# Unexpected Results in Automatic List Extraction on the Web

Tim Weninger  Fabio Fumarola[†]  Rick Barber
Jiawei Han  Donato Malerba[†]
University of Illinois at Urbana-Champaign
[†] Università degli Studi di Bari “Aldo Moro”

weninge1@illinois.edu, ffumarola@di.uniba.it, barber5@illinois.edu,
hanj@illinois.edu, malerba@di.uniba.it

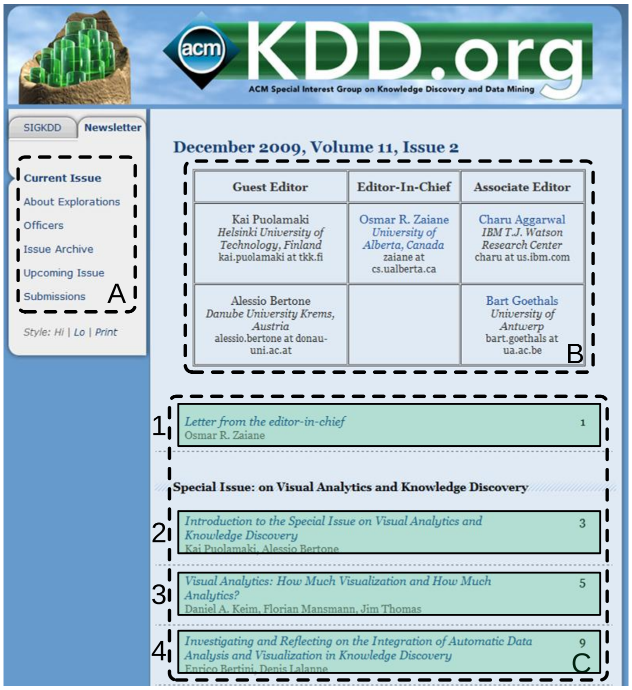
## ABSTRACT

The discovery and extraction of general lists on the Web continues to be an important problem facing the Web mining community. There have been numerous studies that claim to automatically extract structured data (*i.e.* lists, record sets, tables, etc.) from the Web for various purposes. Our own recent experiences have shown that the list-finding methods used as part of these larger frameworks do not generalize well and therefore ought to be reevaluated. This paper briefly describes some of the current approaches, and tests them on various list-pages. Based on our findings, we conclude that analyzing a Web page's DOM-structure is not sufficient for the general list finding task.

## 1. INTRODUCTION

Web information extraction can take two forms: (1) extracting information from natural language text, or (2) extracting information from structured sources. This work focuses on the latter, namely, extracting information from lists on the Web. The characteristics of Web lists vary widely. Consequently, a great variety of computational approaches have been applied to discover and extract the information embedded in lists on the Web. These existing approaches mostly rely on the underlying HTML markup and corresponding DOM structure of a Web page. 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 would be hard to find a general HTML-based tool that is sufficiently robust. In order to cope with this problem existing approaches make assumptions as to what constitutes a “list” in the Web page markup.

The reason list extraction has received so much attention is because it is an important sub-task within information extraction. Several potential use cases have been introduced in the related literature and a few publicly available products, such as Google Sets, use list finding technology, albeit at a crude level. Named entity recognition, disambiguation, and reconciliation could be enhanced by using list co-occurrences. A proper list extraction technique could also be used to annotate relationships on the Web and for discovering parallel hyperlinks. Initially, we had hoped to use an existing list extraction technique to mine link paths through Web sites, where lists would be a diverging point for the paths. Because our mining algorithm relied heavily on the proper extraction of Web lists we were surprised to find that none of the existing list extraction techniques were able to satisfy our needs in the general case. This paper highlights our specific findings and shows a naive method that unexpectedly outperforms existing methods in the general case.



December 2009, Volume 11, Issue 2

| Guest Editor | Editor-In-Chief | Associate Editor |
|---|---|---|
| Kai Puolamaki<br>*Helsinki University of Technology, Finland*<br>kai.puolamaki at tkk.fi | Osmar R. Zaiane<br>*University of Alberta, Canada*<br>zaiane at cs.ualberta.ca | Charu Aggarwal<br>*IBM T.J. Watson Research Center*<br>charu at us.ibm.com |
| Alessio Bertone<br>*Danube University Krems, Austria*<br>alessio.bertone at donau-uni.ac.at | | Bart Goethals<br>*University of Antwerp*<br>bart.goethals at ua.ac.be |

*Letter from the editor-in-chief* 1
Osmar R. Zaiane

**Special Issue: on Visual Analytics and Knowledge Discovery**

*Introduction to the Special Issue on Visual Analytics and Knowledge Discovery* 3
Kai Puolamaki, Alessio Bertone

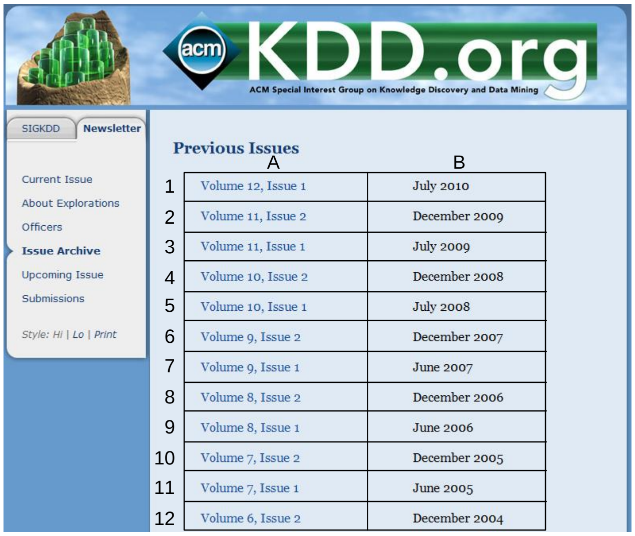
*Visual Analytics: How Much Visualization and How Much Analytics?* 5
Daniel A. Keim, Florian Mansmann, Jim Thomas

*Investigating and Reflecting on the Integration of Automatic Data Analysis and Visualization in Knowledge Discovery* 9
Enrico Bertini, Denis Lalanne

Figure 1: Web page from SIGKDD Explorations. Lettered areas marked by dotted lines are individual lists. Numbered areas in list C marked by solid lines are aligned sibling boxes.

Our observations are based on the rendered, visual Web page in addition to the underlying HTML markup. Similar to the work by Gatterbauer *et al.* [5] we had a difficult time finding an appropriate definition for a *list* on the Web. After significant discussion we settled on a definition based on that of Gatterbauer *et al.*: *A list is a series of similar data items or data records. A list can be either one-dimensional or two-dimensional; in both variants, we do not know the relationships between individual list items except for a possible ordering of the items* [5]. The two-dimensional variant is typically regarded as a *table* where the alignment of columns and rows annotates the logical relationships among groups of data items.

We regard tables on the Web to be wholly within the set of lists on the Web, that is, a table is a special type of list. To show that this is the case, Figure 2 shows a *list* of past KDD Explorations issues with two columns: A and B correspond to the issue volume/number



Previous Issues

|  | A | B |
|---|---|---|
| 1 | Volume 12, Issue 1 | July 2010 |
| 2 | Volume 11, Issue 2 | December 2009 |
| 3 | Volume 11, Issue 1 | July 2009 |
| 4 | Volume 10, Issue 2 | December 2008 |
| 5 | Volume 10, Issue 1 | July 2008 |
| 6 | Volume 9, Issue 2 | December 2007 |
| 7 | Volume 9, Issue 1 | June 2007 |
| 8 | Volume 8, Issue 2 | December 2006 |
| 9 | Volume 8, Issue 1 | June 2006 |
| 10 | Volume 7, Issue 2 | December 2005 |
| 11 | Volume 7, Issue 1 | June 2005 |
| 12 | Volume 6, Issue 2 | December 2004 |

Figure 2: Web page from SIGKDD Explorations. HTML <TABLE> tags are the structure for the list of issues (A) and publication dates (B), denoted by solid lines (1–12)

.

and date respectively. Alternatively, we could view a list as a special type of table, but we believe that our task is best expressed in general terms as lists, rather than tables.

Usually, Web lists contain items which are similar in type or content. By this observation, lists encode important information regarding the nature of their items. For example, the Web page shown in Figure 1 shows three separate lists (surrounded by dashed lines) labeled A, B and C. Looking closely we see that list A is the menu of the Web site, list B is a table depicting an editorial board, and list C is a list containing the table of contents of the current issue. Because the inherent purpose of a list is to group similar items, we can infer that the individual items in each list are related, and that the separate lists on the Web page are interrelated. Annotating these relationships is a topic for future research, and is dependent upon the list extraction task[1].

Continuing the example from Figure 1, Area C shows four list items that are vertically aligned. Note that the “Special Issue…” text should not considered to be an item in the list because it is not ‘similar’ to the other list items. Areas A and B have their own list items, but they are not explicitly outlined in order to maintain clarity.

Of course, SIGKDD Explorations is only one academic magazine, and many similar magazines/journals exist, where similarity is defined not in the content, but in the type of the publication. We assume that most publications have their own Web page with their own menu, editorial board and table of contents. If we were able to effectively and efficiently extract and integrate information from these lists, then our knowledge bases and databases could be improved. This is not a new idea, and several groups are actively working on similar and overlapping problems [2; 6; 1; 9]. Yet, most works use list extraction (in its general sense) as a means to an end. Therefore, despite the great amount of Web mining research, we show that existing approaches are inadequate in the general case.

This article highlights some of the most popular list finding algorithms and tests their ability to find general lists on the Web. The extraction algorithms we consider are from:

1. Google Sets [9]
2. WebTables [2]
3. Mining Data Records (MDR) [7]
4. World Wide Tables (WWT) [6]
5. Tag Path Clustering [8]
6. RoadRunner [4]
6. SEAL [10]
7. Visual List Extraction
8. VIsual-based Page Segmentation (VIPS) [3]
9. Visualized Element Nodes Table extraction (VENTex) [5]

An appropriate list finding test set should be large enough so that we can be statistically confident in our results, and it should contain lists of various sizes and styles. To satisfy this goal we found 107 academic departments from 6 different universities. For each department we manually extracted the faculty members from their respective faculty member directory Web pages. Table 1 lists the universities, departments and correct number of faculty members displayed on each department's Web page[2]. The goal, therefore, is to use each of the above algorithms to automatically extract the same information that we gathered manually. Of course, the faculty member extraction is only one example in the list finding task. The same methodology can be applied in any domain resulting in similar outcomes. We chose the faculty member data set because the lists can be manually labeled quickly and because the list-styles vary greatly from page to page.

Table 1: Data set for list finding experiments. Each department has a single list of faculty members to be extracted

| University | # Depts. | # People |
|---|---|---|
| Stanford | 13 | 634 |
| Illinois | 31 | 1,573 |
| MIT | 22 | 1,351 |
| Cal Berkeley | 20 | 829 |
| CMU | 12 | 560 |
| Cornell | 9 | 362 |
| Total | 107 | 5,309 |

The information we manually gathered is usually the name of the faculty member, and each of the algorithms above return different representations of the list item corresponding to the name of the faculty member. For example, the Google Sets method will return text while MDR will return an HTML subtree. Thus, when judging the results, we erred on the side of leniency in most questionable cases. Each result set shows the total number of correct lists found (*i.e.* recall), not the specific precision and recall for items in each list. Note that most of the results are binary, because either the entire list was found or it was not; in the few cases where a single item was missing or an errant item was included we generally scored the extraction as being correct.

## 2. LIST EXTRACTION CASE STUDY

This section describes the results of each algorithm and highlights the interesting and unexpected details.

### 2.1 Google Sets

Although the explicit framework for Google Sets has not been published in any academic venue, their patent filing adequately de-

[1] Rarely, the same logical Web list spans multiple Web pages; these instances are difficult to detect, and we assume that Web lists are wholly contained on a single Web page.

[2] *e.g.*, `http://cs.illinois.edu/people/faculty`

scribes the underlying framework of their algorithm. Specifically, the Google Sets framework automatically generates lists of items based on their frequent co-occurrence in lists on the Web. Of course, for this to work, Google Sets must first extract items from lists on the Web. This is done by looking for specific HTML tags, namely <UL>, <OL>, <DL> and <H1>-<H6> tags, and extracting the text in the encompassed HTML structure.

We tested the effectiveness of this list extraction technique on the data set described in Table 1. With these HTML-list patterns we found several thousand list items, many of which were menu items, and some of which contained faculty member information.

Table 2: Results for Google Sets extraction

| University | # <UL> | # Extracted | Recall |
|---|---|---|---|
| Stanford | 4 | 0 | 0% |
| Illinois | 6 | 4 | 12.90% |
| MIT | 1 | 0 | 0% |
| Cal Berkeley | 2 | 2 | 10.00% |
| CMU | 1 | 1 | 8.33% |
| Cornell | 1 | 1 | 11.11% |
| Total | 15 | 8 | 7.48% |

Table 2 shows the results for the Google Sets extraction technique. The <OL>, <DL> and <H1>-<H6> tags were not part of any faculty member list, and the <UL> tag was the structure behind 15 of the lists. Of these 15 <UL>-type lists, only 8 complete lists were extracted. This means that either (i) the faculty was divided among multiple <UL> lists or (ii) the individual items in the <UL> list contained errant information (*e.g.*, more than one faculty member, extraneous information).

Google Sets can still be effective at finding lists en masse because (if we generalize) 7.48% of all lists on the Web is still a lot of information. Yet, its extraction technique is not sensitive enough to pick up general lists. In addition, the data collected with Google Sets is biased towards certain Web structures and is not guaranteed to be a uniform sample of Web content.

## 2.2 WebTables

WebTables extracts information from certain tables on the Web in order to automatically generate schemas and consolidate the information contained in these tables into a single, integrated data source. This work is very closely related to Google's Fusion Tables project, and because the WebTables authors – Cafarella, Halevy *et al.* – work(ed) at Google we assume that the same techniques are at least partially responsible for the algorithm underlying Google Fusion Tables.

WebTables works by extracting tables from the Web based on the <TABLE> HTML-tag. From this large set of tables some heuristics are used to find only relational tables, that is, Web tables which contain columns and rows like in a database relation. The authors estimate that 1.1% of the <TABLE>'s on the Web actually represent relational information.

For our purposes, we assume that the directories of faculty members contain the appropriate form, and, as such, in our tests we look for table rows that contain a single faculty member. If a whole table contains table rows of one faculty member each (ignoring table headers), then the table is a positive result.

Table 3 shows the results for the WebTables extraction technique. Although, the <TABLE> tag was the structure behind 70 lists (65%), only 38.32% complete lists were extracted. The errors were similar in nature to the errors experienced by the Google Sets extraction: either the rows contained multiple faculty members each or the table rows contained extraneous information.

Table 3: Results for WebTables extraction

| University | # <TABLE> | # Extracted | Recall |
|---|---|---|---|
| Stanford | 9 | 6 | 46.25% |
| Illinois | 16 | 12 | 38.71% |
| MIT | 15 | 7 | 31.82% |
| Cal Berkeley | 15 | 9 | 45.00% |
| CMU | 7 | 3 | 25.00% |
| Cornell | 8 | 4 | 44.44% |
| Total | 70 | 41 | 38.32% |

### 2.2.1 WWT

World Wide Tables (WWT) is an effort similar to that of WebTables. The list extraction in WWT is described as using "…a group of heuristics…" [6]. While these heuristics are not explicitly mentioned, we infer that they are similar to the techniques of Google Sets and WebTables in that they use simple HTML-tag patterns for extraction.

In the Google Sets and WebTables results, we find that strict HTML-tag matching works in only a very limited set of instances. While these results are not necessarily unexpected, they provide motivation for more robust techniques for extracting lists on the Web.

## 2.3 MDR

One such "robust" technique is the Mining Data Records (MDR) algorithm [7]. MDR aims to extract records from Web pages by mining the page's DOM-structure. MDR is based on two key observations. First, a set of *data records* (*i.e.*, list items) typically appear in a coherent region of a Web page and they are formatted using similar HTML tags and patterns. This set of records is called a *data region*. Second, the records in a data region are typically rooted in a single parent node.

One major goal underlying the construction and execution of MDR is that it does *not* make any assumptions about the type of HTML-tags used to construct the data records. That is, a <UL>-tag is no more likely to indicate a list than a <DIV>-tag.

In order to operate effectively, MDR assumes that the individual records in a data region contain a robust structure which repeats regularly throughout the data region. This assumption is correct when a single data record contains a sufficiently-large sub-structure like in Area C in Figure 1, which contains title and author information for each record. However, this assumption fails when the sub-structure of list elements are small like in Area A in Figure 1, which only contains a link.

Table 4: Results for MDR extraction

| University | # Extracted | Recall |
|---|---|---|
| Stanford | 7 | 53.85% |
| Illinois | 13 | 40.63% |
| MIT | 6 | 26.09% |
| Cal Berkeley | 5 | 27.78% |
| CMU | 3 | 25.00% |
| Cornell | 0 | 0% |
| Total | 34 | 31.78% |

We tested the MDR algorithm under the same conditions as the above experiments. Table 4 shows an overall recall of 31.78%. We note that the experiments in the original MDR paper obtains nearly perfect extraction precision and recall; we assume that this is because the nature of the lists in the experiments conformed to the data record style. We see in Table 4 that MDR cannot be effectively used in *general* list extraction.

Ancillary experiments show that when MDR is applied to a Web page that does not contain lists – a news article, for example – data records are sometimes errantly extracted from nuances in the HTML structure. These "nuances" can typically be attributed to the appearance of regular/repeating structures in a Web page. For an example, consider the typical Wikipedia article that contains a large list of languages on the left-hand side, a table of contents list, an info-box on the right-hand side, and wiki-links throughout the page. We find that MDR usually correctly extracts the infobox, but almost always misses the list of languages and the table of contents.

### *2.3.1 Tag Path Clustering*

The Tag Path Clustering (TPC) algorithm [8] is quite similar to MDR. TPC looks for frequently reoccurring patterns in the DOM structure of a Web page in order to detect and extract data records.

Although we were unsuccessful in our attempt to re-implement TPC, we believe that TPC suffers from the same generalization problem as MDR because the two algorithms share similar assumptions.

## 2.4 Wrapper Generation

The task of automatically learning rules for Web extraction is commonly referred to as "wrapper generation". In wrapper generation techniques, two or more Web pages are compared in order to find common DOM-structures. Information found within these learned-structures can be extracted and stored. Here we present two wrapper-based techniques: RoadRunner and SEAL.

### *2.4.1 RoadRunner*

The RoadRunner algorithm [4] is able to quickly learn the structure of a Web site in order to automatically generate wrappers. These wrappers have been shown to be effective at extracting content from a learned Web template. The Flint system [1] shows that this type of approach can be used to extract structurally repeating objects. However, it is unlikely that a wrapper can be trained to extract general lists from the Web.

In order to test RoadRunner on our data set we would need to train a wrapper on each individual Web page because each department directory contains a different template. This, for all practical purposes, defeats the point of automatic list extraction, and therefore we were unable to provide any experimental evaluation of RoadRunner.

### *2.4.2 SEAL*

Similar wrapper generation work was performed by Cohen and Wang for the Set Expander for Any Language (SEAL) project [10]. SEAL is a system similar to Google Sets, but its extraction technique is essentially a wrapper learner. However, SEAL's wrapper learner is quite dissimilar from the wrapper learner in Roadrunner, in that, SEAL requires one or (preferably) more example texts in order to learn extraction rules. SEAL, therefore, learns wrappers and performs list extraction at query time resulting in a rather large result latency[3]. Because the list extraction phase in SEAL requires example texts, we were unable to provide any experimental evaluation of SEAL.

## 2.5 Visual List Extraction

Due to the unexpected results from the above list extraction methods we implemented a naive list extraction method which explores the visual alignment of objects in a rendered Web page. The result of the Web page rendering process can be regarded as a set of *boxes*. Each rendered box in the resulting page has a position and size, and can either contain content (*i.e.*, text or images) or more boxes. Generally, there is a box created for each DOM element[4].

Sometimes two or more sibling boxes (boxes with the same parent-box) are aligned so as to appear as a list. We therefore define a *Web list* to be any set of sibling boxes which are visually aligned on a rendered Web page. This alignment can occur via the x-axis (*i.e.*, a vertical list), the y-axis (*i.e.*, horizontal lists), or in a tiled manner (*i.e.*, aligned vertically and horizontally).

Our visual list extraction algorithm uses Java's Swing HTMLEditorKit library which is capable of rendering HTML source code into Swing components. Unfortunately, this library is quite old and is only compatible with HTML 3.2 standards. We found that the HTMLEditorKit library frequently rendered Web pages differently than modern browsers, but the resemblance was close enough for our simple extraction attempt.

The next, and most difficult step, was to reconcile the rendered Swing components with the appropriate DOM-tag. After mapping each DOM-tag to the appropriate Swing component, we were able to find (X,Y) coordinates for each DOM-tag. DOM tags which are found to be in either vertical or horizontal alignment and contain the same DOM-parent are considered to be items in a list.

Like MDR and TPC, our visual list extraction method is tag-agnostic and therefore does not look for specific HTML cues like <TABLE> or <UL> tags. One potential drawback of this method is that all of the content of a page needs to be downloaded in order for the Web page to be rendered, including images, style sheets, etc. This will certainly cause an increase in total execution time relative to the DOM-only methods.

Table 5: Results for Visual List Extraction

| University | # Extracted | Recall |
|---|---|---|
| Stanford | 8 | 61.54% |
| Illinois | 16 | 50.00% |
| MIT | 12 | 60.00% |
| Cal Berkeley | 11 | 61.11% |
| CMU | 7 | 58.33% |
| Cornell | 5 | 55.55% |
| Total | 59 | 55.14% |

We tested the visual list extraction method in the same manner as the previous algorithms. Table 5 shows that, although the visual method uses a sub-standard rendering library and a naive alignment heuristic, it extracts 55.14% of the faculty lists. This result is a 74% increase over the MDR method and an obvious improvement over the tag-specific extraction methods of Google Sets and WebTables.

### *2.5.1 VIPS*

VIsion-based Page Segmentation (VIPS) [3] is an existing method similar to our visual link extraction technique. A compiled implementation of the algorithm is available on the Web, but our testing found that VIPS is better suited for detecting blocks of data on Web pages rather than detecting lists and list elements. For VIPS

[3] A demo of SEAL at `http://boowa.com/` usually takes 1 minute per query

[4] For a demo of Web page boxes download the Google Chrome browser, open any Web page, right click anywhere on the Web page, and select "Inspect element" from the menu. Moving the cursor over each HTML element will highlight the respective *box* on the rendered Web page.

to work properly in this setting, its parameters must be manually tuned to each individual Web page. Because of this we were unable to collect results from the VIPS algorithm in a consistent manner. Default parameter settings returned very poor list finding results.

### 2.5.2 *VENTex*

The Visualized Element Nodes Table extraction (VENTex) algorithm is able to to find tables by inspecting the rendered Web page and the corresponding DOM tree [5]. VENTex uses a comprehensive list of heuristics to extract and interpret tables in the Web. So, while this line of research can extract tables with a reported precision and recall of 68% and 81% respectively, we are interested in the broader extraction of Web lists in general. Because of the constrained notion of what constitutes a table on the Web, the heuristics proposed in VENTex are not likely to effectively extract general lists from the Web.

## 3. DISCUSSION

The results shown here are unexpected yet promising. Unexpected because we had hoped that existing techniques would be able to effectively handle list finding. Our results were especially unexpected because our data set of departmental faculty directories contained lists humans can identify with ease.

Table 6: Cumulative Result Comparison

| Algorithm | # Extracted | Recall |
|---|---|---|
| Google Sets | 8 | 7.48% |
| WebTables | 41 | 38.32% |
| MDR | 34 | 31.78% |
| Visual Ext. | 59 | 55.14% |

We initially developed the visual list extraction technique as a naive approach and expected it to perform far worse than existing techniques. As we show in Table 6, the visual list extraction performed the best on average. Tables 2-5 also show that the visual list extraction consistently outperformed existing techniques across all universities.

In fairness, the Google Sets, WebTables and WWT techniques were designed with a different goal in mind. With these approaches the system assumes that it will not obtain a very high recall. Instead, the objective was to gather information from an extremely large data set. Therefore, a 7% list coverage (in the case of the Google Sets extraction technique) of the entire World Wide Web is still an extremely large amount of information. But if, by using a visual extraction technique, we could obtain 55% of the lists on the Web, then our knowledge bases could be vastly improved, and our extracted collection would not exhibit the sampling bias of these current methods.

These experiments contain a set of very different Web lists; therefore, we believe our results should be viewed as representative of the general Web. In our specific domain, by extracting faculty members from the Web we may be able to create a “who’s-who” of researchers at any number of topical or hierarchical resolutions.

We would like to reiterate that our working hypothesis was that an existing technique would work. We invested a lot of time trying to make the algorithms described above work for our Web mining purposes. The list extraction method was never a main goal of our work, but due to the surprising results we initially received, we felt compelled to reevaluate these methods and present them here.

## 4. CONCLUSIONS

In this paper we argue that there is a need for a general list extraction technique, and our experiments show that existing techniques are not individually capable of filling this need. Most available list extraction tools are still based on HTML only. HTML was initially designed for rendering purposes and not for information structuring (as XML). A list can be rendered in several ways in HTML, and it would be hard to find a general HTML-based tool which is sufficiently robust. On the other hand, our naive visual list extraction method shows promising results that, if rigorously defined and implemented, could be used for general list detection and extraction.

Based on our experience and learned intuition, we believe that the most effective list extraction approach may be found in the intersection of DOM pattern finding techniques like MDR and a visual extraction technique (more robust than the one described above). Furthermore, we ask interested members of the community to explore these techniques in order to advance this line of research.

## 5. ACKNOWLEDGMENTS

We thank Marina Danilevsky for her help on this project. This research is funded by an NDSEG Fellowship to the first author, NSF grant IIS-09-05215 to the third and fourth authors. The second and fifth authors are supported by the University of Bari “Aldo Moro,” and the Project DIPIS funded by Apulia Region.

## 6. REFERENCES

[1] L. Blanco, V. Crescenzi, P. Merialdo, and P. Papotti. Flint: Google-basing the web. In *EDBT*, volume 261, pages 720–724. ACM, 2008.

[2] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, pages 538–549, 2008.

[3] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Extracting content structure for web pages based on visual representation. In *APWeb*, pages 406–417, 2003.

[4] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: automatic data extraction from data-intensive web sites. In *SIGMOD*, pages 624–624, New York, NY, USA, 2002. ACM.

[5] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krupl, and B. Pollak. Towards domain independent information extraction from web tables. In *WWW*, 2007.

[6] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *PVLDB*, pages 289–300, 2009.

[7] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *KDD*, pages 601–606, New York, NY, USA, 2003. ACM.

[8] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. In *WWW*, pages 981–990, 2009.

[9] S. Tong and J. Dean. System and methods for automatically creating lists. US Patent: 7350187, Mar 2008.

[10] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM*, 2007.