Extracting General Lists from Web Documents: A Hybrid Approach

Fabio Fumarola[1], Tim Weninger[2], Rick Barber[2],
Donato Malerba[1], and Jiawei Han[2]

[1] Dipartimento di Informatica, Università degli Studi di Bari
"Aldo Moro", Bari, Italy
{ffumarola,malerba}@di.uniba.it
[2] Computer Science Department, University of Illinois at Urbana-Champaign,
Urbana-Champaign, IL
{weninge1,hanj,barber5}@uiuc.edu

Abstract. The problem of extracting structured data (*i.e.* lists, record sets, tables, etc.) from the Web has been traditionally approached by taking into account either the underlying markup structure of a Web page or the visual structure of the Web page. However, empirical results show that considering the HTML structure and visual cues of a Web page independently do not generalize well. We propose a new hybrid method to extract general lists from the Web. It employs both general assumptions on the visual rendering of lists, and the structural representation of items contained in them. We show that our method significantly outperforms existing methods across a varied Web corpus.

Keywords: Web lists, Web mining, Web information integration.

1 Introduction

The extraction of lists from the Web is useful in a variety of Web mining tasks, such as annotating relationships on the Web, discovering parallel hyperlinks, enhancing named entity recognition, disambiguation, and reconciliation. The many potential applications have also attracted large companies, such as Google, which has made publicly available the service Google Sets to generate lists from a small number of examples by using the Web as a big pool of data [13].

Several methods have been proposed for the task of extracting information embedded in lists on the Web. Most of them rely on the underlying HTML markup and corresponding DOM structure of a Web page [13,1,9,12,5,3,14,7,16,17]. Unfortunately, HTML was initially designed for rendering purposes and not for information structuring (like XML). As a result, a list can be rendered in several ways in HTML, and it is difficult to find an HTML-only tool that is sufficiently robust to extract *general* lists from the Web.

Another class of methods is based on the rendering of an HTML page [2,4,6,10,11]. These methods are likewise inadequate for general list extraction, since they tend to focus on specific aspects, such as extracting tables

K.G. Mehrotra et al. (Eds.): IEA/AIE 2011, Part I, LNAI 6703, pp. 285–294, 2011.

where each data record contains a link to a detail page [6], or discovering tables rendered from Web databases [10] (deep web pages) like Amazon.com. Due to the restricted notion of what constitutes a table on the web, these visual-based methods are not likely to effectively extract lists from the Web in the general case.

This work aims to overcome the limitations of previous works for what concerns the generality of extracted lists. This is obtained by combining several visual and structural features of Web lists. We start from the observation that lists usually contain items which are similar in type or in content. For example, the Web page shown in Figure 1a) shows eight separate lists. Looking closely at it, we can infer that the individual items in each list: 1) are visually aligned (horizontally or vertically), and 2) share a similar structure.

The proposed method, called HyLiEn (**Hy**brid approach for automatic **Li**st discovery and **E**xtractio**n** on the Web), *automatically* discovers and extracts *general* lists on the Web, by using both information on the visual alignment of list items, and non-visual information such as the DOM structure of visually aligned items. HyLiEn uses the CSS2 visual box model to segment a Web page into a number of *boxes*, each of which has a position and size, and can either contain content (*i.e.*, text or images) or more boxes. Starting from the box representing the entire Web page, HyLiEn recursively considers inner boxes, and then extracts list boxes which are visually aligned and structurally similar to other boxes. A few intuitive, descriptive, visual cues in the Web page are used to generate candidate lists, which are subsequently pruned with a test for structural similarity in the DOM tree. As shown in this paper, HyLiEn significantly outperforms existing extraction approaches in specific and general cases.

The paper is organized as follows. Section 2 presents the Web page layout model. Sections 3 and 4 explain the methodology used by our hybrid approach for visual candidate generation and Dom-Tree pruning, respectively. Section 5 provides the methodology of our hybrid approach. The experimental results are

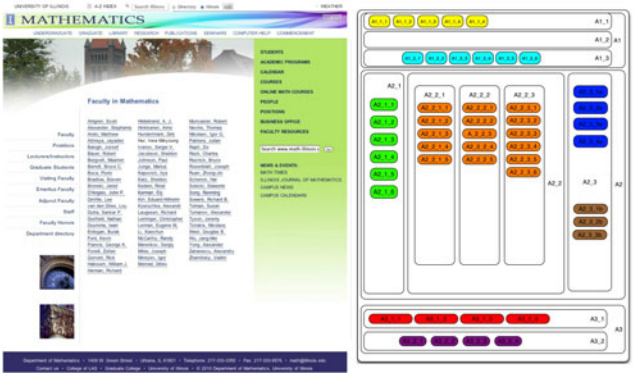

Fig. 1. a) Web page of Mathematics Department of University of Illinois. b) The boxes structure of the Web page.

reported in Section 6. Section 7 summarizes our contributions and concludes the paper.

2 Web Page Layout

When an HTML document is rendered in a Web browser, the CSS2 visual formatting model [8] represents the elements of the document by rectangular boxes that are laid out one after the other or nested inside each other. By associating the document with a coordinate system whose origin is at the top-left corner, the spatial position of each text/image element on the Web page is fully determined by both the coordinates (x, y) of the top-left corner of its corresponding box, and the box's height and width. The spatial positions of all text/image elements in a Web page define the *Web page layout*.

Each Web page layout has a tree structure, called *rendered box tree*, which reflects the hierarchical organization of HTML tags in the Web page. More precisely, let $\mathcal{H}$ be the set of occurrences of HTML tags in a Web page p, $\mathcal{B}$ the set of the rendered boxes in p, and $map : \mathcal{H} \rightarrow \mathcal{B}$ a bijective function which associates each $h \in \mathcal{H}$ to a box $b \in \mathcal{B}$. The markup text in p is expressed by a rooted ordered tree, where the root is the unique node which denotes the whole document, the other internal nodes are labeled by tags, and the leaves are labeled by the contents of the document or the attributes of the tags. The bijective map defines an isomorphic tree structure on $\mathcal{B}$, so that each box $b \in \mathcal{B}$ can be associated with a parent box $u \in \mathcal{B}$ and a set $CB = \{b_1, b_2, \cdots, b_n\}$ of child boxes. Henceforth, a box b will be formally defined as a 4-tuple $\langle n, u, CB, P \rangle$, where $n = map^{-1}(b)$ and $P = (x, y)$ is the spatial position of b, while a rendered box tree T will be formally defined as a directed acyclic graph $\mathcal{T} = \{\mathcal{B}, \mathcal{V}, r\}$,

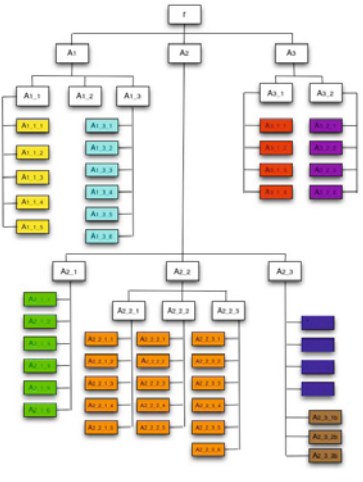

Fig. 2. The box tree structure of Illinois Mathematics Web Page

where $\mathcal{V} \subset \mathcal{B} \times \mathcal{B}$ is the set of directed edges between the *boxes*, and $r \in B$ is the root box representing the whole page. An example of a rendered box tree is shown in Figure 2. The leaves of a rendered box tree are the non-breakable boxes that do not include other boxes, and they represent the minimum units of the Web page. Two properties can be reported for the rendered box trees.

Property 1. If box a is contained in box b, then b is an ancestor of a in the rendered box tree.

Property 2. If a and b are not related under property 1, then they do not overlap visually on the page.

3 Candidate Generation with Visual Features

Even though there is no strictly followed set of best practices for Web design, different information-carrying structures within Web page often have distinct visual characteristics which hold in general. In this work, we make the following basic assumption about the visual characteristics of a list:

Definition 1. *A list candidate $l = \{l_1, l_2, \ldots, l_n\}$ on a rendered Web page consists of a set of vertically and/or horizontally aligned boxes.*

We describe in [15] that this assumption alone is sufficient to outperform all existing list extraction methods. Indeed, it seems that a human user might find these alignment features to be most important in identifying lists manually. Therefore, with this assumption we can generate list candidates by comparing the boxes of a rendered Web page. Unfortunately, this assumption by itself does not cover Web pages such as the one in Figure 1 where the list elements are not necessarily all mutually aligned. For instance, all of the orange boxes inside Box A_{2_2} correspond to a single list in the page, but there are many pairs of elements in this list which are not visually aligned. Therefore, inside the region of A_{2_2}, the first step of our method will find three vertical list candidates and many horizontal list candidates based on our definition, and there will be some degree of overlap between these lists.

Definition 2. *Two lists l and l' are* related *($l \sim l'$) if they have an element in common. A set of lists $\mathcal{S}$ is a* tiled structure if for every list $l \in \mathcal{S}$ there exists at least one other list $l' \in \mathcal{S}$ such that $l \sim l'$ and $l \neq l'$. Lists in a tiled structure are called *tiled lists*.

Three tiled lists, namely $A_{2\_2\_1}$, $A_{2\_2\_2}$ and $A_{2\_2\_3}$, are shown in Figure 1b). As explained below, the notion of tiled list is useful to handle more problematic cases by merging the individual lists of a tiled structure into a single tiled list.

4 Pruning with DOM-Tree Features

Several papers exploit the DOM-structure of Web pages to generate wrappers, identify data records, and discover lists and tables [13,1,9,12,5,3,14,7,16,17]. We

Algorithm 1: `HybridListExtractor`

input : Web site S, level of similarity α, max DOM-nodes β
output: set of lists L

```
RenderedBoxTree T(S);
Queue Q;
Q.add(T.getRootBox());
while !Q.isEmpty() do
    Box b = Q.top();
    list candidates = b.getChildren();
    list aligned = getVisAligned(candidates);
    Q.addAll(getNotAligned (candidates));
    Q.addAll(getStructNotAligned (candidates, α, β));
    aligned = getStructAligned (aligned, α, β);
    L.add(aligned);
return L;
```

notice that, even if purely DOM-centric approaches fail in the general list finding problem, the DOM-tree could still be a valuable resource for the comparison of visually aligned boxes. In a list on a Web page, we hypothesize that the DOM-subtrees corresponding to the elements of the list must satisfy a structural similarity measure ($structSim$) to within a certain threshold α and that the subtrees not have a number of DOM-nodes ($numNodes$) greater than β.

Definition 3. *A candidate list $l = \{l_1, l_2, \ldots, l_n\}$ is a genuine list if and only if for each pair (l_i, l_j), $i \neq j$, $structSim\ (l_i, l_j) \leq \alpha$, $numNodes(l_i) \leq \beta$ and $numNodes(l_j) \leq \beta$.*

This DOM-structural assumption serves to prune false positives from the candidate list set. The assumption we make here is shared with most other DOM-centric structure mining algorithms, and we use it to determine whether the visual alignment of a certain boxes can be regarded as a real list or whether the candidate list should be discarded. Specifically, the α and β parameters are essentially the same as the K and T thresholds from MDR [9] and DEPTA [16,17], and the α and C thresholds from Tag Path Clustering [12].

5 Visual-Structural Method

The input to our algorithm is a set of unlabeled Web pages containing lists. For each Web page, Algorithm 1 is called to extract Web lists. We use the open source library *CSSBox*[1] to render the pages.

The actual rendered box tree of a Web page could contain hundreds or thousands of boxes. Enumerating and matching all of the boxes in search of lists of arbitrary size would take time exponential in the number of boxes if we used a

[1] `http://cssbox.sourceforge.net`

Algorithm 2: `getVisAligned`

```
input  : list candidates
output: list visAligned

Box head = candidates[0];
list visAligned;
for (i = 1; i < candidates.length; i++) do
    Box tail = candidates[i];
    if (head.x == tail.x || head.y == tail.y) then
        visAligned.add(tail);
if visAligned.length > 1 then
    visAligned.add(head);
return visAligned;
```

brute force approach. To avoid this, our proposed method explores the space of the boxes in a top-down manner (*i.e.*, from the root to boxes that represent the elements of a list) using the edges of the rendered box tree ($\mathcal{V}$). This makes our method more efficient with respect to those reported in the literature.

Starting from the root box r, the *rendered box tree* is explored. Using a *Queue*, a breadth first search over children boxes is implemented. Each time a box b is retrieved from the *Queue*, all the children boxes of b are tested for visual and structural alignment using Algorithms 2 and 3, thereby generating candidate and genuine lists among the children of b. All the boxes which are not found to be visually or structurally aligned are enqueued in the *Queue*, while the tested boxes are added the resultset of the Web page. This process ensures that the search does not need to explore each atomic element of a Web page, and thus makes the search bounded on the complexity of the actual lists in the Web page.

Algorithm 2 uses the visual information of boxes to generate candidate lists which are horizontally or vertically aligned. To facilitate comprehension of the approach, we present a generalized version of the method where the vertical and horizontal alignments are evaluated together. However, in the actual implementation of the method these features are considered separately; this enables the method to discover both the horizontal, vertical and tiled lists on the Web page.

Algorithm 3 prunes false positive candidate lists. The first element of the visually aligned candidate list is used as an element for the structural test. Each time the *tail* candidate is found to be structurally similar to the *head*, the *tail* is added to the result list. At the end, if the length of the result list is greater than one, the head is added. If none of the boxes are found to be structurally similar, an empty list is returned.

To check if two boxes are structurally similar, Algorithm 4 exploits the DOM-tree assumption described in Def. 3. It works with any sensible tree similarity measure. In our experiments we use a simple string representation of the corresponding tag subtree of the boxes being compared for our similarity measurement.

Algorithm 3: `getStructAligned`

input : list *candidates*, min. similarity α, max. tag size β
output: list *structAligned*

```
Box head = candidates[0];
list structAligned;
for (i = 1; i < candidates.length; i++) do
    Box tail = candidates[i];
    if getStructSim (head, tail, β) ≤ α then
        structAligned.add(tail);
if structAligned.length > 1 then
    structAligned.add(head);
return structAligned;
```

Algorithm 4: `getStructSim`

input : Box a, b, max. tag size β
output: double $simValue$

```
TagTree tA = a.getTagTree();
TagTree tB = b.getTagTree();
double simValue;
if (tA.length ≥ β || tB.length ≥ β) then
    return MAXDouble;
simValue = Distance(tA, tB);
return simValue;
```

At the end of the computation, Algorithm 1 returns the collection of all the lists extracted from the Web page. A post-processing step is finally applied to deal with tiled structures. Tiled lists are not directly extracted by this algorithm. Based on Section 2, each list discovered is contained in a box b. Considering the position P of these boxes, we can recognize tiled lists. We do this as a post-process by: 1) identifying the boxes which are vertically aligned, and 2) checking if, the element lists contained in that boxes are visually and structurally aligned. Using this simple heuristic we are able to identify tiled lists and update the result set accordingly.

6 Experiments

We showed in [15] that implementing a method that uses assumption in Def. 1 is sufficient to outperform all existing list extraction methods. Thus we tested HiLiEn on a dataset used to validate the Web Tables discovery in VENTex [4]. This dataset contains Web pages saved by WebPageDump[2] including the Web page after the "x-tagging", the coordinates of all relevant Visualized Words (VENs), and the manually determined ground truth. From the first 100 pages of

[2] `http://www.dbai.tuwien.ac.at/user/pollak/webpagedump/`

the original dataset we manually extracted and verified 224 tables, with a total number of 6146 data records. We can use this dataset as test set for our method because we regard tables on the Web to be in the set of lists on the Web, that is, a table is a special type of list. This dataset was created by asking students taking a class in Web information extraction at Vienna University of Technology to provide a random selection of Web tables. This, according to Gatterbauer *et al.*[4], was done to eliminate the possible influence of the Web page selection on the results. We use the generality advantage of this dataset to show that also our method is robust and the results are not biased from the selected test set.

HyLiEn returns a text and visual representation of the results. The former consists of a collection of all the discovered lists, where each element is represented by its HTML tag structure and its inner text. The latter is a png image, where all the discovered lists are highlighted with random colors.

We compared HyLiEn to VENTex, which returns an XML representation of the frames discovered. Because of the differences in output of the two methods, we erred on the side of leniency in most questionable cases. In the experiment, the two parameters α and β required by HeLiEn are empirically set to 0.6 and 50, respectively.

Table 1 shows that VENTex extracted 82.6% of the tables and 85.7% of the data records, and HyLiEn extracted 79.5% of the tables and 99.7% of the data records. We remind readers that HyLiEn was not initially created to extract tables, but we find that our method can work because we consider tables to be a type of list.

We see that VENTex did extract 8 more tables than HyLiEn. We believe this is because HyLiEn does not have any notion of element distance that could be used to separate aligned but separated lists. On the contrary, in HyLiEn, if elements across separate lists are aligned and structurally similar they are merged into one list. Despite the similar table extraction performance, HyLiEn extracted many more records (*i.e.*, rows) from these tables than VENTex.

We did judge the precision score here for comparison sake. We find that, from among the 100 Web pages only 2 results contained false positives (*i.e.*, incorrect list items) resulting in 99.9% precision. VENTex remained competitive with a precision of 85.7%. Table 2 shows the full set of results on the VENTex data set. We see that HyLiEn consistently and convincingly outperforms VENTex.

Interestingly, the recall and precision values that we obtained for VENTex were actually higher than the results presented in Gatterbauer *et al.* [4] (they show precision: 81%, and recall: 68%). We are confident this difference is because we use only the first 100 Web pages of the original dataset.

Table 1. Recall for table and record extraction on the VENTex data set

	Ground truth	VENTex	HyLiEn
# tables	224	**82.6%**	79.5%
# records	6146	85.7%	**99.7%**

Table 2. Precision and Recall for record extraction on the VENTex data set

	Recall	Precision	F-Measure
VENTex	85.7%	78.0%	81.1%
HyLiEn	**99.7%**	**99.9%**	**99.4%**

7 Discussion and Concluding Remarks

We have empirically shown that by exploiting the visual regularities in Web page rendering and structural properties of pertinent elements, it is possible to accurately extract general lists from Web pages. Our approach does not require the enumeration a large set of structural or visual features, nor does it need to segment a Web page into atomic elements and use a computationally demanding process to fully discover lists.

In contrast, the computation cost of HyLiEn list extraction is actually bounded on the structural complexity of lists in a Web page. Considering the assumption that the number of lists in a Web page is many orders of magnitude smaller than the number of all the HTML tags, the computation time for HyliEn is quite small: only 4.2 seconds on average.

As a matter of future work we plan on finding a better HTML rendering engine, which can be used to speed up the execution of our method. When such an engine is found we will perform a more rigorous computational performance evaluation. With our new list finding algorithm we plan on using extracted lists to annotate and discover relationships between entities on the Web.

Part of this future work is the theory that entities (*e.g.*, people) which are commonly found together in lists are more similar than those which are not frequently found together in lists. Other interesting avenues for future work involve tasks such as indexing the Web based on lists and tables, answering queries from lists, and entity discovery and disambiguation using lists.

Acknowledgments

This research is funded by an NDSEG Fellowship to the second author. The first and fourth authors are supported by the projects "Multi-relational approach to spatial data mining" funded by the University of Bari "Aldo Moro," and the Strategic Project DIPIS (Distributed Production as Innovative Systems) funded by Apulia Region. The third and fifth authors are supported by NSF IIS-09-05215, U.S. Air Force Office of Scientific Research MURI award FA9550-08-1-0265, and by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 (NS-CTA).

References

1. Cafarella, M.J., Halevy, A., Wang, D.Z., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. Proc. VLDB Endow. 1(1), 538–549 (2008)
2. Cai, D., Yu, S., Rong Wen, J., Ying Ma, W.: Extracting content structure for web pages based on visual representation. In: Zhou, X., Zhang, Y., Orlowska, M.E. (eds.) APWeb 2003. LNCS, vol. 2642, pp. 406–417. Springer, Heidelberg (2003)
3. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: automatic data extraction from data-intensive web sites. SIGMOD, 624–624 (2002)
4. Gatterbauer, W., Bohunsky, P., Herzog, M., Krüpl, B., Pollak, B.: Towards domain-independent information extraction from web tables. In: WWW, pp. 71–80. ACM, New York (2007)
5. Gupta, R., Sarawagi, S.: Answering table augmentation queries from unstructured lists on the web. Proc. VLDB Endow. 2(1), 289–300 (2009)
6. Lerman, K., Getoor, L., Minton, S., Knoblock, C.: Using the structure of web sites for automatic segmentation of tables. SIGMOD, 119–130 (2004)
7. Lerman, K., Knoblock, C., Minton, S.: Automatic data extraction from lists and tables in web sources. In: IJCAI. AAAI Press, Menlo Park (2001)
8. Lie, H.W., Bos, B.: Cascading Style Sheets:Designing for the Web, 2nd edn. Addison-Wesley Professional, Reading (1999)
9. Liu, B., Grossman, R., Zhai, Y.: Mining data records in web pages. In: KDD, pp. 601–606. ACM Press, New York (2003)
10. Liu, W., Meng, X., Meng, W.: Vide: A vision-based approach for deep web data extraction. IEEE Trans. on Knowl. and Data Eng. 22(3), 447–460 (2010)
11. Mehta, R.R., Mitra, P., Karnick, H.: Extracting semantic structure of web documents using content and visual information. In: WWW, pp. 928–929. ACM, New York (2005)
12. Miao, G., Tatemura, J., Hsiung, W.-P., Sawires, A., Moser, L.E.: Extracting data records from the web using tag path clustering. In: WWW, pp. 981–990. ACM, New York (2009)
13. Tong, S., Dean, J.: System and methods for automatically creating lists. In: US Patent: 7350187 (March 2008)
14. Wang, R.C., Cohen, W.W.: Language-independent set expansion of named entities using the web. In: ICDM, pp. 342–350. IEEE, Washington, DC, USA (2007)
15. Weninger, T., Fumarola, F., Barber, R., Han, J., Malerba, D.: Unexpected results in automatic list extraction on the web. SIGKDD Explorations 12(2), 26–30 (2010)
16. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: WWW, pp. 76–85. ACM, New York (2005)
17. Zhai, Y., Liu, B.: Structured data extraction from the web based on partial tree alignment. IEEE Trans. on Knowl. and Data Eng. 18(12), 1614–1628 (2006)