Automatically Extracting Complex Data Structures from the Web

Laura Fontán, Rafael López-García, Manuel Álvarez and Alberto Pan
Department of Information and Communication Technologies, University of A Coruña, Facultade de Informática - Campus de Elviña S/N, A Coruña, Spain
laura.fontan@udc.es, rafael.lopez@udc.es, manuel.alvarez@udc.es, apan@udc.es

Keywords: Information Retrieval, Automatic Web Data Extraction.

Abstract: This paper presents a new technique for detecting and extracting lists of structured records from Web pages. With respect to most of the state-of-the-art systems, our approach is capable of detecting nested data structures (sublists) and it also incorporates some heuristics to delete unwanted content such as banners and navigation menus from the data region. This article also describes the experiments we have performed to validate the system. The precision and recall we have obtained in our tests surpass 90%.

1 INTRODUCTION

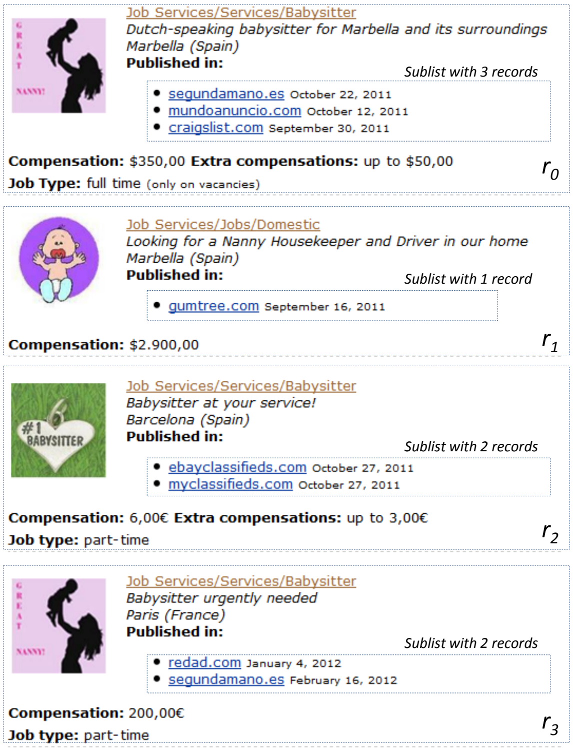
Many websites provide an underlying database containing structured data. However, in most of the cases, this information is only offered in HTML format, which makes it difficult for programs to access it. Fortunately, the layout used by websites is often regular, allowing us to infer the data structure with an acceptable accuracy. An example of list with structured records is shown in Figure 1.

Classic automatic Web data extraction systems use the so-called supervised approach (Zhai and Liu, 2005; Raposo et al., 2007), in which a human administrator configures a Web automation process (‘wrapper’) for each data extraction process in every target website. The process must also be updated every time the website changes. Although this approach can produce very accurate results, it does not scale for a high number of websites.

On the other hand, we have the unsupervised approach, in which the extraction process is fully automatic. Although this approach tends to produce less accurate results, it can scale to large numbers of websites. Most systems using this approach present two important problems:

1. Inability to deal with nested data structures.
2. Lack of measures to detect unwanted content such as navigation menus and banners inside the data region.

In this paper, we present an unsupervised Web data extraction system capable of obtaining data with high precission and recall even in pages presenting the aforementioned complexities.



Job Services/Services/Babysitter
Dutch-speaking babysitter for Marbella and its surroundings
Marbella (Spain)
Published in:

- segundamano.es October 22, 2011
- mundoanuncio.com October 12, 2011
- craigslist.com September 30, 2011

Compensation: $350,00 **Extra compensations:** up to $50,00
Job Type: full time (only on vacancies)

Job Services/Jobs/Domestic
Looking for a Nanny Housekeeper and Driver in our home
Marbella (Spain)
Published in:

- gumtree.com September 16, 2011

Compensation: $2.900,00

Job Services/Services/Babysitter
Babysitter at your service!
Barcelona (Spain)
Published in:

- ebayclassifieds.com October 27, 2011
- myclassifieds.com October 27, 2011

Compensation: 6,00€ **Extra compensations:** up to 3,00€
Job type: part-time

Job Services/Services/Babysitter
Babysitter urgently needed
Paris (France)
Published in:

- redad.com January 4, 2012
- segundamano.es February 16, 2012

Compensation: 200,00€
Job type: part-time

Figure 1: An HTML page with a list of nested records.

2. RELATED WORK

RoadRunner (Crescenzi et al., 2001) and ExAlg (Arasu & Garcia-Molina, 2003) were some of the first web data extraction systems using the unsupervised approach. They share an important limitation: they need multiple pages with the same template in order to infer the extraction rules to obtain the data. In addition, RoadRunner does not support disjunctions in the schema.

DEPTA (Zhai & Liu, 2006) needs only one page to detect and extract its structured records. It analyses the visual layout of the page and calculates tree-edit distances between elements. However, it assumes features that are not always verified in real websites, such as that all the records are composed by the same number of subtrees and the space that separates two data records is bigger than the one that separates two data values from the same record.

Álvarez et al. (2008) present a system that reuses some of the models presented in ExAlg and extracts data from a single page. Their method does not present the limitations we have found in DEPTA. However, neither this system nor DEPTA support extracting data from nested structures. Their system also addresses the problem of cleaning unwanted data, but it shows important limitations. Our system uses the one by Álvarez et al. (2008) as a starting point, so section 3 studies it more thoroughly.

Miao et al. (2009) introduce a system that is capable of detecting nested structures and non-consecutive records, but it simply identifies the data records without extracting the individual data fields of each record.

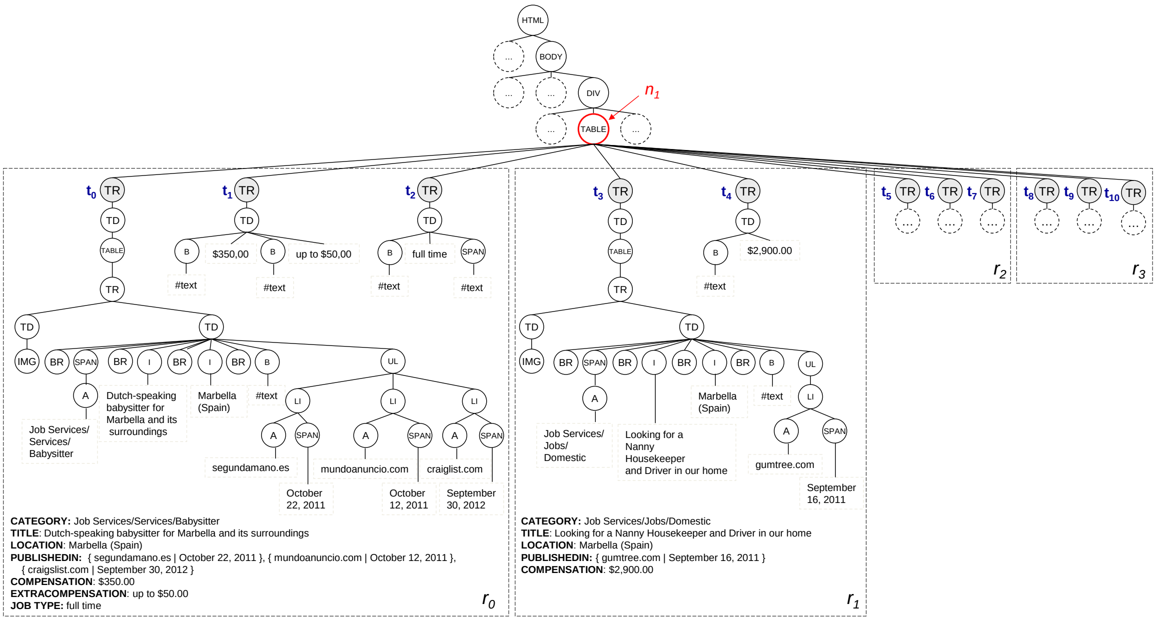
G-STM (Jindal and Liu, 2010) is an extension to the Simple Tree Matching algorithm. It is capable of dealing with nested sublists, but their method may identify non-lists as lists when subtrees are not deep enough. In turn, our system includes additional heuristics for these cases.

3 OVERVIEW OF THE EXISTING SOLUTION

The purpose of our algorithm is to extract a list of data records from an HTML page such as the one shown in Figure 1. Our approach is based on a state-of-the-art solution (Alvarez et al, 2008), summarized in this section, and we have extended it to process nested sublists and other complexities (section 4).

The system is based on the DOM tree representation of HTML pages. Starting from this tree, a three-step algorithm is applied to extract the data. Figure 2 shows an excerpt of the DOM tree corresponding to our example page.

The first step is finding the dominant list of records of the page. This is equivalent to finding the



CATEGORY: Job Services/Services/Babysitter
TITLE: Dutch-speaking babysitter for Marbella and its surroundings
LOCATION: Marbella (Spain)
PUBLISHEDIN: { segundamano.es | October 22, 2011 }, { mundoanuncio.com | October 12, 2011 }, { craigslist.com | September 30, 2012 }
COMPENSATION: $350.00
EXTRACOMPENSATION: up to $50.00
JOB TYPE: full time

CATEGORY: Job Services/Jobs/Domestic
TITLE: Looking for a Nanny Housekeeper and Driver in our home
LOCATION: Marbella (Spain)
PUBLISHEDIN: { gumtree.com | September 16, 2011 }
COMPENSATION: $2,900.00

Figure 2: An excerpt of the DOM tree of the page in Figure 1.

common parent node of the sibling subtrees which form the data records. In our example of Figure 2, the node we should discover is n_1. The subtree with that node as root is called the *data region*.

The second step is dividing the data region into records. We assume that each record is composed of one or several consecutive subtrees of the root node of the data region. For instance, in Figure 2, we need to detect that the first record is formed by the subtrees labelled as *t0*, *t1* and *t2*, the second record is formed by the subtrees labelled *t3* and *t4*, etc.

The key idea in this step is to choose the record division that maximizes the similarity between records. The similarity measure used is based on serializing the subtrees as strings and then applying string edit-distance techniques (Levenstein, 1966).

The main problem to apply this idea is that the number of candidate divisions is 2^{n-1}, where n is the number of subtrees. In most real Web pages, this number is too high to compute the similarities for all the possible divisions. Therefore, we first need to generate a set of candidate divisions.

The method for choosing the candidate divisions starts by clustering all the subtrees in the data region according to their similarity. Then, we assign an identifier to each cluster we have generated and we build a sequence by listing the subtrees in the data region in order, representing each subtree with the identifier of the cluster it belongs to (see Figure 3). The string will tend to be formed by a repetitive sequence of cluster identifiers, with each repetition corresponding to a data record. Since there may be optional data fields in the extracted records, the sequence for one record may be slightly different from the sequence corresponding to another record. Nevertheless, we will assume that all records either start or end with a subtree belonging to the same cluster (i.e. all the data records always either start or end in a similar way). This is built on the heuristic that in most Web sites, records are visually delimited in an unambiguous manner to improve clarity.

This process reduces the number of candidate divisions to $1+2k$, where k is the number of clusters.

Figure 3 shows the subtrees of Figure 2 that have been chosen for each record r_0-r_3 of our example by applying this process and choosing the subtree with the highest similarity between records.

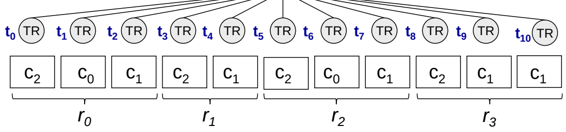


Figure 3: Record division for the page in Figure 1.

The third step is extracting the attributes of the data records. This point involves two stages: (1) transforming each record into a string and (2) applying string alignment techniques to each record in order to identify its attributes. Variations of the "center star" algorithm (Gonnet et al., 1992) are used to solve this problem. The key idea is that the aligned variable values in several records represent the different values of the same field in different records. Figure 4 shows an excerpt of the alignment between the strings that represent the records of our example. In this excerpt, it can be seen how the last data fields of each record are aligned (fields that correspond to the attributes *COMPENSATION*, *EXTRACOMPENSATION* and *JOB_TYPE*).

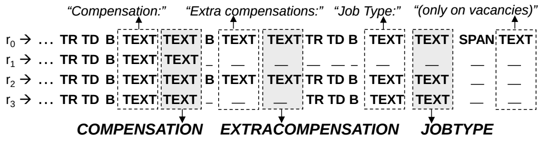

"Compensation:"
"Extra compensations:"
"Job Type:"
"(only on vacancies)"
r0 → … TR TD B TEXT TEXT B TEXT TEXT TR TD B TEXT TEXT SPAN TEXT
r1 → … TR TD B TEXT TEXT _ — — — — _ — — — —
r2 → … TR TD B TEXT TEXT B TEXT TEXT TR TD B TEXT TEXT — —
r3 → … TR TD B TEXT TEXT _ — — TR TD B TEXT TEXT — —
COMPENSATION
EXTRACOMPENSATION
JOBTYPE

Figure 4: Alignment between records of Figure 1.

4 IMPROVEMENTS TO THE EXISTING SOLUTION

The method described in the previous section has several limitations. For instance, it is unable to identify the sublists marked in our example of Figure 1. In this section, we will describe the improvements we have incorporated to deal with these limitations.

The main idea behind our algorithm is very simple: each record from the list identified by the above method can be considered as a reduced HTML page by simply adding a root node to their subtrees. Then we can simply execute the method again on these "reduced pages" to find sublists inside them. For instance, applying our algorithm on the first record of Figure 1 (*r0*) will correctly identify the 3-record sublist inside it. In the cases in which sublists are found, we can keep applying the method recursively for each record. Unfortunately, this simple method has several important drawbacks:

1. Sublists with very few records will not be identified. Our method for finding lists is based on finding records with high inter-similarity, but, for instance, in record *r1* of Figure 1, the sublist contains only one record, so the method cannot work directly.
2. As the "pages" used to search for lists become more and more reduced, lists tend to be shorter and their elements tend to have fewer

fields. In this situation, the list identification process will have less information, so we need additional heuristics to maintain the accuracy.

Section 4.1 discusses these issues. The problem of removing unwanted content (ads, pagination controls, etc.) contained inside the data region is discussed in section 4.2.

The results of the algorithm are stored in a multilevel map due to the hierarchical nature of the data model. Figure 5 shows the resulting multilevel structure from applying our algorithm on the Figure 1 example.

4.1 Processing nested structures

The biggest challenge in detecting nested data structures inside records is to distinguish sublists from sequences of data fields. For instance, in the Web page of Figure 1 the challenge is recognising that the pairs (URL, date) present in each ad form a sublist, while the remaining parts of the ad (title, description, compensation, etc.) are individual data fields of the top-level records.

Our method is based on the following observations about the nature of lists and records:

1. By definition, the data records of a list are of the same type, while the data fields of an individual record may be of different types.
2. The number of elements in a list can be very variable, while the number of data fields in different records of the same type tends to be fixed. The variability in the number of fields is usually more limited than the variability in the number of elements of a list.

From the first feature, we conjecture that the different records of a list will be formatted in the same (or very similar) way in the page, while data fields of an individual record will tend to show some formatting differences. Therefore, we set an inter-record similarity threshold for candidate sublists.

From the second characteristic, we conjecture that the corresponding sublists found in different records of the top-level list should tend to have a different number of elements.

These conjectures are combined to determine if a candidate sublist should pass the filter. For instance, in the page of Figure 1, we notice how the number of pairs (url, date) is different among the records in the main list (there are three pairs in *r0*, one in *r1* and two in *r2* and *r3*). In addition, the formatting of all pairs is identical, and, therefore, it will result in a very high inter-record similarity. Hence, this candidate sublist will be considered as correct.

The way to combine these heuristics depends on the average number of data fields in the records of the candidate sublist. If this number is medium-high, the inter-record similarity will be used as crucial source of evidence. However, if this number is low (e.g. below 2), variability will be more important.

The rationale behind this is that, on the one hand, when the elements making up the candidate sublist have many fields, inter-record similarity provides enough evidence to conclude that the sublist is correct. On the other hand, when the records of the candidate sublist consist of very few elements (e.g.: a single text or link), then the inter-record similarity gives us less information and variability should also be considered.

For instance, consider the case of a simple

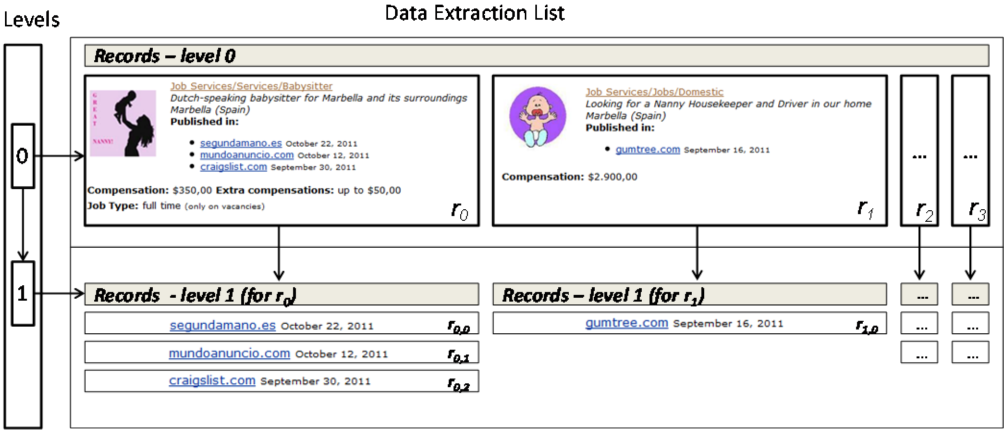

Levels
Data Extraction List
Records – level 0
0
1
Job Services/Services/Babysitter
Dutch-speaking babysitter for Marbella and its surroundings
Marbella (Spain)
Published in:
segundamano.es October 22, 2011
mundoanuncio.com October 12, 2011
craigslist.com September 30, 2011
Compensation: $350,00 Extra compensations: up to $50,00
Job Type: full time (only on vacancies)
r_0
Job Services/Jobs/Domestic
Looking for a Nanny Housekeeper and Driver in our home
Marbella (Spain)
Published in:
gumtree.com September 16, 2011
Compensation: $2.900,00
r_1
...
r_2
...
r_3
Records - level 1 (for r_0)
segundamano.es October 22, 2011 $r_{0,0}$
mundoanuncio.com October 12, 2011 $r_{0,1}$
craigslist.com September 30, 2011 $r_{0,2}$
Records – level 1 (for r_1)
gumtree.com September 16, 2011 $r_{1,0}$
...

Figure 5: Result of the algorithm applied to the example.

HTML table representing one record in each row, one data field in each column and where all the column values are single text elements. Considering only inter-record similarity would incorrectly identify the sequence of columns in each row as a sublist, since all the data fields are formatted identically. However, the variability measure will discard those candidate sublists, since all of them have exactly the same number of records.

Some refinements are still needed by this method in order to deal with some special cases. These cases are studied in subsections 4.1.1 and 4.1.2.

4.1.1 Sublists that contain a low number of elements

When sublists contain a low number of elements, the method for finding lists will not detect them. For instance, the record *r1* of Figure 1 contains a sublist with only one element, so it will not be detected.

However, the existence of a sublist in other records of the same list (or in the records of another list at the same level) can help to detect these situations. In our example, finding sublists in records *r0*, *r2* and *r3* suggests that *r1* may also contain a sublist of the same type. For this reason, when we confirm the existence of sublists in a certain list, we inspect again those records where no sublists have been found and we align them with the remaining ones using the alignment process which has been described in section 3 and illustrated in Figure 4. If one record in which sublists have not been found does contain some portions aligning with the sublists found in other records, those portions will be considered sublists too.

4.1.2 Use of label information

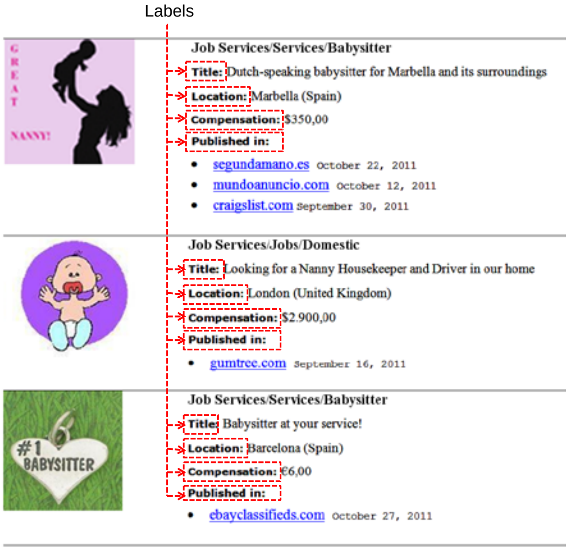
As we have explained in the previous sections, one of the assumptions of our approach is that the records of a list will be formatted similarly in the page, while the different data fields of a record will tend to show some formatting differences.

However, there are also cases like the one shown in Figure 6, in which all the data fields of a record are formatted in the same way. This could cause the system to incorrectly identify the sequence of data fields called “title”, “location”, and “compensation” as elements of a sublist.

Fortunately, it is very frequent in these cases that some or all of the data fields are prefixed by labels identifying them in order to avoid confusions. For instance, in Figure 6, the data fields called “title”, “location”, and “compensation” are prefixed by their corresponding labels.

Labels can be easily detected by the aligning process: when a data field always appears prefixed by the same text in all records of the list, we consider that text a label for the field.

Our hypothesis in this case is that labels are indicative of the existence of data fields. A sequence containing (label, value) pairs is very improbable to be a list because all the elements of a list are of the same type and, therefore, it is not needed any label to distinguish between them.



Job Services/Services/Babysitter

Title: Dutch-speaking babysitter for Marbella and its surroundings

Location: Marbella (Spain)

Compensation: $350,00

Published in:

- segundamano.es October 22, 2011
- mundoanuncio.com October 12, 2011
- craigslist.com September 30, 2011

Job Services/Jobs/Domestic

Title: Looking for a Nanny Housekeeper and Driver in our home

Location: London (United Kingdom)

Compensation: $2.900,00

Published in:

- gumtree.com September 16, 2011

Job Services/Services/Babysitter

Title: Babysitter at your service!

Location: Barcelona (Spain)

Compensation: €6,00

Published in:

- ebayclassifieds.com October 27, 2011

Figure 6: Using labels to identify data fields.

4.2 Removing unwanted data

The first step in extracting lists of data records is finding the common parent node of the sibling subtrees forming the data records.

In some websites, the data region also contains other tokens that do not belong to the actual data. Typical examples are banners, navigation links, etc. These items are normally located at the beginning or at the end of the data region. We use three heuristics to deal with this problem, although the first one had already been included in the system by Álvarez et al. (2008):

1. Remove leading and trailing unaligned tokens: since all the data records always either start or end in the same way, we can conclude that tokens at the beginning and at the end of the data region that do not align with anything else are candidates to be removed.
2. Remove tokens that only align in the first and last data record: sometimes unwanted data is repeated in the first and last records of the data

region. Navigation links to other result pages are an example of this.

3. Remove leading and trailing records with low similarity: if a record in the data region has low similarity with the other records, it will likely contain unwanted information.

5 ASSESSMENT

We have chosen 170 well-known websites in 5 different application domains for our experiments: classified ads (40), job sites (31), social networks (24), online shops (5) and bookstores (70).

We have executed one query in each website and collected the first page containing the list of results. We have manually selected the queries to guarantee that the collection includes pages having a very variable number of results (from two or three to dozens).

We have calculated the precision and recall of the system in two different ways: (1) considering only the records that have been extracted perfectly and (2) considering partially correct records too (records where most of the data fields are correctly identified, but where some attributes are missing).

As it can be seen in Table 1, our algorithm achieves more than 86% for both precision and recall. If we consider partially correct records, the results are close to 93%.

Table 1: Results of the empirical evaluation.

	With correct records only	With partially correct records
Records to extract	2,416	2,416
Extracted records	2,427	2,427
Correctly extracted	2,089	2,247
Precision	86.07%	92.58%
Recall	86.47%	93.00%

6 CONCLUSIONS

This paper has presented a system for automatic Web data extraction. The system detects and extracts lists of structured records in Web pages following an unsupervised approach. It is capable of obtaining data from nested structures (sublists) and it also includes heuristics to remove unwanted and unimportant data such as banners, navigation links and menus.

The system has been tested in a high number of real websites, reaching precision and recall values higher than 86% when only correctly extracted records are considered and values close to 93% when partially correct records are considered too.

ACKNOWLEDGEMENTS

This paper has been supported by the SCIIMS Project, financed by the 7th Framework Program of the EU (code FP7-SEC-2007-1-218223).

REFERENCES

Álvarez, M. Pan, A., Raposo, J., Bellas, F., Cacheda, F., 2008. *Extracting lists of data records from semi-structured web pages. In Data and Knowledge Engineering, vol. 64, num 2.*

Arasu, A., Garcia-Molina, H., 2003. Extracting structured data from web pages. In *Proceedings of the ACM SIGMOD International Conference on Management Data.*

Crescenzi, V., Mecca, G., Merialdo, P., 2001. ROADRUNNER: towards automatic data extraction from large web sites. In *Proceedings of the 2001 International VLDB Conference, pp. 109-118.*

Gonnet, G.H., Baeza-Yates, R., Snider, T. 1992. New Indices for Text Pat Trees and Pat Arrays. In *Information Retrieval: Data Structures and Algorithms.* Prentice Hall.

Jindal, N., Liu, B., 2010. A Generalizad Tree Matching Algorithm Considering Nested Lists for Web Data Extraction. In *Proceedings of SIAM International Conference on Data Mining 2010, pp.930-941.*

Levenstein, V.I., 1966. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady, 10, pp. 707-710.*

Miao, G., Tatemura, J., Hsiung, W.P., Sawires, A., Moser, L., 2009. Extracting Data Records from the Web Using Tag Path Clustering. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09), pp. 981-990.*

Raposo, J., Pan, A., Alvarez, M., Hidalgo, J., 2007. Automatically maintaining wrappers for web sources. In *Data and Knowledge Engineering Journal, vol. 61, num. 2, pp. 331-358.*

Zhai, Y., Liu, B., 2005. Extracting web data using instance-based learning. In *Proceedings of Web Information Systems Engineering Conference (WISE), pp. 318-331.*

Zhai, Y., Liu, B., 2006. Structured data extraction from the web based on partial tree alignment. In *IEEE Transactions on Knowledge and Data Engineering, vol. 18, num. 12, pp. 1614-1628.*