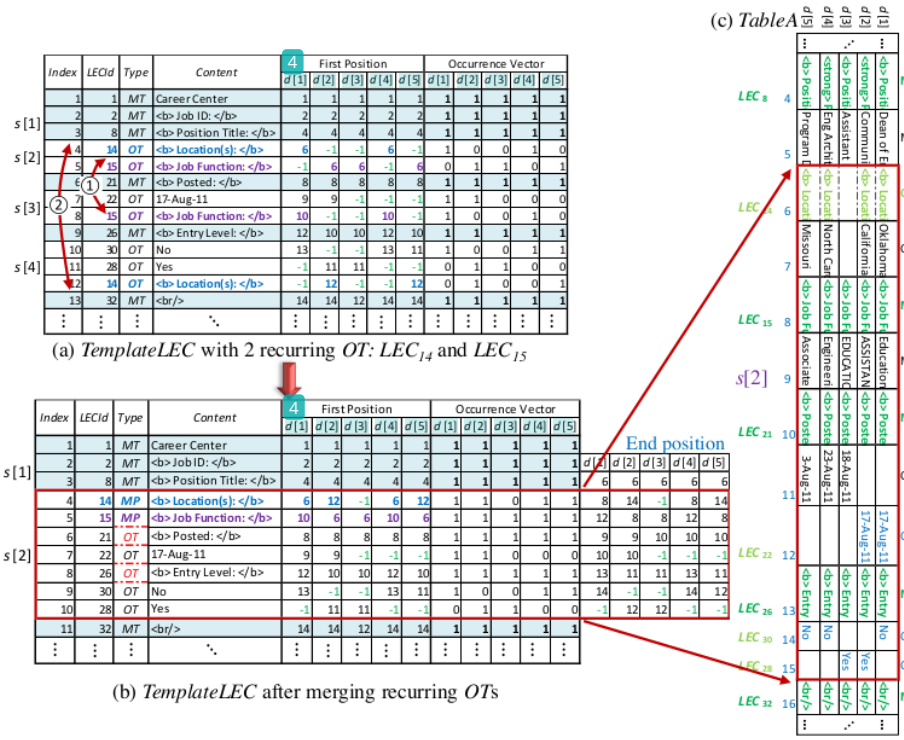


Fig. 8: Example of removing false positive MTs and merging recurring OTs to obtain template LECs [28]

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

1. Change the mandatory template $TLEC[c].Type$ to OT, $i < c < i'$.
2. Mark $TLEC[i].Type$ as MP.
3. Update the first position of $TLEC[i]$ by $max(TLEC[i].FP, TLEC[i'].FP)$ and the occurrence vector of $TLEC[i]$ by $TLEC[i].OV + TLEC[i'].OV$.
4. Remove $TLEC[i']$.

Example 3 Consider $TLEC[5]$ and $TLEC[8]$, both has $LECId$=15 ($Content$ ="Job Function:", see ① in Fig. 8), since $TLEC\ [5].OV + TLEC[8].OV = \mathbf{1}$, LEC_{15} is a recurring OT. Thus, the system merges $TLEC[5]$ with $TLEC[8]$ and changes the type of the $TLEC[6]$ from MT to OT. The first positions and occurrence counts of $TLEC[5]$ are updated to [10,6,6,10,6] and [1,1,1,1,1] respectively as shown in Fig. 8(b). Similarly, LEC_{14} is also a recurring OT. The system will merge $TLEC[4]$ and $TLEC[12]$ accordingly.

In the above example, LEC_{15} (the pruned MT in the LIS procedure for MT detection) and LEC_{14} (an OT in two separate segments with two MTs in-between) 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 a two pass alignment procedure as described below.

5 Alignment phase

After deteting template LECs, the next step is to align LECs that are not in the $TemplateLEC$ table. We show how to align the corresponding leaf nodes from $TableL$ into an aligned table called $TableA$. There are three kinds of templates including MT, OT, or MP in $TemplateLEC$ table. For each segment between two adjacent MTs: $TLEC[b]$ and $TLEC[b']$ (where $b < b'$), in the $TLEC$ table, if there is no $TLEC[t]$ ($b < t < b'$) of type MP in-between, e.g. $TLEC[2]$ and

$TLEC[3]$ in Fig. 8(b), the leaf nodes with index between $TLEC[b].FP[j]$ and $TLEC[b'].FP[j]$ for all $d[j]$ are aligned into $TableA$ as follows.

- **Phase 1 (Aligning leaf nodes belonging to TemplateLEC)**: For each template $TLEC[t]$ $(b \leq t \leq b')$, we align the corresponding leaf nodes $TLEC[t].FP[j]$-th leaf node in each $d[j]$ to the same column, and label it as an attribute column in $TableA$ and assign type to be MT if the occurrence vector is **1** or OT otherwise.
- **Phase 2 (Aligining leaf nodes not belonging to TemplateLEC)**: For the remaining leaf nodes with index between $TLEC[t].FP[j]$ and $TLEC[t+1].FP[j]$ for all $d[j]$, they are considered as value nodes and will be clustered respectively based on their similarity defined by a weighted average of *Path*, *IDSeq*, *ClassSeq*, and *TypeSet* with weight 0.3, 0.2, 0.2, and 0.3, respectively.

$$NodeSim(f1, f2) = Sim(f1.Path, f2.Path) \times \omega_P + Sim(f1.IDSeq, f2.IDSeq) \times \omega_I + Sim(f1.ClassSeq, f2.ClassSeq) \times \omega_C + Sim(f1.TypeSet, f2.TypeSet) \times \omega_T \quad (4)$$

Two leaf nodes $f1$ and $f2$ are considered in the same group if $NodeSim(f1, f2)$ is greater than or equal to $\theta_{Nsim} = 0.5$.

5.1 Multi-order AV-pair alignment

The challenging task here is how to align data nodes and deal with multi-order attribute-value pairs. While most templates and data are arranged in a consistent order across all pages, attribute-value pairs (AV-pair) can sometimes break the rule and render arbitrarily in different pages. For example, we can see that all leaf nodes of $TLEC[t]$ $(3 < t < 11)$ are unordered in all documents except for $d[3]$ as shown in the first positions of these $TLEC[t]$ in Fig. 8(b). Therefore, the procedure in phase 2 could not work properly.

Since each document has its own ordering of attribute-value pairs, we need to record such ordering for each document to align non-template leaf nodes. Assuming each $TLEC[t]$ represents some attribute and the following leaf nodes before the next template LEC will be the value of this attribute, we define the end position EP for each $TLEC[t]$ in $d[j]$ as follows:

$$TLEC[t].EP[j] = \min_i \{TLEC[i].FP[j] \mid b < i < b', TLEC[i].FP[j] > TLEC[t].FP[j]\} \quad (5)$$

Formally, when there exists $TLEC[t]$ of type MP between two MTs, e.g. LEC_{14} and LEC_{15} between two MTs $TLEC[3]$ and $TLEC[11]$ in Fig. 8(b), we record the minimum FP that is larger than $TLEC[t]$. $FP[j]$ $(b \leq t \leq b')$ in each document $d[j]$. During **Phase 2**, we collect leaf nodes with index between $TLEC[t].FP[j]$ and $TLEC[t]$ $.EP[j]$ in each $d[j]$ and cluster them based on their similarity. We then insert each cluster to a data column between two template column $TLEC[t]$ and $TLEC[t+1]$. Similarly, the leaf nodes with index between $TLEC[b]$ and the first template $TLEC[b+1]$ will be clustered respectively and inserted to a data column before all template/attribute columns. Finally, we label data columns to be mandatory data (MD) if the support is **1** or optional data (OD) otherwise.

Example 4 For the segment between two adjacent MTs $TLEC[3]$ and $TLEC[11]$, there exist two MPs, i.e. LEC_{14} and LEC_{15}, and several OTs. In **Phase 1**, leaf nodes corresponding to the same template LEC are aligned in the same column. For instance, the leaf nodes specified by $TLEC[4].FP = [6, 12, -1, 6, 12]$, i.e. $d[1][6]$, $d[2][12]$, $d[4][6]$, and $d[5][12]$ of $TableL$ in Fig. 6, are aligned in column 6 of $TableA$ in Fig. 8. Since $TLEC[5]$, $TLEC[6]$ and $TLEC[8]$ all have occurrence vector **1**, their type are changed to MT. As for $TLEC[4]$, which is of MP type with occurrence vector unequal to **1**, we change its type to OT.

In **Phase 2**, we first compute the end position for each $TLEC[t]$ $(b \leq t \leq b')$ in each document $d[j]$. For example, the first position for $TLEC[4]$ is [6,12,-1,6,12], thus the end position for $TLEC[4]$ is [8,14,-1,8,14].We consider the leaf nodes $d[1][7]$, $d[2][13]$, $d[4][7]$, and $d[5][13]$ as value nodes and cluster them into column 7 of $TableA$. As another example, the first position of $TLEC[6]$ is [8,8,8,8,8] and the end position for $TLEC[6]$ is [9,9,10, 10,10]. Since there are no leaf nodes between the first position and end position in $d[1]$ and $d[2]$, we could only cluster leaf nodes $d[3][9]$, $d[4][9]$, and $d[5][9]$. In other words, it is possible that no leaf nodes between two template LECs. The process repeats for each $TLEC[t]$ for $t < 11$.

5.2 Merging disjunctive/similar columns

In reality, the system may misalign data columns as template during OT detection (called false positive OT, e.g. column 12, 14, and 15 of $TableA$ in Fig. 8c). On the contrary, there are also false negative OT because of different occurrence count in documents or small support. These will result in incorrect alignment of leaf nodes and generate a sparse matrix with a large number of columns. To handle this problem, we rearrange

leaf nodes between two adjacent MTs in $TableA[b]$ and $TableA[b']$, by merging disjunctive columns, similar columns, and low density columns as follows.

- **Merge disjunctive columns**: for two adjacent columns t and $t+1$, if their occurrence vectors are disjunctive, i.e. $TableA[t].OV + TableA[t+1].OV \leq \mathbf{1}$, merge $TableA[t+1]$ into $TableA[t]$ and remove $TableA[t+1]$. For example, column 11 and 12 are disjunctive columns.
- **Merge similar columns**: for two adjacent columns t and $t+1$, if both columns have $colDensity$ smaller than the given θ_{Den}, and the column similarity between $TableA[t]$ and $TableA[t+1]$ is lower than the given θ_{Csim}, we merge $TableA[t]$ with $TableA[t+1]$ and delete $TableA[t+1]$.
- **Merge low density columns**: for contiguous optional columns $TableA[t] \sim TableA[t']$ (where $t < t'$) with $colDensity$ and $secDensity$ less than the given threshold θ_{Den}, we replace $TableA[t]$ by $\bigcup_{t \leq i \leq t'} TableA[i]$ and delete $TableA[i]$ for $t < i \leq t'$.

Here, the column density $colDensity(t)$, the column similarity between two columns $colSim(t1, t2)$, and the $secDensity$ for a section of contiguous columns are defined in Eq. (7), Eq. (6) and Eq. (8), respectively.

$$colDensity[t] = \frac{\#\,leaf\ nodes\ in\ column\ t}{|D|} \tag{6}$$

$$ColSim(t1,t2) = \big(Sim(t1.Path, t2.Path) + Sim(t1.IDSeq, t2.IDSeq) + Sim(t1.ClassSeq, t2.ClassSeq)\big)/3 \tag{7}$$

$$secDensity[t1,t2] = \frac{\#\,leaf\ nodes\ in\ column\ (t1 \sim t2)}{(t2 - t1 + 1) \times |D|} \tag{8}$$

6 Experiments

As mentioned in the introduction, we focus on attribute-value pairs data extraction from singleton pages. Since many algorithms have been proposed for data record extraction in list pages [20,24] and tables [6,17,21], so we do not take data set containing tables into account. We use three datasets: WEIR[3], TEX[4], and EXALG[5] for the following experiments. We exclude website containing tables and select only 22 from 41 websites (660 webpages) in TEX and 4 from 9 websites (152 singleton webpages) in EXALG. For WEIR, we use all 40 websites (24,038 webpages). Table 1 shows the averages number of web pages, leaf nodes for each website as well as the number of template and data columns, and golden answer of selected data items per website. We manually label an average of 208 template and 52 data columns based on the output of DCA for full schema evaluation.

We follow TEX [23] and define precision (P_c) and recall (R_c) for each data column c and average the precision and recall for the selected data columns (C) for evaluation of selected data items.

$$P_c = \frac{\#\,correct\ aligned\ leaf\ nodes\ in\ the\ extracted\ column}{\#\,leaf\ nodes\ in\ the\ extracted\ column} \tag{9}$$

$$R_c = \frac{\#\,correct\ leaf\ nodes\ in\ the\ extracted\ column}{\#\,leaf\ nodes\ in\ the\ golden\ answer\ column} \tag{10}$$

$$\bar{P} = \frac{\sum_{c=1}^{|C|} P_c}{|C|}, \bar{R} = \frac{\sum_{c=1}^{|C|} R_c}{|C|} \tag{11}$$

$$F_c = \frac{2 \times P_c \times R_c}{P_c + R_c}, \bar{F} = \frac{\sum_{c=1}^{|C|} F_c}{|C|} \tag{12}$$

For full schema evaluation, we count the number of correctly extracted columns (cc), where a column is considered correctly extracted if $R_c \geq 0.85$. By dividing ec by the number of data columns in $TableA$ or the number of golden data columns gc, we obtain full schema precision (P_F) and full schema recall (R_F), respectively.

$$P_F = \frac{cc}{ec}, R_F = \frac{cc}{gc}, F_F = \frac{2P_F R_F}{P_F + R_F} \tag{13}$$

For the following experiments, we set default threshold of path similarity θ_{Path} to 0.8, OT θ_{OT} to 0.3, leaf node similarity θ_{Nsim} to 0.7, and section density θ_{Den} to 0.7.

6.1 Performance comparison

First, we compare DCA with RoadRunner, WEIR, and TEX on the selected data columns of WEIR dataset. We run TEX application on WEIR dataset, however, TEX can deal with only 28 websites of WEIR dataset (a total 40 websites). The statistics for RoadRunner and WEIR are obtained from [4]. As shown in Fig. 9, DCA presents the best average performance ($P = 0.99$, $R = 0.93$, $F = 0.96$) for four categories in books, stock quotes, video games, and soccer players, followed by WEIR with average performance ($P = 0.92$, $R = 0.85$, $F = 0.88$).

Next, we evaluate DCA on the selected data columns of TEX-22 dataset with RoadRunner, FivaTech and TEX using the statistics from [23]. As shown in Fig.

[3] http://www.dia.uniroma3.it/db/weir
[4] http://www.tdg-seville.info/Hassan/TEX
[5] http://infolab.stanford.edu/arvind/extract/

Table 1: Data description

Dataset	#Pages per Site	#Leaf Nodes	#Template Columns	#Data Columns	#Golden Answer	Data Density
EXALG-4	38	213	140	41	5	0.910
TEX-22	30	292	215	48	5	0.885
WEIR-40	601	439	267	66	6	0.915
Average	223	315	208	52	6	0.903

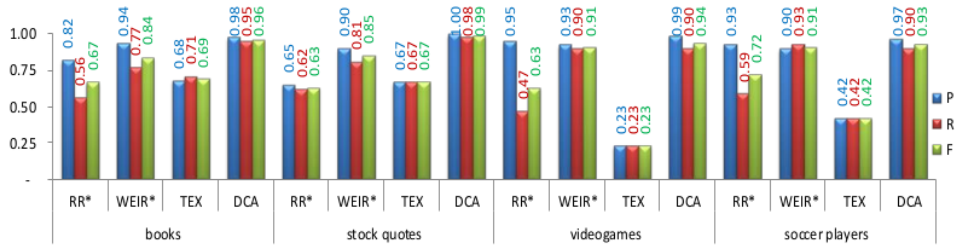


Fig. 9: Performance comparison of four methods on selected data columns of WEIR dataset [4]

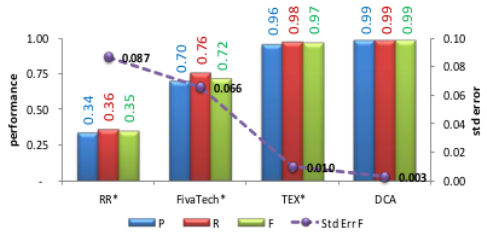


Fig. 10: Performance comparison of four methods on selected items of TEX-22 dataset [23]

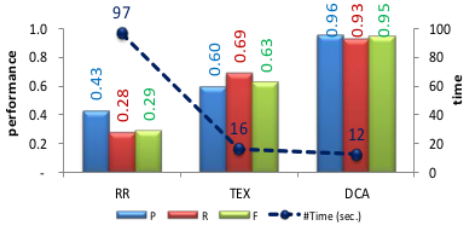


Fig. 11: Performance comparison of full schema on TEX-22 and EXALG-4 datasets

10. DCA presents the best performance ($P = 0.99$, $R = 0.99$, $F = 0.99$) followed by TEX performance ($P = 0.96$, $R = 0.98$, $F = 0.97$). Note that FiVaTech deals with only 22 websites and RoadRunner can process only 11 web sites as reported in [23].

Finally we consider all data columns of TEX-22 and EXALG-4 datasets and show the number of data columns generated and processing time per website as shown in Table 2. The performances on all data columns is degraded ($P = 0.60$, $R = 0.69$, $F = 0.63$). The average number of data columns for the golden answer is 47 columns per website and DCA generates the closest column number (46). RoadRunner merged more columns and produced the smallest number of data columns. On the contrary, TEX suffered from false positive data attributes and produced the highest number of data columns, resulting low precision.

In terms of efficiency, the processing time of DCA (12 seconds) is the fastest compared with RoadRunner (97 seconds) and TEX (16 seconds) as shown in Figure 11. We attribute this advantage to DCA's divide-and-conquer mechanism and the use of leaf nodes as processing units (thus reducing the number of units processed).

6.2 Sensitivity analysis

There are four major parameters in the proposed DCA algorithm including path similarity of two leaf nodes θ_{Path} (for LEC generation), occurrence vector support θ_{OT} (for OT detection), node similarity θ_{Nsim} (for data node clustering), and column/section density θ_{Den} To see how these parameters affect the performance and determine the thresholds, we conduct sensitivity analysis based on EXALG-4 and TEX-22 datasets.

Table 2: Performance comparison of full schema on TEX-22 and EXALG-4 datasets [2]

ID	Webpage	GA	DCA				TEX				RR			
			#EC	P_F	R_F	F_F	#EC	P_F	R_F	F_F	#EC	P_F	R_F	F_F
T01	www.abebooks.com	44	43	0.97	0.98	0.97	52	0.63	0.59	0.61	-	-	-	-
T02	www.awesomebooks.com	21	21	1.00	1.00	1.00	37	0.64	0.76	0.70	20	1.00	0.95	0.98
T03	www.manybooks.net	39	37	0.97	0.93	0.95	44	0.34	0.33	0.34	-	-	-	-
T04	www.autotrader.com	117	113	0.98	0.96	0.97	156	0.55	0.54	0.54	-	-	-	-
T05	www.carmax.com	101	100	0.93	0.87	0.90	119	0.35	0.36	0.35	8	0.50	0.04	0.07
T06	www.classiccarsforsale.co.uk	51	43	0.98	0.84	0.90	93	0.52	0.84	0.65	-	-	-	-
T07	www.internetautoguide.com	92	90	0.96	0.93	0.94	101	0.67	0.65	0.66	-	-	-	-
T08	www.mbendi.com	13	13	1.00	1.00	1.00	13	0.61	0.85	0.71	13	1.00	1.00	1.00
T09	www.rdlearning.org.uk	15	16	0.86	0.91	0.88	11	0.77	0.87	0.81	1	1.00	0.07	0.13
T10	extapps.ama-assn.org	21	19	0.83	0.75	0.79	24	0.70	0.67	0.68	6	0.67	0.19	0.30
T11	www.drscore.com	31	30	0.97	0.96	0.96	64	0.49	0.58	0.53	-	-	-	-
T12	www.steadyhealth.com	106	106	0.95	0.93	0.94	54	0.65	0.52	0.58	3	1.00	0.03	0.05
T13	careers.insightintodiversity.com	14	14	1.00	1.00	1.00	52	0.12	0.43	0.19	4	1.00	0.07	0.13
T14	www.6figurejobs.com	48	48	0.97	0.97	0.97	55	0.70	0.83	0.76	-	-	-	-
T15	www.careerbuilder.com	38	37	0.97	0.93	0.95	35	0.78	0.74	0.76	-	-	-	-
T16	www.jobofmine.com	13	13	1.00	1.00	1.00	22	0.60	0.92	0.73	12	1.00	0.92	0.96
T17	www.albaniam.com	17	17	1.00	1.00	1.00	17	1.00	0.94	0.97	17	1.00	1.00	1.00
T18	www.allmovie.com	70	67	0.96	0.91	0.94	66	0.70	0.73	0.71	-	-	-	-
T19	www.citwf.com	36	35	0.99	0.96	0.97	21	0.89	0.92	0.90	35	1.00	0.97	0.99
T20	www.disneymovieslist.com	30	30	0.97	0.95	0.96	53	0.44	0.71	0.54	-	-	-	-
T21	www.emax.com	60	60	0.98	0.98	0.98	83	0.54	0.70	0.61	-	-	-	-
T22	www.atpworldtour.com	78	78	0.96	0.96	0.96	104	0.50	0.64	0.56	-	-	-	-
E01	teams.uefa.com	14	14	1.00	1.00	1.00	18	1.00	1.00	1.00	14	1.00	1.00	1.00
E02	www.ausopen.com	35	35	0.97	1.00	0.98	54	0.55	0.63	0.59	35	1.00	1.00	1.00
E03	www.ebay.com	46	47	0.86	0.76	0.81	81	0.38	0.54	0.45	-	-	-	-
E04	www.netflix.com	70	70	0.97	0.81	0.88	95	0.42	0.60	0.49	-	-	-	-
	Avg.	47	46	0.96	0.93	0.95	59	0.60	0.69	0.63	8	0.43	0.28	0.29

*The aligned output of these websites can be obtained from https://sites.google.com/site/nculab/project/WDEMS/dca [2]

Fig.12a shows the distribution of path similarity and the performance with the varying (from 0.1 to 0.9) path similarity threshold, θ_{Path}. As we can see, most (83.6%) leaf nodes with the same content have path similarity higher than 0.9, however, there are still leaf nodes with very different path, which we consider as incorrect landmark. Second, the F1 performance ranges varies between 0.94 and 0.95, showing that using leaf nodes with the same text content is a good choice for LEC even without path similarity. However, increasing θ_{Path} will exclude false positive MTs, thus reducing the number of divide-and-conquers (DC) iterations (from 56 to 48). Therefore, We define $\theta_{Path} = 0.8$ for two leaf nodes with the same text content to be considered the same LECs, i.e $Eq.(2) \geq 0.8$.

Fig. 12b shows the distribution of LEC support and the performance w.r.t $varying \theta_{OT}$ threshold. Here, most (93.9%) LECs have support less than 0.1 since most data leaf nodes form an LEC with 1 occurrence. Increasing θ_{OT} will reduce the number of optional templates OTs. DCA achieves the best performance when $\theta_{OT} = 0.3$ and remains good (F1 > 0.92) from 0.1 to 0.6. When the θ_{OT} threshold is set too larger, we lose true positive OTs. Therefore we define $\theta_{OT} = 0.3$ for an LEC to be considered an optional template, i.e $Eq.(3) \geq 0.3$.

Fig. 12c shows the distribution of leaf node similarity for data node clustering during the alignment phase. As we can see, most (85.4%) leaf node pairs have a similarity higher than or equal to 0.7. Second, for different similarity thresholds, the performance of the proposed DCA has little change (between 0.94 and 0.95 in F1), showing the robustness of the proposed algorithm. The best performance is achieved when $\theta_{Nsim} = 0.7$ in our experiment. Therefore, we set $Eq.(4) \geq 0.7$ for two leaf nodes to be clustered in the same column.

Finally, Fig. 12d shows the distribution of section density for contiguous columns between two MTs. The distribution is quite average between 0 and 1. Section density plays an important role in the DCA performances. The best performance ($P = 0.96$, $R = 0.93$, $F = 0.95$) is achieved when $\theta_{Den} = 0.7$ as shown in Fig. 12d. Without this merging mechanism, the F1 performance could only be 0.75.

In summary, support threshold θ_{OT} and section density threshold θ_{Den} are the two major parameters that could affect the performances of DCA. From the above experiments, DCA obtains good performance (F1 measure between 0.92 to 0.95) for $0.1 \leq \theta_{OT} \leq 0.6$ and $0.6 \leq \theta_{Den} \leq 0.8$. Furthermore, only path threshold θ_{Path} has an impact on the number of divide-and-conquer iterations.

7 Conclusions and Future Work

In this paper, we present an unsupervised approach for web data extraction on singleton pages. We define landmark equivalence class (LEC) as leaf nodes with the same text content and similar paths and use them for template mining. (In comparison, WEIR requires leaf nodes to have the same path and sets 40% support for optional templates.) We then prioritize the discovery of templates in order of mandatory and optional via occurrence vectors and ensure the consistency of such templates through LIS (longest increasing sequence) algorithm. Then, the discovered MTs divide the input table into segments for recursive processing.

Since there might be false positive templates during the template mining phase, we design the merging of recurring OTs to remove such MTs and mark them for multi-order AV-pair alignment via end position to locate the value leaf nodes for each template attribute. Finally, DCA adopts a similarity measure for data node clustering based on Path, IDSeq, ClassSeq, and TypeSet. Therefore, the system can decide how to merge leaf nodes to get final result.

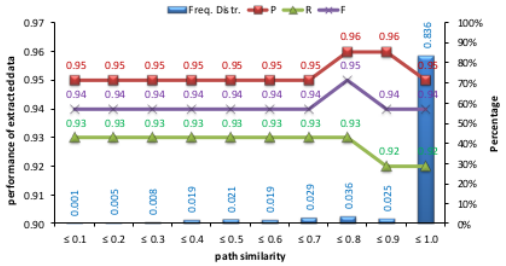
We conducted experiments on real-world datasets from WEIR, TEX-22 and EXALG-4. Overall, DCA outperforms RoadRunner, TEX, and WEIR not only on the selected data items but also on complete data columns (with 0.95 F-measure) in terms of full schema evaluation (compared with 0.63 F-measure for TEX and 0.29 F-measure for RoadRunner). In addition, we conduct sensitivity analysis to show the robustness of the DCA algorithm with various parameter thresholds for path similarity, OT support, leaf node similarity, and section density.

For future work, we will design dynamic encoding for leaf node abstraction to enhance the alignment performance on data columns. Furthermore, we will extend the proposed approach to handle list and table extraction inside singleton pages. Finally, we will implement a wrapper generation module for efficient extraction on testing pages.

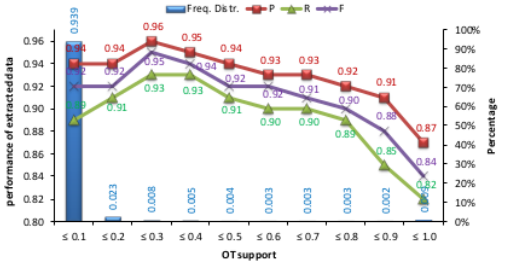
Acknowledgements This research is supported by Ministry of Science and Technology Taiwan, under grant MOST 105-2628-E-008-004-MY2.

References

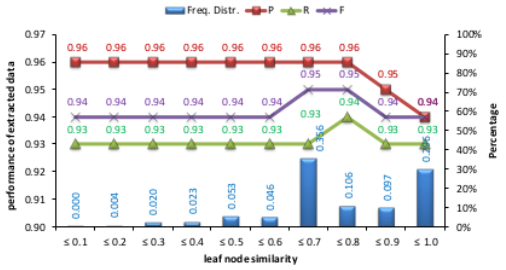
1. Arasu A, Molina HG (2003) Extracting Structured Data from Web Pages. In: SIGMOD, pp 337-348
2. Augenstein I, Maynard D, Ciravegna F (2016) Distantly Supervised Web Relation Extraction for Knowledge Base Population. Semantic Web, 7(4), pp 335-349



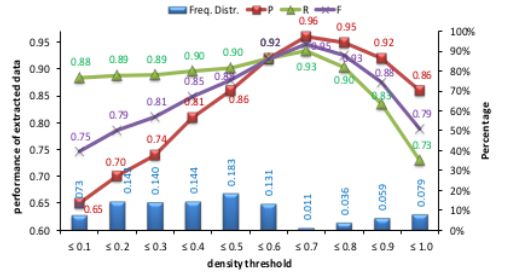
(a) *Path* threshold tuning



(b) *OT* threshold tuning



(c) Leaf node similarity threshold tuning



(d) Column/section density threshold tuning

Fig. 12: Sensitivity analysis for parameter tuning

3. Bing L, Lam W, Wong TL (2013) Wikipedia Entity Expansion and Attribute Extraction from the Web Using Semi-supervised Learning. In: Web search and data mining, pp 567-576
4. Bronzi M, Crescenzi V, Merialdo P, Papotti P (2013) Extraction and Integration of Partially Overlapping Web Sources. In: VLDB, 6(10), pp 805-816
5. Chang CH, Kayed M, Girgis MR, Shaalan KF (2010) A Survey of Web Information Extraction Systems. IEEE Transactions on Knowledge and Data Engineering, 18(10), pp 1411-1428
6. Chu X, He Y, Chakrabarti K, Ganjam K (2015) Tegra: Table extraction by global record alignment. In: SIGMOD, pp 1713-1728
7. Cortez E, da Silva AS, Gonçalves MA, de Moura ES (2010) Ondux: On-demand Unsupervised Learning for Information Extraction. In: SIGMOD, pp 807-818
8. Cortez E, Oliveira D, da Silva AS et al (2011) Joint Unsupervised Structure Discovery and Information Extraction. In: SIGMOD, pp 541-552
9. Crescenzi V, Mecca G (2004) Automatic Information Extraction from Large Websites. Journal of the ACM, 51(5), pp 731-779
10. Crescenzi V, Merialdo P, Qiu D (2013) Alfred: Crowd Assisted Data Extraction. In: WWW, pp 297-300
11. Dalvi BB, Cohen WW, Callan J (2012) Websets: Extracting Sets of Entities from the Web Using Unsupervised Information Extraction. In: Web search and data mining, pp 243-252
12. Dhillon PS, Sellamanickam S, Selvaraj SK (2011) Semi-supervised Multi-task Learning of Structured Prediction Models for Web Information Extraction. In: Information and knowledge management, pp 957-966
13. Ferrara E, De Meo P, Fiumara G, Baumgartner R (2014) Web data extraction, applications and techniques: A survey. Knowledge-Based Systems, 70, pp 301-323
14. Fossati M, Dorigatti E, Giuliano C (2017) N-ary Relation Extraction for Simultaneous T-Box and A-Box Knowledge Base Augmentation. Semantic Web, pp 1-27
15. Hao Q, Cai R, Pang Y, Zhang L (2011) From one Tree to a Forest: a Unified Solution for Structured Web Data Extraction. In: SIGIR, pp 775-784
16. He B, Patel M, Zhang Z, Chang KCC (2007) Accessing the Deep Web. Communications of the ACM, 50(5), pp 94-101
17. Ibrahim Y, Riedewald M, Weikum G (2016) Making sense of entities and quantities in web tables. In: CIKM, pp 1703-1712
18. Jou C (2015) Semantics-Assisted Deep Web Query Interface Classification. In: Computer Science & Software Engineering, pp 70-78
19. Kayed M, Chang CH (2010) FiVaTech: Page-level Web Data Extraction from Template Pages. IEEE Transactions on Knowledge and Data Engineering, 22(2), pp 249-263
20. Lu Y, He H, Zhao H et al (2013) Annotating Search Results from Web Databases. IEEE Transactions on Knowledge and Data Engineering, 25(3), pp 514-527
21. Sarawagi S, Chakrabarti S (2014) Open-domain Quantity Queries on Web Tables: Annotation, Response, and Consensus Models. In: SIGKDD, pp 711-720
22. Sequeda JF, Arenas M, Miranker DP (2012) On Directly Mapping Relational Databases to RDF and OWL. In: WWW, pp 649-658
23. Sleiman HA, Corchuelo R (2013) TEX: An Efficient and Effective Unsupervised Web Information Extractor. Knowledge-Based Systems, 39, pp 109-123

24. Su W, Wang J, Lochovsky FH, Liu Y (2012) Combining Tag and Jouvalue Similarity for Data Extraction and Alignment. IEEE Transactions on Knowledge and Data Engineering, 24(7), pp 1186-1200
25. Vieira K, da Costa Carvalho AL, Berlt K at al (2009) On Finding Templates on Web Collections. WWW Journal, 12(2), pp 171-211
26. Weninger T, Hsu WH, Han J (2010) CETR: Content Extraction via Tag Ratios. In: WWW, pp 971-980
27. Wu S, Liu J, Fan J (2015) Automatic Web Content Extraction by Combination of Learning and Grouping. In: WWW, pp 1264-1274 (ACM)
28. Yuliana OY, Chang CH (2016) Aligning singleton pages for Full Schema Induction. In: TAAI, pp 220-227
29. Zhai Y, Liu B (2006) Structured Data Extraction from the Web Based on Partial Tree Alignment. IEEE Transactions on Knowledge and Data Engineering, 18(12), pp 1614-1628
30. Zheng X, Gu Y, Li Y (2012) Data Extraction from Web Pages Based on Structural-Semantic Entropy. In: WWW, pp 93-102

A novel alignment algorithm for effective web data extraction from singleton-item pages

ORIGINALITY REPORT

69%	60%	69%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	link.springer.com Internet Source	34%
2	Oviliani Yenty Yuliana, Chia-Hui Chang. "A novel alignment algorithm for effective web data extraction from singleton-item pages", Applied Intelligence, 2018 Publication	33%
3	repository.petra.ac.id Internet Source	1%
4	ns2.thinkmind.org Internet Source	1%
5	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	1%

Exclude quotes	On	Exclude matches	< 1%
Exclude bibliography	Off		